AOS-CX 10.11 REST v1 API Guide

All AOS-CX Series Switches

a Hewlett Packard
Enterprise company

Published: February 2023
Edition: 2

Copyright Information

© Copyright 2023 Hewlett Packard Enterprise Development LP.

Open Source Code

This product includes code licensed under the GNU General Public License, the GNU Lesser General Public License, and/or certain other open source licenses. A complete machine-readable copy of the source code corresponding to such code is available upon request. This offer is valid to anyone in receipt of this information and shall expire three years following the date of the final distribution of this product version by Hewlett Packard Enterprise Company. To obtain such source code, send a check or money order in the amount of US $10.00 to:

Hewlett Packard Enterprise Company
6280 America Center Drive
San Jose, CA 95002
USA

Notices

The information contained herein is subject to change without notice. The only warranties for Hewlett Packard Enterprise products and services are set forth in the express warranty statements accompanying such products and services. Nothing herein should be construed as constituting an additional warranty. Hewlett Packard Enterprise shall not be liable for technical or editorial errors or omissions contained herein.

Confidential computer software. Valid license from Hewlett Packard Enterprise required for possession, use, or copying. Consistent with FAR 12.211 and 12.212, Commercial Computer Software, Computer Software Documentation, and Technical Data for Commercial Items are licensed to the U.S. Government under vendor's standard commercial license.

Links to third-party websites take you outside the Hewlett Packard Enterprise website. Hewlett Packard Enterprise has no control over and is not responsible for information outside the Hewlett Packard Enterprise website.

Contents

Chapter 1
About this document

This document describes features of the AOS-CX network operating system. It is intended for administrators responsible for installing, configuring, and managing Aruba switches on a network.

Applicable products

This document applies to the following products:

- Aruba 4100i Switch Series (JL817A, JL818A)
- Aruba 6000 Switch Series (R8N85A, R8N86A, R8N87A, R8N88A, R8N89A, R9Y03A)
- Aruba 6100 Switch Series (JL675A, JL676A, JL677A, JL678A, JL679A)
- Aruba 6200 Switch Series (JL724A, JL725A, JL726A, JL727A, JL728A, R8Q67A, R8Q68A, R8Q69A, R8Q70A, R8Q71A, R8V08A, R8V09A, R8V10A, R8V11A, R8V12A, R8Q72A)
- Aruba 6300 Switch Series (JL658A, JL659A, JL660A, JL661A, JL662A, JL663A, JL664A, JL665A, JL666A, JL667A, JL668A, JL762A, R8S89A, R8S90A, R8S91A, R8S92A)
- Aruba 6400 Switch Series (R0X31A, R0X38B, R0X38C, R0X39B, R0X39C, R0X40B, R0X40C, R0X41A, R0X41C, R0X42A, R0X42C, R0X43A, R0X43C, R0X44A, R0X44C, R0X45A, R0X45C, R0X26A, R0X27A, JL741A)
- Aruba 8320 Switch Series (JL479A, JL579A, JL581A)
- Aruba 8325 Switch Series (JL624A, JL625A, JL626A, JL627A)
- Aruba 8360 Switch Series (JL700A, JL701A, JL702A, JL703A, JL706A, JL707A, JL708A, JL709A, JL710A, JL711A, JL700C, JL701C, JL702C, JL703C, JL706C, JL707C, JL708C, JL709C, JL710C, JL711C, JL704C, JL705C, JL719C, JL718C, JL717C, JL720C, JL722C, JL721C)
- Aruba 8400 Switch Series (JL366A, JL363A, JL687A)
- Aruba 9300 Switch Series (R9A29A, R9A30A, R8Z96A)
- Aruba 10000 Switch Series (R8P13A, R8P14A)

Latest version available online

Updates to this document can occur after initial publication. For the latest versions of product documentation, see the links provided in Support and Other Resources.

Command syntax notation conventions

Convention	Usage
`example-text`	Identifies commands and their options and operands, code examples, filenames, pathnames, and output displayed in a command window. Items that appear like the example text in the previous column are to be entered exactly as shown and are required unless enclosed in brackets (`[ ]`).

Convention	Usage
example-text	In code and screen examples, indicates text entered by a user.
Any of the following: ■ *<example-text>* ■ <example-text> ■ *example-text* ■ *example-text*	Identifies a placeholder—such as a parameter or a variable—that you must substitute with an actual value in a command or in code: ■ For output formats where italic text cannot be displayed, variables are enclosed in angle brackets (< >). Substitute the text—including the enclosing angle brackets—with an actual value. ■ For output formats where italic text can be displayed, variables might or might not be enclosed in angle brackets. Substitute the text including the enclosing angle brackets, if any, with an actual value.
|	Vertical bar. A logical OR that separates multiple items from which you can choose only one. Any spaces that are on either side of the vertical bar are included for readability and are not a required part of the command syntax.
{ }	Braces. Indicates that at least one of the enclosed items is required.
[ ]	Brackets. Indicates that the enclosed item or items are optional.
... or . . .	Ellipsis: ■ In code and screen examples, a vertical or horizontal ellipsis indicates an omission of information. ■ In syntax using brackets and braces, an ellipsis indicates items that can be repeated. When an item followed by ellipses is enclosed in brackets, zero or more items can be specified.

About the examples

Examples in this document are representative and might not match your particular switch or environment.

The slot and port numbers in this document are for illustration only and might be unavailable on your switch.

Understanding the CLI prompts

When illustrating the prompts in the command line interface (CLI), this document uses the generic term `switch`, instead of the host name of the switch. For example:

```
switch>
```

The CLI prompt indicates the current command context. For example:

```
switch>
```

Indicates the operator command context.

```
switch#
```

Indicates the manager command context.

```
switch(CONTEXT-NAME)#
```

Indicates the configuration context for a feature. For example:

```
switch(config-if)#
```

Identifies the `interface` context.

Variable information in CLI prompts

In certain configuration contexts, the prompt may include variable information. For example, when in the VLAN configuration context, a VLAN number appears in the prompt:

```
switch(config-vlan-100)#
```

When referring to this context, this document uses the syntax:

```
switch(config-vlan-<VLAN-ID>)#
```

Where *<VLAN-ID>* is a variable representing the VLAN number.

Identifying switch ports and interfaces

Physical ports on the switch and their corresponding logical software interfaces are identified using the format:

member/slot/port

On the 4100i Switch Series

- *member*: Always 1. VSF is not supported on this switch.
- *slot*: Always 1. This is not a modular switch, so there are no slots.
- *port*: Physical number of a port on the switch.

For example, the logical interface `1/1/4` in software is associated with physical port 4 on the switch.

On the 6000 and 6100 Switch Series

- *member*: Always 1. VSF is not supported on this switch.
- *slot*: Always 1. This is not a modular switch, so there are no slots.
- *port*: Physical number of a port on the switch.

For example, the logical interface `1/1/4` in software is associated with physical port 4 on the switch.

On the 6200 Switch Series

- *member*: Member number of the switch in a Virtual Switching Framework (VSF) stack. Range: 1 to 8. The primary switch is always member 1. If the switch is not a member of a VSF stack, then member is 1.
- *slot*: Always 1. This is not a modular switch, so there are no slots.
- *port*: Physical number of a port on the switch.

For example, the logical interface `1/1/4` in software is associated with physical port 4 in slot 1 on member 1.

On the 6300 Switch Series

- *member*: Member number of the switch in a Virtual Switching Framework (VSF) stack. Range: 1 to 10. The primary switch is always member 1. If the switch is not a member of a VSF stack, then member is 1.
- *slot*: Always 1. This is not a modular switch, so there are no slots.
- *port*: Physical number of a port on the switch.

For example, the logical interface `1/1/4` in software is associated with physical port 4 on member 1.

On the 6400 Switch Series

- *member*: Always 1. VSF is not supported on this switch.
- *slot*: Specifies physical location of a module in the switch chassis.
  - Management modules are on the front of the switch in slots 1/1 and 1/2.
  - Line modules are on the front of the switch starting in slot 1/3.
- *port*: Physical number of a port on a line module.

For example, the logical interface `1/3/4` in software is associated with physical port 4 in slot 3 on member 1.

On the 83xx, 9300, and 10000 Switch Series

- *member*: Always 1. VSF is not supported on this switch.
- *slot*: Always 1. This is not a modular switch, so there are no slots.
- *port*: Physical number of a port on the switch.

For example, the logical interface `1/1/4` in software is associated with physical port 4 on the switch.

If using breakout cables, the port designation changes to x:y, where x is the physical port and y is the lane when split to 4 x 10G or 4 x 25G. For example, the logical interface 1/1/4:2 in software is associated with lane 2 on physical port 4 in slot 1 on member 1.

On the 8400 Switch Series

- *member*: Always 1. VSF is not supported on this switch.
- *slot*: Specifies physical location of a module in the switch chassis.
  - Management modules are on the front of the switch in slots 1/5 and 1/6.
  - Line modules are on the front of the switch in slots 1/1 through 1/4, and 1/7 through 1/10.
- *port*: Physical number of a port on a line module

For example, the logical interface `1/1/4` in software is associated with physical port 4 in slot 1 on member 1.

Identifying modular switch components

- Power supplies are on the front of the switch behind the bezel above the management modules. Power supplies are labeled in software in the format: *member/power supply*:
  - *member*: 1.
  - *power supply*: 1 to 4.
- Fans are on the rear of the switch and are labeled in software as: *member/tray/fan*:
  - *member*: 1.
  - *tray*: 1 to 4.
  - *fan*: 1 to 4.
- Fabric modules are not labeled on the switch but are labeled in software in the format: *member/module*:
  - *member*: 1.
  - *member*: 1 or 2.
- The display module on the rear of the switch is not labeled with a member or slot number.

REST v1 is deprecated with the the release of AOS-CX 10.09. Following AOS-CX 10.09, REST v1 will be hidden and can be consulted, but it is no longer supported.

Starting with AOS-CX 10.08, the REST v1 API is deprecated. It is still active, but no new features were added with 10.08 and no new features will be subsequently added.

The Aruba 6000 Switch Series and 6100 Switch Series only support the default VRF and has no management port. Therefore, references in this guide to other VRFs or the management port do no apply to the 6000 Switch Series and 6100 Switch Series. Configuration for these switches should be done over an SVI having a physical port with access to the SVI, since the physical ports in the 6000 and 6100 are not routed.

The Aruba 4100i Switch Series only supports the default VRF and has no management port. Therefore, references in this guide to other VRFs or the management port do no apply to the 4100i Switch Series. Configuration for these switches should be done over an SVI having a physical port with access to the SVI, since the physical ports in the 4100i are not routed.

AOS-CX REST API

Switches running the AOS-CX software are fully programmable with a REST (REpresentational State Transfer) API, allowing easy integration with other devices both on premises and in the cloud. This programmability—combined with the Aruba Network Analytics Engine—accelerates network administrator understanding of and response to network issues.

The AOS-CX REST API is a web service that performs operations on switch resources using HTTPS `POST`, `GET`, `PUT`, and `DELETE` methods.

The AOS-CX REST API enables programmatic access to the AOS-CX configuration and state database at the heart of the switch. By using a structured model, changes to the content and formatting of the CLI output do not affect the programs you write. The configuration is stored in a structured database, instead of a text file, making it easier to roll back changes, and dramatically reducing the risk of downtime and performance issues.

REST API versions

From the AOS-CX release 10.04, the AOS-CX switches support access through multiple versions of the REST API. The REST API versions supported on the AOS-CX switches are v1 and v10.04. The REST API version v10.04 is supported from AOS-CX release 10.04 and later.

The version declared in the REST request must match one of the versions of the REST API supported on the switch. The REST API version is included in the Uniform Resource Identifier (URI) used in REST requests.

In the following example, the REST API version is `v10.04`:

`https://192.0.2.5/rest/v10.04/system`

In the following example, the REST API version is `v1`:
`https://192.0.2.5/rest/v1/system`

Each REST API version has its own *REST API Guide*.

Differences among REST API versions

Resources, attributes, and behaviors might differ among the REST API versions and AOS-CX software release.

AOS-CX Network Analytics Engine scripts

URIs that specify monitors in Network Analytics Engine scripts must be REST v1 URIs.

Interfaces and ports

The REST v10.04 API provides the `interfaces` resource to configure and get information about switch ports and interfaces of all types. The `ports` resource is not supported by the REST v10.04 API.

GET method

The GET method query parameters differ between REST v1 and REST v10.04:

- The REST v10.04 `selector` parameter includes a value of `writable`, which enables you to get only the mutable attributes of a resource.
- The REST v1 `depth` parameter has a default of 0 and a range of 0 through 3.

The REST v10.04 `depth` parameter has a default of 1 and a range of 1 through 4. The REST v10.04 `depth=1` is equivalent to the REST v1 `depth=0`, and so on.

Resource collections

In REST v10.04, the members of a resource collection are represented as JSON objects, where the key is the index and the value is the URI of the resource.

For example, the response to a GET request to `/rest/v10.04/system/vrfs` is as follows:

```
{
  "default": "/rest/v10.04/system/vrfs/default",
  "mgmt": "/rest/v10.04/system/vrfs/mgmt"
}
```

In contrast, the response to a GET request to `/rest/v1/system/vrfs` on the same switch is as follows:

```
[
  "/rest/v1/system/vrfs/default",
  "/rest/v1/system/vrfs/mgmt"
]
```

The Aruba 6000 Switch Series and 6100 Switch Series only support the `default` VRF.

REST API access modes

The REST API supports two access modes:

- read-write (default)
- read-only

The default `read-write` access mode is not displayed in the `show running-configuration` command. You can change the access mode to read-only using the `https-server rest access-mode read-only` CLI command from the global configuration (`config`) context. You can validate the mode set using the `show https-server` command.

Read-write access mode

In the read-write access mode:

- The AOS-CX REST API Reference shows most of the supported read and write methods for all switch resources.
- The REST API can access and change every configurable aspect of the switch as modeled in the configuration and state database.

The REST API is powerful, but must be used with extreme caution: For most values, no semantic validation is performed on the data that you write to the database, and configuration errors can destabilize the switch.

Read-only access mode

In the read-only access mode:

- Most switch resources support only GET methods, but some resources allow PUT or POST methods. For example, you can use POST to log into the switch, use PUT to upload a new running configuration, or use POST to upload a new firmware version.
- For most switch resources, the AOS-CX REST API Reference does not show any write methods (POST, PUT, and DELETE) the resource might support. To show those write methods, read-write mode must be enabled.

REST API URI

A switch resource is indicated by its Uniform Resource Identifier (URI). A URI is the location of a specific web resource. A URI can be made up of several components, including the host name or IP address, port number, the path, and an optional query string.

Parts of a URI

The two main parts of a URI are the path and the (optional) query component.

URI path, including path parameters

The path is the part of the URI starting with the server URL and ending with the resource ID. In URIs that have a query component, the path is everything before the question mark (`?`). The path has a hierarchy. In a path, the forward slash (`/`) indicates the hierarchical relationship between resources.

Because the forward slash has a special meaning, the forward slash characters that are part of the URI path must be percent-encoded with the code `%2F`, which represents the forward slash. For example, the following URI represents the resource utilization for the management module in slot 1/5:

```
https://192.0.2.5/rest/v1/system/subsystems/management_module,1%2F5?attributes=resource_
utilization
```

URI prefix

The URI prefix is the system URL and REST API version information. This information is specific to a particular switch and REST API version, and is the same for every REST API request to that switch.

Script writers often create a variable for the URI prefix. Using a variable enables the writer to update a script or use the same script logic for a different switch by updating the value of the URI prefix variable.

The URI prefix contains the following:

Server URL

The web server address of the switch.

Examples:

- `https://192.0.2.5`
- `https://10.17.0.1`
- `https://myswitch.mycompany.com`

If Virtual Switching Extension (VSX) is enabled, you can access most resources of the peer switch from this switch by adding `/vsx-peer` in the URI path between the server URL and `/rest`. For more information about VSX, see VSX peer switches and REST API access.

For example:
`GET https://192.0.2.5/vsx-peer/rest/v1/system/vsx?attributes=oper_status`

REST API and version identifier

For example: `/rest/v1`

Path parameters

A path parameter is a part of the URI path that can vary. Typically path parameters indicate a specific instance of a resource in a collection, such as a specific VLAN in the `vlans` collection. The path can contain several path parameters. Path parameters are indicated by braces `{}`.

For example, the AOS-CX REST API Reference displays the resource for specific VLAN as the following:
`/system/vlans/{id}`

When you send a request for VLAN 10, the URI you provide must substitute the VLAN ID, `10`, for the `{id}` query parameter. For example:
`/system/vlans/10`

In the AOS-CX REST API Reference, you enter the value of the path parameter in the **Value** field of the **id** parameter.

Query component

In many cases, the unique identification of a resource requires a URI that contains both a path and a query component. The query component is sometimes called the query string.

For example, CPU utilization is a resource represented by the following URI:
`https://192.0.2.5/rest/v1/system/subsystems/management_module,1%2F5?attributes=resource_utilization`

In a URI, the question mark (`?`) indicates the beginning of the query component. The query component contains nonhierarchical data, and the format of the query string depends on the implementation of the REST API.

The query component often contains "*<key>=<value>*" pairs separated by the ampersand (`&`) character. Multiple attribute values are supported and are separated by commas. For example:
`https://192.0.2.5/rest/v1/system/vlans?depth=1&attributes=id,name,type`

"Dot" notation for Network Analytics Engine URIs only

When a URI defines a monitor in an Aruba Network Analytics Engine (NAE) script, attribute values in the query string support an additional dot notation that the Network Analytics Engine uses to access additional information. For example:
`https://192.0.2.5/rest/v1/system/subsystems/management_module,1%2F5?attributes=resource_utilization.cpu`

The dot notation is supported for certain URIs that define monitors only in NAE scripts. URIs in NAE scripts must only be REST v1 URIs.

Resources

In a REST API, the primary representation of data is called a **resource**. A resource is a representation of an entity in the system as a URI. The entities can include hardware objects, statistical information, configuration information, and status information. The URI might or might not include a query component. Resources are nouns—anything that can be named can be a resource.

Examples of resources:

- The resource utilization information:

`https://192.0.0.5/rest/v1/system/subsystems?attributes=resource_utilization`

- The list of configured VLANs:

`https://192.0.2.5/rest/v1/system/vlans`

- The list of all users:

`https://192.0.2.5/rest/v1/system/users`

- The user with the ID: `myadmin`:

`https://192.0.2.5/rest/v1/system/users/myadmin`

- The secondary firmware image:

`https://192.0.2.5/rest/v1/firmware?image=secondary`

Resource collections and singletons

Collections

A collection is a directory of resources managed by the server. Typically, a resource collection contains multiple resource instances and the collection name is in the plural form.

For example:

- `/system/vlans`
- `/system/users`
- `/fullconfigs`

A GET request to a collection returns the set of JSON objects representing the members of the collection. The following curl example shows the GET request and response returned for the `vlans` collection:

```
$ curl -k GET -b /tmp/auth_cookie "https://192.0.2.5/rest/v1/system/vlans"
{
  "1": "/rest/v1/system/vlans/1",
  "10": "/rest/v1/system/vlans/10",
  "20": "/rest/v1/system/vlans/20"
}
```

Each URI in the list represents a configured VLAN.

To get the JSON data for VLAN 10, you must either send the GET request to the URI representing VLAN 10 (`"/rest/v1/system/vlans/10"`), or you must use the depth parameter to expand the list of URIs in the `vlans` collection to get the JSON data for all the VLANs in the collection.

Subcollections

A single resource instance can also contain subcollections of resources.

- In the following example, `vlans` is a subcollection of the `system` resource:
`/system/vlans`
- In the following example, `routes` is a subcollection of the `default` VRF resource instance:
`/system/vrfs/default/routes`

Singletons

There are some resources that can only have one instance. These resources are called singletons and the resource collection name is in the singular form.

For example:

- `/system`
- `/system/vsx`
- `/firmware`

Because there is only one resource in a singleton collection, GET requests return the JSON representation of the resource instead of a URI list of one item. In addition, you do not need to supply a resource ID in the URL of a GET request. For example, the following GET request to the firmware URI returns the JSON data that represents the firmware resource:

```
$ curl -k GET -b /tmp/auth_cookie "https://192.0.2.5/rest/v1/firmware"
{
  "current_version": "TL.10.00.0006E-686-g4a43ab9",
  "primary_version": "TL.10.00.0006E-686-g4a43ab9",
  "secondary_version": "",
  "default_image": "primary",
  "booted_image": "primary"
}
```

Categories of resource attributes

Resources can contain many attributes, and they are organized into the following categories to enable more efficient management:

Configuration attributes

Configuration attributes represent user-owned data. Although an attribute must be in the configuration category to be modified by a user, not all attributes in the configuration category can be modified after the resource instance is created. Configuration attributes that cannot be changed after the resource is created are called **immutable** attributes. This distinction matters when using a PUT request, because immutable attributes cannot be included in the request body.

For example, a VLAN ID is an immutable attribute. You cannot change the ID of the VLAN after the VLAN is created. The VLAN name, in contrast, is a **mutable** (**writable**) attribute. You can change the VLAN name after the VLAN is created.

Writable attributes

Writable attributes are the subset of configuration attributes that are **mutable**. Writable attributes can be modified by a user after the resource is created. When using the PUT method to modify a resource, only writable attributes can be included in the request body.

In REST v10.04 and later versions, the GET method `selector` parameter includes a value of `writable`, which enables you to get only the mutable configuration attributes of a resource.

Status attributes

Status attributes contain system-owned data such as the admin account and various status fields. You cannot create or modify instances of attributes in this category.

Statistics attributes

Statistics attributes contain system-owned data such as counters. You cannot create or modify instances of attributes in this category.

Attribute categories might vary

A given attribute need not necessarily be in the same category from resource to resource, or even resource instance to resource instance. If the system owns an instance of a resource, attributes of that resource (which might be configuration attributes if a user owns the resource instance) become status attributes, which cannot be modified by users.

For example, a user can create VLANs. However, the system can also create VLANs. System-owned VLANs have many attributes that are considered to be in the status category and not the configuration category. The status category is used when the data is owned by the system and cannot be overwritten by a user.

Often a resource has a single attribute that indicates whether the resource is owned by the system or by a user. For example, for a VLAN, the `type` attribute indicates whether the VLAN was created by a user.

When this indicator attribute indicates that the resource is owned by the system, the other attributes that might have been in the configuration category are categorized as status attributes. Likewise, when the indicator attribute indicates that the resource is owned by a user, the other configuration attributes remain available for modification by users. In other words, the categories for other attributes on the resource follow the indicator attribute.

The AOS-CX Web UI and AOS-CX real-time notifications subsystem rely on the REST API, therefore, all three are enabled or disabled together.

To access the REST API, Web UI, or notifications subsystem, the HTTPS server must be enabled on the specified VRF. The VRF you specify determines from which network the features can be accessed. You can enable access on multiple VRFs, including user-defined VRFs, by entering the `https-server vrf` command for each VRF on which you want to enable access.

Prerequisites

- You must be in the global configuration context: `switch(config)#`.
- The password for the `admin` user must be configured on the switch.

Procedure

Enable HTTPS server access for the specified VRF.

For example:

- To enable access on all data ports on the switch, specify the default VRF:

```
switch(config)# https-server vrf default
```

The Aruba 6000 Switch Series and 6100 Switch Series only support the `default` VRF.

- To enable access on the OOBM port (management interface IP address), specify the management VRF (not applicable to the 6000 and 6100):

```
switch(config)# https-server vrf mgmt
```

- To enable access on ports that are members of the VRF named `vrfprogs`, specify `vrfprogs`:

```
switch(config)# https-server vrf vrfprogs
```

If the switch responds with the following error, the `admin` user must have a valid password:

```
Failed to enable https-server on VRF mgmt. 'admin' password is not set
```

The switch is shipped from the factory with a default user named `admin` without a password. The `admin` user must set a valid password before HTTPS servers can be enabled.

Setting the admin password

Use the following API to login as the admin.

```
POST /rest/v1/login?username=admin
```

A new session is started and a response code `268` is returned along with the message: `"Session is restricted. Administrator password must be set before continuing."`

This session is valid only to change the admin password and logout from the REST API UI. Any other request will return a `Forbidden code (403).`

Use the following API to change the admin password. Ellipses (...) represent data not included in the example.

```
PUT /rest/v1/system/users/admin
{
...
"password": "<enter the password>"
...
}
```

After the password is changed successfully, the session restriction is removed.

Setting the REST API access mode to read-write

Enabling the read-write mode on the REST API allows write operations (POST, PUT, and DELETE) to be called on all configurable elements in the switch database.

The REST API in read-write mode is intended for use by advanced users who have a good understanding of the system schema and data relationships in the switch database.

The REST API in read-write mode can access every configurable element in the database. The REST API is powerful, but must be used with extreme caution: For most values, no semantic validation is performed on the data that you write to the database, and configuration errors can destabilize the switch.

Setting the access mode is independent from enabling or disabling access to the REST API.

Prerequisites

You must be in the global configuration context: `switch(config)#`.

Procedure

Set the REST API access mode to `read-write`.

```
switch(config)# https-server rest access-mode read-write
```

Showing the REST API access configuration

To show the REST API access configuration, in the manager context (`#`) of the CLI, enter the `show https-server` command.

For example:

```
switch# show https-server
       HTTPS Server Configuration
```

```
       ----------------------------
        VRF               : mgmt, default
        REST Access Mode  : read-write
```

The Aruba 6000 Switch Series and 6100 Switch Series only support the `default` VRF.

The Aruba 4100i Switch Series only supports the `default` VRF.

The command output lists the VRFs on which access to REST API is enabled and shows the current REST API access mode.

If access is not enabled on any VRF, the VRF configuration is displayed as `<none>`.

For example:

```
switch# show https-server
       HTTPS Server Configuration
       ----------------------------
        VRF               : <none>
        REST Access Mode  : read-write
```

Disabling access to the REST API

The AOS-CX Web UI and AOS-CX real-time notifications subsystem rely on the REST API, therefore, all three are enabled or disabled together.

Prerequisites

You must be in the global configuration context: `switch(config)#`.

Procedure

Disable HTTPS server access for the specified VRF by using the `no` form of the `https-server vrf` command.

For example, the following command disables REST API access on the switch data ports in the default VRF:

```
switch(config)# no https-server vrf default
```

You can use the `show https-server` command to show the current configuration:

```
switch# show https-server

HTTPS Server Configuration
----------------------------

 VRF               : mgmt
 REST Access Mode  : read-write
```

HTTPS server commands

https-server max-user-sessions

`https-server max-user-sessions <SESSION-AMT>`

Description

Sets the maximum amount of concurrent open sessions for any given user through the HTTPS server. The amount of concurrent open sessions may have an impact on system performance, so it is recommended to set this value to the minimum necessary.

Parameter	Description
<SESSION-AMT>	Specifies the maximum number of user sessions allowed. Default: 6. Maximum value: 8.

Example

Set the maximum number of concurrent user sessions to the maximum of 8:

```
switch(config)# https-server max-user-sessions 8
```

Command History

Release	Modification
10.08	Command introduced

Command Information

Platforms	Command context	Authority
All platforms	`config`	Administrators or local user group members with execution rights for this command.

https-server rest access-mode

`https-server rest access-mode {read-only | read-write}`

Description

Changes the REST API access mode. The default mode is read-write.

Parameter	Description
`read-write`	Selects the read/write mode. Allows POST, PUT, PATCH, and DELETE methods to be called on all configurable elements in the switch database.
`read-only`	Selects the read-only mode. Write access to most switch resources through the REST API is disabled.

Usage

Setting the mode to `read-write` on the REST API allows POST, PUT, PATCH, and DELETE methods to be called on all configurable elements in the switch database.

By default, REST APIs in the device are in the read-write mode. Some switch resources allow POST, PUT, PATCH, and DELETE regardless of REST API mode. REST APIs that are required to support the Web UI or the Network Analytics Engine expose POST, PUT, PATCH, or DELETE operations, even if the REST API access mode is set to read-only.

The REST API in read/write mode is intended for use by advanced programmers who have a good understanding of the system schema and data relationships in the switch database.

Because the REST API in read/write mode can access every configurable element in the database, it is powerful but must be used with extreme caution: No semantic validation is performed on the data you write to the database, and configuration errors can destabilize the switch.

On 6300 switches or 6400 switches, by default, the HTTPS server is enabled in `read-write` mode on the `mgmt` VRF. If you enable the HTTPS server on a different VRF, the HTTPS server is enabled in `read-only` mode.

Example

```
switch(config)# https-server rest access-mode read-only
```

Command History

Release	Modification
10.07 or earlier	--

Command Information

Platforms	Command context	Authority
All platforms	`config`	Administrators or local user group members with execution rights for this command.

https-server rest firmware-site-distribution

```
https-server rest firmware-site-distribution
no https-server rest firmware-site-distribution
```

Description

Enables the firmware site distribution server.

The firmware site distribution allows you to use a switch to distribute a firmware image file to other switches in the same network. This prevents the switches from connecting to the cloud or an external network to download a firmware image file.

On enabling the firmware site distribution, it exposes a REST endpoint that allows the switches to download a switch primary or secondary firmware image.

As per the limitation, up to two switches can download the firmware image simultaneously.

This endpoint is to be used along with REST `/firmware` endpoint to handle the firmware download and installation process.

The `no` form of this command disables the firmware site distribution server.

Example

Enabling the firmware site distribution server:

```
switch(config)# https-server rest firmware-site-distribution
```

Disabling the firmware site distribution server:

```
switch(config)# no https-server rest firmware-site-distribution
```

Command History

Release	Modification
10.10	Command introduced

Command Information

Platforms	Command context	Authority
All platforms	`config`	Administrators or local user group members with execution rights for this command.

https-server session close all

`https-server session close all`

Description

Invalidates and closes all HTTPS sessions. All existing Web UI and REST sessions are logged out and all real-time notification feature WebSocket connections are closed.

Usage

Typically, a user that has consumed the allowed concurrent HTTPS sessions and is unable to access the session cookie to log out manually must wait for the session idle timeout to start another session. This command is intended as a workaround to waiting for the idle timeout to close an HTTPS session. This command stops and starts the `hpe-restd` service, so using this command affects all existing REST sessions, Web UI sessions, and real-time notification subscriptions.

Example

```
switch# https-server session close all
```

Command History

Release	Modification
10.07 or earlier	--

Command Information

Platforms	Command context	Authority
All platforms	Manager (#)	Administrators or local user group members with execution rights for this command.

https-server session-timeout

```
https-server session-timeout <MINUTES>
```

Description

Configures the timeout, in minutes, for any given HTTPS server session. A value of 0 disables the timeout.

Parameter	Description
<MINUTES>	Specifies the maximum idle time, in minutes for an HTTPS session. Default: 20. Maximum: 480 (8 hours). 0 disables the timeout, but the maxium is still enforced.

Example

```
switch(config)# https-server session-timeout 10
```

Command History

Release	Modification
10.08	Command introduced

Command Information

Platforms	Command context	Authority
All platforms	config	Administrators or local user group members with execution rights for this command.

https-server vrf

```
https-server vrf <VRF-NAME>
no https-server vrf <VRF-NAME>
```

Description

Configures and starts the HTTPS server on the specified VRF. HTTPS server features include the REST API and the web user interfaces.

The `no` form of the command stops any HTTPS servers running on the specified VRF and removes the HTTPS server configuration.

Parameter	Description
<VRF-NAME>	Specifies the VRF name. Required. Length: Up to 32 alpha numeric characters.

Usage

By using this command, you enable access to both the Web UI and to the REST API on the specified VRF. You can enable access on multiple VRFs.

By default, 8320, 8325, 8360, 8400, 9300, and 10000 Switch Series have an HTTPS server enabled on the `mgmt` VRF.

By default, the 6200, 6300, and 6400 Switch Series have an HTTPS server enabled on the `mgmt` VRF and on the `default` VRF.

When the HTTPS server is not configured and running, attempts to access the Web UI or REST API result in `404 Not Found` errors.

The VRF you select determines from which network the Web UI and REST API can be accessed.

For example:

- If you want to enable access to the REST API and Web UI through the OOBM port (management IP address), specify the built-in management VRF (`mgmt`).
- If you want to enable access to the REST API and Web UI through the data ports (for "inband management"), specify the built-in default VRF (`default`).
- If you want to enable access to the REST API and Web UI through only a subset of data ports on the switch, specify other VRFs you have created.

Aruba Network Analytics Engine scripts run in the default VRF, but you do not have to enable HTTPS server access on the default VRF for the scripts to run. If the switch has custom Aruba Network Analytics Engine scripts that require access to the Internet, then for those scripts to perform their functions, you must configure a DNS name server on the default VRF.

Examples

Enabling access on all ports on the switch, specify the default VRF:

```
switch(config)# https-server vrf default
```

Enabling access on the OOBM port (management interface IP address), specify the management VRF:

```
switch(config)# https-server vrf mgmt
```

Enabling access on ports that are members of the VRF named `vrfprogs`, specify `vrfprogs`:

```
switch(config)# https-server vrf vrfprogs
```

Enabling access on the management port and ports that are members of the VRF named `vrfprogs`, enter two commands:

```
switch(config)# https-server vrf mgmt
switch(config)# https-server vrf vrfprogs
```

The 6200 switches support only two VRFs. One management VRF and one default VRF. You cannot add another VRF.

Command History

Release	Modification
10.07 or earlier	--

Command Information

Platforms	Command context	Authority
All platforms	`config`	Administrators or local user group members with execution rights for this command.

show https-server

```
show https-server [vsx-peer]
```

Description

Shows the status and configuration of the HTTPS server. The REST API and web user interface are accessible only on VRFs that have the HTTPS server features configured.

Parameter	Description
`vsx-peer`	Shows the output from the VSX peer switch. If the switches do not have the VSX configuration or the ISL is down, the output from the VSX peer switch is not displayed. This parameter is available on switches that support VSX.

Usage

Shows the configuration of the HTTPS server features.
`VRF`
Shows the VRFs, if any, for which HTTPS server features are configured.
`REST Access Mode`
Shows the configuration of the REST access mode:
`read-write`
POST, PUT, and DELETE methods can be called on all configurable elements in the switch database. This is the default value.
`read-only`
Write access to most switch resources through the REST API is disabled.

Examples

```
switch# show https-server
```

```
HTTPS Server Configuration
---------------------------

 VRF               : default, mgmt
 REST Access Mode  : read-write

 Max sessions per user  : 6

 Session timeout        : 20
```

Command History

Release	Modification
10.07 or earlier	--

Command Information

Platforms	Command context	Authority
All platforms	Manager (#)	Administrators or local user group members with execution rights for this command.

You can access the REST API using any REST client interface that supports HTTPS requests, and supports obtaining and passing a session cookie.

Examples of client interfaces include the following:

Scripts and programs that support HTTPS requests

A flexible way to access the AOS-CX REST API is to use a programming language that supports HTTPS requests, such as Python, to write programs that automate network management tasks.

The curl command-line interface

You can use curl commands either interactively or within a script to make REST requests. Using curl commands is a way to execute GET requests without writing a script. Using curl commands is a way to test REST requests that you are considering to incorporate into an application.

Browser-based interfaces such as Postman or the AOS-CX REST API Reference

Examples of browser-based interfaces include Postman and the AOS-CX REST API Reference.

The AOS-CX REST API Reference documents the switch resources, parameters, and JSON models for each HTTPS method supported by the resource. Because the AOS-CX REST API Reference is browser-based, it can share the session cookie with a Web UI session active in another browser tab. The AOS-CX REST API Reference is not intended to be used as a configuration tool and is not required for day-to-day operations.

The AOS-CX REST API Reference is one way to execute GET requests without writing a script. The AOS-CX REST API Reference can be used during script coding to help you construct the URIs—with their query parameters—that you use in a script or curl command.

Authenticating REST API sessions

When you start a REST API session, you use the POST method to access the `login` resource of the switch and pass the username and password information as data. Ensure that HTTPS is configured to use port 443. Requests to port 80 are redirected to port 443.

If the credentials are accepted, your authenticated session is started for that username, and the switch returns a cookie containing encoded session information.

In subsequent calls to the API—including to the `logout` resource—the session cookie is passed back to the switch.

The same session cookie is shared across browser tabs, and depending on the browser, multiple browser windows. However, the same session cookie is not shared across devices and scripts. For example, if a user logs into the Web UI from a laptop, again with a tablet, and then uses the same user name in a curl command, that user has three concurrent client sessions.

The maximum number of concurrent HTTPS sessions per user per switch is six. There is an upper limit of 48 total sessions per switch. It is a best practice to log out of HTTPS sessions when you are finished using them.

HTTPS sessions will automatically time out after 20 minutes of inactivity, and have a hard time limit of eight hours, regardless of whether the session is active. You can run the `https-server session close all` command to close all current HTTPS sessions. For more information about using the command, see `https-server session close all`.

Authentication through methods other than the session cookie, such as OAuth or certificates, is not supported. The server uses self-signed certificates.

The procedure to pass the session cookie back and forth from the switch depends on how you access the REST API.

For example:

- If you log in to the REST API using the AOS-CX REST API Reference or using the Web UI and open the API Reference in another browser tab, the browser handles the session cookie for you. You do not have to save or otherwise manage the session cookie.
- If you access the REST API using another method, such as the curl tool, you must do the following:
  - Save the session cookie returned from the login request.
  - Pass that saved cookie to the switch with every subsequent request you make to the REST API, including the `logout` resource.

Although it is possible to pass the user name and password information as a query string in the login URL, browser logs or tools outside the switch might save the accessed URL in cleartext in log entries. Instead, Hewlett Packard Enterprise recommends that you pass the credential information as data when using programs such as curl to log in to the switch.

For examples of accessing the REST API using curl, see Accessing the REST API using curl.

User groups and access authorization

For switch resources, the access authorization granted to a user is determined by the group to which the user belongs. Each user group is assigned a number that represents a privilege level. This number is used to represent the user group in logs and in places in which the group name is too long to display.

The following predefined user groups are supported:

User group	Privilege level	Description
`operators`	1	Authorized for read access to non-sensitive data.
`administrators`	15	Authorized for read and write access to all switch resources. Write access also requires that the REST API is in read-write access mode.
`auditors`	19	Authorized for read access to audit log (`/logs/audit`) and event log (`/logs/event`) resources only.

All users can access the POST method of the `login` and `logout` resources. However, the login requests fail if the user is not a member of one of the predefined user groups. For example, login attempts fail when attempted by a member of a user-defined local user group.

If a user attempts a request for which they are not authorized, the switch returns an HTTP 403 "Forbidden" error.

If an authorized user attempts a write request but the REST API is in read-only mode, the switch returns an HTTP 404 "Page not found" error.

The AOS-CX operating system includes the AOS-CX REST API Reference, which is a web interface based on the Swagger 3.0 UI. For more information about Swagger, see https://swagger.io/.

The AOS-CX REST API Reference provides the reference documentation for REST API, including the switch resources, parameters, errors, and JSON models for each HTTPS method supported by the resource. The AOS-CX REST API Reference shows most of the supported read and write methods for all switch resources.

Since the AOS-CX REST API Reference is browser-based, it can share the session cookie with a Web UI session active in another browser tab. The AOS-CX REST API Reference is not intended to be used as a configuration tool and is not required for day-to-day operations.

The AOS-CX REST API Reference is one way to execute HTTP requests like GET, PUT, POST, and DELETE, without writing a script. The AOS-CX REST API Reference can be used during script coding to help you construct the URIs and data body (in the case of POST or PUT)—with their query parameters—that you use in a script or curl command.

Accessing the REST API using the AOS-CX REST API Reference

Although the AOS-CX REST API Reference interacts directly with the REST API, the AOS-CX REST API Reference is not intended as a management or configuration interface. Use caution when using the **Submit** button for POST or PUT methods because this action can result in changes to your current environment.

Prerequisites

- HTTPS server access must be enabled on the VRF from which you are accessing the switch.
- With a few exceptions, using the PUT, POST, or DELETE methods require the following conditions to be true:
  - The REST API access mode must be set to `read-write`.
  - The user name you use to log in must be a member of the `administrators` group.

Procedure

- To view the reference documentation for the REST v10.04 API, access the following URL using a browser: `https://<IP-ADDR>/api/v10.04/`

<IP-ADDR> is the IP address or hostname of your switch.

For example: `https://192.0.2.5/api/v10.04/`

- To open the reference for the REST v1 API, open a browser at: `https://<IP-ADDR>/api/v1/` or at `https://<IP-ADDR>/api/`

<IP-ADDR> is the IP address or hostname of your switch.

For example: `https://192.0.2.5/api/v1/`

Logging in and logging out using the AOS-CX REST API Reference

Prerequisites

- Access to the switch REST API must be enabled.
- You must have used a supported browser to access the switch at:

`https://<IP-ADDR>/api/v10.04/`

<IP-ADDR> is the IP address or hostname of your switch.

Procedure

Log in to the switch using the **Login** resource.

1. Expand the **Login** section.

The POST method for the login resource is displayed.

2. Expand the resource by clicking POST or the resource name, `/login`.
3. Click **Try it out**.
4. Enter your user name in the **User name** field.
5. Enter your password in the **Password** field.
6. Click **Execute**. If the operation is successful, the REST API returns response code 200.

When you finish your session, log out by expanding the **Logout** resource and clicking **Execute**.

AOS-CX REST API Reference basics

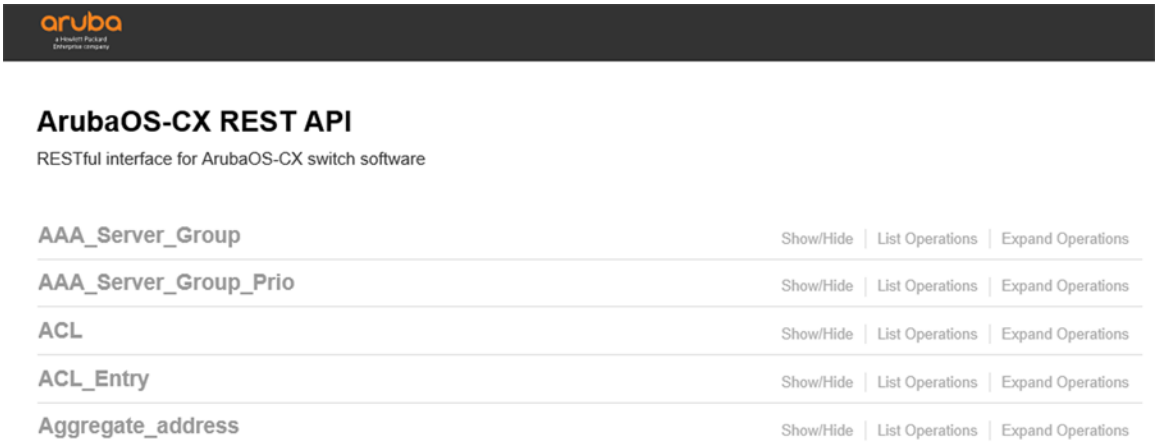
The AOS-CX REST API Reference is a web interface based on Swagger 2.0. The AOS-CX REST API Reference does the following:

- Documents the switch REST API resources, methods, models, and responses.
- Enables you to interact with the switch to view or change the configuration.

For more information about Swagger, see https://swagger.io/.

AOS-CX REST API Reference home page

The following is an example of a portion of the AOS-CX REST API Reference home page for a switch running AOS-CX software:



At the bottom of the page, the AOS-CX REST API Reference shows base URL and version information. For example:

```
[ BASE URL: /rest/v1, API VERSION: 1.0.0 ]
```

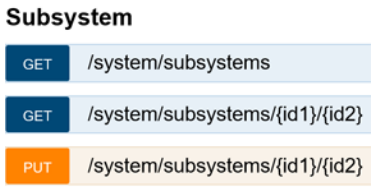
- The switch resource URIs are organized in groups. The group names are listed in alphabetical order on the AOS-CX REST API Reference home page.

The group name does not always match the resource collection name. Use the group names as a navigation aid only.

- Group names that are in gray have the URI entries—also called endpoints—collapsed. When you hover over the group name, it turns black. Click the group name to expand it and show the list of methods and URIs in the group.

You can also use the **Show/Hide**, **List Operations**, and **Expand Operations** controls to expand or collapse all the members of the group.

- The following example shows the list of the methods and resource URIs in the **Subsystem** group:

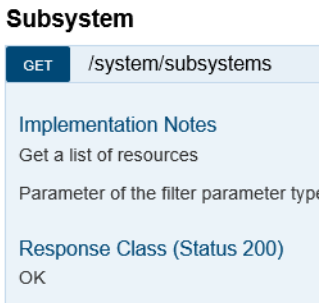


This view is the same view that is shown when you click the **List Operations** control of the **Subsystem** group.

  - The methods that are shown might depend on the REST API access mode. Some methods might not be displayed if the REST API access mode is `read-only`.
  - Methods and resources might be displayed that you do not have the authorization to access. For example, users with operator rights are not authorized to make PUT or POST requests to most resources. If you submit a request for which you are not authorized, the switch returns the following error:

    ```
    HTTP error 403 "Forbidden"
    ```

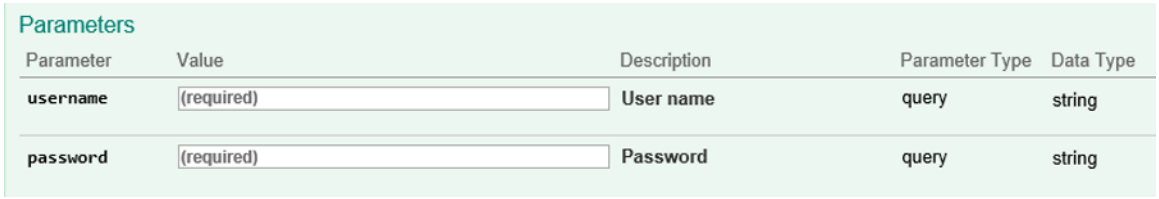
  - The resource collection name is `subsystems` (not Subsystem).
  - Items in braces, such as `{id 1}`, are path parameters. If you submit a request to a resource URI that includes a path parameter, you are required to supply a value for the parameter.
- To show more information about an item on the list, click the URI path. The following shows part of the information displayed when /system/subsystems is selected:



  You can use the browser scroll bar to navigate to information about the implementation of this method and resource, including the required and optional parameters.

  - Required parameters are shown in **bold**.

    For example, the POST method of the login resource requires a user name and password:

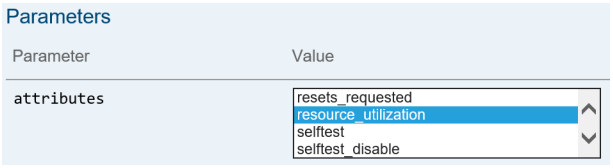


Path parameters such as `{id}` are listed as required parameters:

- The **Submit** button sends the request.

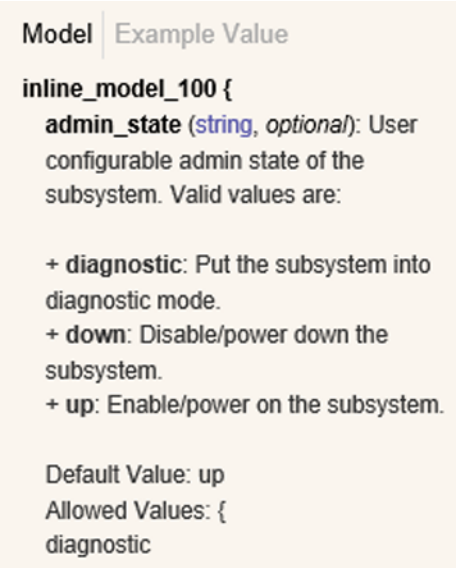
Although the AOS-CX REST API Reference interacts directly with the REST API, the AOS-CX REST API Reference is not intended as a management or configuration interface. Use caution when using the **Submit** button for POST or PUT methods because this action can result in changes to your current environment.

- In GET requests, there can be multiple attributes and parameters you can use to filter results. For example:



You can select multiple attributes:

  - To select a range of attributes, click the first attribute, then press **Shift**, and then click the last attribute in the range you want to select.
  - To select attributes that are not adjacent in the list, press **Ctrl**, then click each attribute you want to select.
- The JSON model for the resource is described in **Model** and shown with example values in **Example Values** for each method. The following example shows the JSON model and example values for PUT method of the `/system/subsystems/{id1}/{id2}` resource:



- After a request is submitted, the AOS-CX REST API Reference shows additional information, including the following:
  - The curl command equivalent of the submitted request
  - The submitted request URL, including the specified parameters and values.
  - The response body returned by the switch
  - The response code returned by the switch
  - The response headers returned by the switch
- The curl command and request URLs are displayed using percent encoding for certain characters in the query string portion of the URL:

Character	Percent-encoded equivalent
, (comma)	%2C
: (colon)	%3A

When you enter curl commands or submit requests through other means, percent encoding is permitted but not required in the query string of the URI.

Write methods (POST, PUT, and DELETE)

The supported write methods are POST, PUT, and DELETE:

- POST creates a resource.
- PUT replaces a resource.
- DELETE removes a resource.

Not all resources support all write methods. See the AOS-CX REST API Reference for the methods supported by each resource. The REST API must be in read-write mode for the AOS-CX REST API Reference to show all the write methods a resource supports.

Considerations when making configuration changes

The REST API can access and change every configurable aspect of the switch as modeled in the configuration and state database. However, changing the configuration of a switch through the REST API can be different than changing the configuration through the CLI.

A single configuration change to the switch can require changes to multiple resources in the configuration and state database. Often these changes must be made in a specific order.

The CLI commands have been programmed to work "behind the scenes" to make the correct database changes and to perform data validation checks on the user input. In contrast, when you use the REST API to make a configuration change, you must become familiar with the representational models of the switch resources, the type and format of the data required, and the required order of write operations to various resources.

The REST API is powerful but must be used with extreme caution: No semantic validation is performed on the data you write to the database, and configuration errors can destabilize the switch. Hewlett Packard Enterprise recommends that you refer to the tested examples when using the REST API to make configuration changes.

Considerations for ports and interfaces

The REST v1 API provides the `interfaces` resource to configure and get information about switch ports and interfaces of all types. You do not use the `ports` resource to manage ports.

Hardware (system) interfaces

- Hardware interfaces are of type `system`.
- Hardware interfaces are included in the database automatically.
- Interfaces of type `system` cannot be added or deleted.

LAG interfaces

- LAG interfaces are of type `lag`.
- You can use the DELETE method to delete a LAG interface.

Example of creating a LAG interface with member ports 1/1/1 and 1/1/2:

Method and URI:

```
POST "/rest/v1/system/interfaces"
```

Request body:

```
{
  "name": "lag50",
  "vrf": "/rest/v1/system/vrfs/default",
  "type": "lag",
  "interfaces": [
    "/rest/v1/system/interfaces/1%2F1%2F1",
    "/rest/v1/system/interfaces/1%2F1%2F2"
  ]
}
```

VLAN interfaces

- VLAN interfaces are of type `vlan`.
- You can use the DELETE method to delete a VLAN interface.

Example of creating a VLAN interface:

Method and URI:

```
POST "/rest/v1/system/interfaces"
```

Request body:

```
{
   "name": "vlan2",
   "vlan_tag": "/rest/v1/system/vlans/2",
   "vrf": "/rest/v1/system/vrfs/default",
   "type": "vlan"
}
```

Write methods (POST, PUT) supported in read-only mode

The following switch resources support write methods (POST, PUT, or both) even when the REST API access mode is set to read-only:

- Configuration management: `*/rest/v1/fullconfigs*`
- Firmware: `*/rest/v1/firmware*`
- User login and logout:
  - `*/rest/v1/login`
  - `*/rest/v1/logout`
- Aruba Network Analytics Engine and scripts: `*/rest/v1/system/nae_scripts*`

The * indicates more text to be added in URI path.

GET method usage and considerations

The GET method is a read method that gets the resource specified by the URI. Data is returned in JSON format in the response body.

Using GET on a resource collection results in a list of URIs. Each URI in the list corresponds to a specific resource in the collection.

Using GET on a specific resource returns the attributes of that resource.

GET method parameters

The GET method supports the following parameters in the query string of the URI:

- `attributes`
- `count`
- `depth`
- `filter`
- `selector`

A path query parameter is specified as a "key=value" pair. When permitted, multiple values are separated by the comma (,) character.

For example:

- `attributes=id,name,type`
- `count=true`

- `depth=1`
- `filter=type:static`
- `selector=writable`

A path query parameter can be used alone or in combination with other parameters. The ampersand (`&`) character separates each parameter in the string.

For example:

```
GET "https://192.0.2.5/rest/v1/system/vlans?depth=1&attributes=id,name,type"
```

The `count` and `filter` attributes and wildcard character are supported from AOS-CX release 10.05 and later.

Attributes parameter

The `attributes` parameter of the GET method reduces the returned data for each entry to include only the attributes specified in the comma-separated list. The attribute names in the URI must match the attribute names in the AOS-CX REST API Reference.

For a list of the available attributes for a resource, see the GET method of that resource in the AOS-CX REST API Reference.

Example request:

```
GET "https://192.0.2.5/rest/v1/system/vlans?depth=1&attributes=id,name,type"
```

Example response:

```
{
  {
    "id": 1,
    "name": "DEFAULT_VLAN_1",
    "type": "default"
  },
  {
    "id": 2,
    "name": "VLAN2",
    "type": "static"
  },
  {
    "id": 3,
    "name": "VLAN3",
    "type": "static"
  }
}
```

Count parameter

The `count` parameter of the GET method returns the number of entries that match the specified URI. The count parameter can be useful when specifying resource collections or for getting statistical information.

You can specify the count parameter as either of the following:

- `count`
- `count=true`

Examples:

- Use the `count` parameter to get the total number of VLANs:

```
GET "https://192.0.2.5/rest/v1/system/vlans?count=true"
```

- Use the `count` parameter with the `filter` parameter to get the total number of interfaces in a `down` administrative state:

```
GET "https://192.0.2.5/rest/v1/system/interfaces?count&filter=admin_state:down"
```

Depth parameter

The `depth` parameter of the GET method specifies to what level the URIs in response bodies must be expanded and replaced by the JSON representation of those resource:

- Default: 0
- Maximum: 3

For each level of depth, the REST API expands one level of URIs into their JSON data representations in the response body.

Using the depth parameter can result in large amounts of returned data, depending on the number of items in the list and the amount of JSON data that represents each item.

For example, a GET request on the `vlans` resource returns a list of URIs. Example request:

```
GET "https://192.0.2.5/rest/v1/system/vlans"
```

Example response:

```
[
  "/rest/v1/system/vlans/1",
  "/rest/v1/system/vlans/10",
  "/rest/v1/system/vlans/20"
]
```

To specify that those URIs also be expanded and replaced with the JSON data, specify `depth=1` as a parameter in the GET request.

Example request:

```
GET "https://192.0.2.5/rest/v1/system/vlans?depth=1"
```

Example response (ellipses represent data omitted from this example):

```
[
  {
    "id": 1,
          "name": "DEFAULT_VLAN_1",
          "type": "default",
          …
    "flood_enabled_subsystems": [
      {
         URI-of-first-subsystem
      },
      …
      {
         URI-of-last-subsystem
      }
    ]
  },
        { "id": 10,
          "name": "vlan10",
          "type": "static",
          …
    "flood_enabled_subsystems": [
```

```
        {
           URI-of-first-subsystem
        },
        …
        {
           URI-of-last-subsystem
        }
     ]
   }
]
```

Each VLAN in the preceding example includes an attribute, `flood_enabled_subsystems`, which contains a list of URIs that represent the flood-enabled systems. To specify that those URIs also be expanded and replaced with the JSON data, specify `depth=2` as a parameter in the GET request.

Filter parameter

The `filter` parameter of the GET method reduces the returned data to include only those entries that match the filter criteria. Specify the filter criteria in a comma-separated list of attribute `name:value` pairs.

Examples:

- Use the `filter` parameter to get only the static VLANS:

```
GET "https://192.0.2.5/rest/v1/system/vlans?filter=type:static"
```

- Use the `filter` parameter to get the BGP routes that have 1.1.1.1 as a peer:

```
GET "https://192.0.2.5/rest/v1/system/vrfs/default/bgp_routes?filter=peer:1.1.1.1"
```

Selector parameter

The `selector` parameter of the GET method filters the returned data to include only those attributes that belong to the specified category. By using the `selector` parameter, you avoid having to list attributes individually using the `attributes` parameter. The default is to include all categories. Use a comma (,) to separate multiple category values.

The selector categories are the following:

`configuration`

Contains user-owned information. Attributes in the configuration category can be supplied by users through REST requests or through the switch CLI. Although an attribute must be in the configuration category to be modified by a user, not all attributes in the configuration category can be modified after the resource instance is created.

`statistics`

Contains system-supplied data such as counters. Attributes in the `statistics` category cannot be written by users.

`status`

Contains system-owned data such as the admin account and various status fields. Attributes in the `status` category cannot be written by users.

For example, to get the configuration attributes of all VLANs, when you specify the URI of the GET method, do the following:

- Specify `depth=1` to direct the REST API return the JSON representations of each VLAN instead of the URI of each VLAN in the list. If you do not specify `depth=1`, the REST API returns each VLAN represented as a URI, which does not include the attributes of the individual VLANs.
- Specify the `selector` parameter with the value `configuration`.

```
GET "https://192.0.2.5/rest/v1/system/vlans?depth=1&selector=configuration"
```

Example response:


```
[
  {
    "admin": "up",
    "id": 20,
    "mgmd_enable": {},
    "mgmd_igmp_block_ports": [],
    "mgmd_igmp_fastleave_ports": [],
    "mgmd_igmp_forcedfastleave_ports": [],
    "mgmd_igmp_forward_ports": [],
    "mgmd_igmp_static_groups": [],
    "mgmd_mld_block_ports": [],
    "mgmd_mld_fastleave_ports": [],
    "mgmd_mld_forcedfastleave_ports": [],
    "mgmd_mld_forward_ports": [],
    "mgmd_mld_static_groups": [],
    "name": "VLAN20",
    "type": "static"
  },
  {
    "admin": "up",
    "id": 10,
    "mgmd_enable": {},
    "mgmd_igmp_block_ports": [],
    "mgmd_igmp_fastleave_ports": [],
    "mgmd_igmp_forcedfastleave_ports": [],
    "mgmd_igmp_forward_ports": [],
    "mgmd_igmp_static_groups": [],
    "mgmd_mld_block_ports": [],
    "mgmd_mld_fastleave_ports": [],
    "mgmd_mld_forcedfastleave_ports": [],
    "mgmd_mld_forward_ports": [],
    "mgmd_mld_static_groups": [],
    "name": "VLAN10",
    "type": "static"
  }
]
```


POST method usage and considerations

The POST method creates an instance of a resource in the collection specified by the URI:

- Not all resources support the POST method. See the AOS-CX REST API Reference for the methods supported by each resource. The REST API must be in read-write mode to see all the POST methods supported.
- Some resources support the POST method even when the REST API is in read-only mode.
- When you use the POST method, the URI must point to the collection—not to the resource you are creating. The resource you are creating is sent in JSON format in the request body.
  - The JSON representation must include all fields required by the JSON model of that resource.
  - The JSON representation can contain only configuration attributes. It must not contain attributes in the `status` or the `statistics` category.
- You can POST only one resource at a time.
- Most resources have a hierarchical relationship. You must create the parent before you can create the child.

For example, to create an ACL entry:

1. The ACL must be created first by sending the JSON data of the ACL in the request body in a POST request to the URI of the `acls` collection:
   `/system/acls`
2. The entry can then be created by sending the JSON data of the entry in the request body in a POST request to the URI of the ACL:
   `/system/acls/<ACL-name>,<ACL-type>/cfg_aces`

PUT method usage and considerations

The PUT method updates an instance of a resource by replacing the existing resource with the resource provided in the request body.

Configuration attributes that are set at the time a resource is created and that cannot be changed afterward are called **immutable** attributes. Configuration attributes that can be changed after a resource is created are called **mutable** or **writable** attributes. The PUT method is used replace writable attributes only.

- Not all resources support the PUT method. For information about the methods supported for a resource, see the AOS-CX REST API Reference. The REST API must be in `read-write` mode to see all the PUT methods supported.
- The URI must specify a specific resource, not a collection.
- The URI must specify a resource that currently exists.
- For almost all resources, the PUT method is implemented as a strict replace operation.

All mutable configuration attributes are replaced. Any mutable attribute that the JSON data in request body does not include is either removed (if there is no default value) or reset to its default value.

PUT request body requirements

The JSON data in the request body must include mutable (writable) configuration attributes only.

The JSON model used for the PUT method request body is different from the JSON model used for the GET or the POST method.

The JSON model of a PUT method for a resource contains the mutable attributes only. In contrast, the JSON models for GET and POST methods can include both mutable and immutable attributes.

See the AOS-CX REST API Reference for the JSON model of a PUT method for a resource.

PUT behavior

The PUT operation is a replace operation—not an update operation—because the resource instance in the request body replaces every changeable configuration attribute of the existing resource. PATCH partial updates are not supported.

Any mutable attribute that the JSON data in request body does not include is either removed (if there is no default value) or reset to its default value.

For example:

- If you attempt a PUT operation on the System resource to change the host name, and you supply only the host name, you will destabilize the switch because the other attributes will be reset to their defaults, which might be empty.

- If you attempt to change the name of a VLAN and supply only the name in the PUT request, every other attribute in that VLAN is set to its default of empty.

Exceptions to the PUT strict replace behavior

For Network Analytics Engine agents, the PUT behavior is not a strict replace implementation. You can enable or disable agents without the supplying the entire set of configuration attributes in the PUT request body. For more information about the Network Analytic Engine resources, see the *Network Analytics Engine Guide*.

Best practice for building the PUT request body

Hewlett Packard Enterprise recommends the following procedure for building the PUT request body.

Procedure

1. Use the GET method with `selector=writable` to obtain the writable (mutable) configuration attributes for the resource you want to change.

   For example:
   ```
   GET "https://192.0.2.5/rest/v1/system/interfaces/vlan200?selector=writable"
   ```
2. Change the values of the attributes to match your wanted configuration.

   Any attribute you do not include in the request body will be set to its default value, which could be empty.
3. Use the resulting JSON data as the request body for the PUT request.

DELETE method usage and considerations

The DELETE method deletes an instance of a resource.

- Not all resources support the DELETE method. See the AOS-CX REST API Reference for the methods supported by each resource. The REST API must be in `read-write` mode to see all the DELETE methods supported.
- The URI must specify a specific resource instance. The URI must not specify a collection.
- Child subcollections and resources are deleted when you delete the parent resource. For example, if you delete an ACL, its ACL entries are deleted automatically.
- DELETE requests do not contain a request body.
- DELETE requests do not return a response body.

REST requests and accounting logs

All REST requests—including GET requests—are logged to the accounting (audit) log.

The URI of the REST API resource for accounting logs is the following:

`/rest/v1/logs/audit`

In an accounting log entry for a REST request:

- `service=https-server` indicates that the log entry is a result of a REST API request or a Web UI action.
- The string value of `data` identifies the REST API request that was executed.

For more information about accounting log entries, see the description of the `show accounting log` CLI command.

AOS-CX REST API reference summary

The following information is intended as a quick reference for experienced users. Values are not configurable unless noted otherwise.

Switch REST API access default

8320, 8325, 8360, 8400, 9300, 10000 Switch Series: Disabled

6200, 6300, 6400 Switch Series: Enabled on the `mgmt` VRF

6000 and 6100 Switch Series: Enabled on the `default` VRF

Switch REST API default access mode

Read-write

Enabling access to the Web UI and REST API

CLI command:

`https-server vrf <VRF-NAME>`

Example:

```
switch(config)# https-server vrf mgmt
```

Setting the REST API access mode to read-write

CLI command:

`https-server rest access-mode read-write`

Example:

```
switch(config)# https-server rest access-mode read-write
```

Showing the REST API access configuration

CLI command:

`show https-server`

Example:

```
switch(config)# show https-server

HTTPS Server Configuration
--------------------------

 VRF               : default, mgmt
 REST Access Mode  : read-write
```

AOS-CX REST API Reference URL:

REST v10.04 API: `https://<IP-ADDR>/api/v10.04/`

REST v1 API: `https://<IP-ADDR>/api/v1` or `https://<IP-ADDR>/api/`

<IP-ADDR> is the IP address or hostname of your switch.

Example: `https://192.0.2.5/api/v1/`

REST API versions and switch software versions

REST API version	Switch software version
v10.04	AOS-CX 10.04 and later
v1	AOS-CX 10.00 and later

Getting REST API version information from a switch

Method and URI to get the REST API versions supported on the switch:
`GET "https://<IP-ADDR>/rest"`

<IP-ADDR> is the IP address or hostname of your switch.

Protocol

HTTPS

Port

443

Request and response body format

JSON

Session idle timeout

20 minutes

Session hard timeout

Eight hours, regardless of whether the session is active.

Authentication

Session cookie from successful HTTPS login request.

HTTPS client sessions

- Maximum of 48 sessions per switch.
- Maximum of six concurrent client sessions per user.
- The same session cookie is shared across browser tabs and, depending on the browser, multiple browser windows.
- The same session cookie is not shared across devices and scripts. For example, if a user logs into the Web UI from a laptop, again with a tablet, and then uses the same user name in a curl command, that user has three concurrent client sessions.

VSX peer switch access

If Virtual Switching Extension (VSX) is enabled on both switches, and the ISL is up, you can access the VSX peer switch from your connected switch. To access the peer VSX switch, insert `/vsx-peer` in the URI path between the server URL and `/rest`. Not supported for login, Web UI, or AOS-CX REST API Reference access. For more information about VSX, see VSX peer switches and REST API access.

For example:

- Accessing a VSX switch:

`https://192.0.2.5/rest/v1/…`

- Accessing its VSX peer switch:

`https://192.0.2.5/vsx-peer/rest/v1/…`

There are several tools available for accessing RESTful web service APIs, one of which is curl. The curl tool, created by the cURL project, is a command-line application for transferring data using URL syntax.

For details on installing the curl application, see https://curl.haxx.se/download.html.

The curl application has many options, which are described in detail in the curl manual (run `curl --manual`) and at https://curl.haxx.se/docs/manpage.html.

About the curl command examples

In the curl examples, the workstation is running a Linux-based operating system and curl version 7.35 is installed.

The curl examples generated by the AOS-CX REST API Reference might use different options than in other examples, and do not include cookie file handling because the cookie is handled by the browser.

Many examples of curl commands are formatted in multiple lines for readability. The backslash (`\`) continuation character at the end of the line indicates that the command continues on the next line.

The curl command examples in this document use minimal options. The following options are commonly used in the curl command examples:

`-b <cookie-file>`

Specifies that the file *<cookie-file>*, which contains the session cookie, be passed with the request. *<cookie-file>* specifies the path and name of the cookie file.

When you use curl, you log in at the beginning of your session and log out at the end of the session. When you log in, you must save the cookie returned from the login request. You must provide the cookie with every subsequent curl command.

`-k`

Specifies that the curl program not attempt to verify the server certificate against the list of certificate authorities included with the curl software.

The switch uses self-signed certificates. By default, the curl program attempts to verify certificates against its list of certificate authorities, and attempts to verify self-signed certificates will fail. Therefore you must use the `-k` option to disable attempts to verify self-signed certificates against a certificate authority.

`--noproxy`

Specifies that a web proxy is not required. The `--noproxy` option is appropriate where execution of curl commands does not need a proxy to access the applications.

If your network is configured to require a proxy to access applications, use the `--proxy` option instead of the `--noproxy` option.

`-d '<string>'`

Specifies that curl send the data in *<string>* in a POST request using the content-type application/x-www-form-urlencoded.

`-X`

Specifies a method that curl would not use by default. Typically used with PUT, DELETE, and POST methods only.

`-H` or `--header <header>`

Specifies an extra header in the HTTP request.

`-D`

Specifies that curl write the returned protocol headers to the standard output file. Used for debugging.

More options can be used to customize your experience for your environment. For more information about curl options, see:

https://curl.haxx.se/docs/manpage.html

Getting the REST API versions on the switch

To get information about the latest and all available REST API versions on a switch, execute a GET request to the following URI:

`"https://<IP-ADDR>/rest"`

<IP-ADDR> is the IP address or hostname of your switch.

Example method and URI:

```
GET "https://192.0.2.5/rest"
```

Example curl command:

```
$ curl -k GET \
-b /tmp/auth_cookie \
"https://192.0.2.5/rest"
```

Example response body:

```
{
  "latest": {
    "version": "v10.04",
    "prefix": "/rest/v10.04",
    "doc": "/api/v10.04"
  },
  "v10.04": {
    "version": "v10.04",
    "prefix": "/rest/v10.04",
    "doc": "/api/v10.04"
  },
  "v1": {
    "version": "v1",
    "prefix": "/rest/v1",
    "doc": "/api/v1"
  }
}
```

Accessing the REST API using curl

When you use curl, you log in at the beginning of your session and log out at the end of the session. When you log in, you must save the cookie returned from the login request so that you can pass that same cookie value to the switch in subsequent curl commands.

Prerequisites

- Access to the switch REST API must be enabled.

Procedure

1. To access the AOS-CX REST API using curl, use curl version 7.35 or later. The examples provided in this document are tested with version 7.35.
2. For all curl commands, use the `-k` option to disable certificate validation.

   The switch uses self-signed certificates. By default, the curl program attempts to verify certificates against its list of certificate authorities, and attempts to verify self-signed certificates fail. Therefore you must use the `-k` option to disable attempts to verify self-signed certificates against a certificate authority.
3. Start your session by logging in. When you log in, save the cookie file by specifying the `-c` option with a file name.
4. In all subsequent curl commands—including logging out—pass the cookie value back to the switch by specifying the `-b` option with the same file name.
5. At the end of the session, log out of the switch using curl.

Logging out at the end of the session is important because the number of concurrent HTTPS sessions per client and per switch are limited, and session cookies are not shared across devices and scripts.

Logging in using curl

Prerequisites

- Access to the switch REST API must be enabled.

Credential information (user name, password, domain, and authentication tokens) used in curl commands entered at a command-line prompt might be saved in the command history. For security reasons, Hewlett Packard Enterprise recommends that you disable command history before executing commands containing credential information.

Procedure

Use the following curl command to access the `login` resource of the switch and provide your user name and password as data:

Syntax:

```
curl [--noproxy <IP-ADDR>] -k -X POST
-c <COOKIE-FILE>
-H 'Content-Type: multipart/form-data'
"https://<IP-ADDR>/rest/v1/login"
-F 'username=<USER-NAME>' -F 'password=<PASSWORD>'
```

Options:

`-k`

Specifies that the curl program not attempt to verify the server certificate against the list of certificate authorities included with the curl software.

The switch uses self-signed certificates. By default, the curl program attempts to verify certificates against its list of certificate authorities, and attempts to verify self-signed certificates fail. Therefore you must use the `-k` option to disable attempts to verify self-signed certificates against a certificate authority.

`-X`

Specifies a method that curl would not use by default. Typically, used only with POST, PUT, or DELETE methods.

`--noproxy <IP-ADDR>`

Optional. The `--noproxy` option is appropriate where execution of curl commands does not need a proxy to access the applications. If your network is configured to require a proxy to access applications, use the `--proxy` option. *<IP-ADDR>* specifies the IP address or hostname of the switch.

`-C <COOKIE-FILE>`

Specifies the file in which to store the session cookie. This session cookie is required when you execute subsequent curl commands.

`-H` or `--header <header>`

Specifies an extra header in the HTTP request.

`-F`

Specifies that the curl command will emulate a filled-in form in which a user has pressed the submit button for the HTTP protocol family. This causes curl to POST data using the Content-Type multipart/form-data.

<USER-NAME>

Specifies the user name.

<PASSWORD>

Specifies the password for the user.

Although it is possible to pass the user name and password information as a query string in the login URI, system logs save the accessed URI in cleartext in log entries. Hewlett Packard Enterprise recommends that you pass the credential information as data instead of in the URI when using programs such as curl to log in to the switch.

Example:

```
$ curl --noproxy "192.0.2.5" -k -X POST \
-c /tmp/auth_cookie \
-H 'Content-Type: multipart/form-data' \
"https://192.0.2.5/rest/v1/login" \
-F 'username=test' -F 'password=test'
```

Passing the cookie back to the switch

Prerequisites

Start a session by logging in to the REST API and save the cookie file.

Procedure

Use the following curl command to pass the cookie file back to the switch using the -b option.

Syntax:

```
curl [--noproxy <IP-ADDR>] -k GET
-b <COOKIE-FILE>
-H 'Content-Type:application/json'
-H 'Accept: application/json'
"https://<IP-ADDR>/rest/v1/system"
```

Options:

`--noproxy <IP-ADDR>`

Optional. The --noproxy option is appropriate where execution of curl commands does not need a proxy to access the applications. If your network is configured to require a proxy to access applications, use the --proxy option. <IP-ADDR> specifies the IP address or hostname of the switch.

`-k`

Specifies that the curl program not attempt to verify the server certificate against the list of certificate authorities included with the curl software.

The switch uses self-signed certificates. By default, the curl program attempts to verify certificates against its list of certificate authorities, and attempts to verify self-signed certificates fail. Therefore you must use the –k option to disable attempts to verify self-signed certificates against a certificate authority.

`-b <COOKIE-FILE>`

Specifies that the file <cookie-file>, which contains the session cookie, be passed with the request. The <cookie-file> specifies the path and name of the cookie file.

When you use curl, you log in at the beginning of your session and log out at the end of the session. When you log in, you must save the cookie returned from the login request. You must provide the cookie with every subsequent curl command.

`-H` or `--header <header>`

Specifies an extra header in the HTTP request.

Example:

```
$ curl --noproxy -k GET
-b /tmp/auth_cookie \
-H 'Content-Type:application/json' \
-H 'Accept: application/json' \
"https://192.0.2.5/rest//system"
```

Logging out using curl

Procedure

Use the following curl command to access the `logout` resource of the switch:

Syntax:

`curl [--noproxy <IP-ADDR>] -k -X POST`
`-b <COOKIE-FILE>`
`"https://<IP-ADDR>/rest/v1/logout"`

Options:

`-k`

Specifies that the curl program not attempt to verify the server certificate against the list of certificate authorities included with the curl software.

The switch uses self-signed certificates. By default, the curl program attempts to verify certificates against its list of certificate authorities, and attempts to verify self-signed certificates fail. Therefore you must use the `-k` option to disable attempts to verify self-signed certificates against a certificate authority.

`--noproxy <IP-ADDR>`

Optional. The `--noproxy` option is appropriate where execution of curl commands does not need a proxy to access the applications. If your network is configured to require a proxy to access applications, use the `--proxy` option. `<IP-ADDR>` specifies the IP address or hostname of the switch.

`-b <COOKIE-FILE>`

Specifies the file that contains the session cookie.

When you use curl, you log in at the beginning of your session and log out at the end of the session. When you log in, you must save the cookie returned from the login request so that you can pass that same cookie value to the switch in subsequent curl commands. When you log in, save the cookie file by specifying the `-c` option with a file name.

In subsequent curl commands, pass the cookie value back to the switch by specifying the `-b` option with the same file name.

`-X`

Specifies a method that curl would not use by default. Typically, used only with POST, PUT, or DELETE methods.

Example:

```
$ curl --noproxy "192.0.2.5" -k -X POST \
-b /tmp/auth_cookie \
"https://192.0.2.5/rest/v1/logout"
```

Examples

The following examples show how you can use curl with AOS-CX REST API. The response body might vary based on the AOS-CX switch series. For example, the 8320 and 6400 switches show VSX information, whereas the 6300 and 6200 switches show VSF and PoE information.

As a best practice, before you perform a GET, PUT, POST, or DELETE operation, you must login to create a session and save the cookie file by specifying the `-c` option with a file name. After you perform the operation, you must logout to end the session and pass the cookie file back to the switch by specifying the `-b` option with the same file name. The following examples assume that you are performing the step to login before performing the operations in the example and logout after performing the operations. For more information, see Accessing the REST API using curl.

The request and response body in the following examples contain a truncated part of the actual data. The data that you see after running the curl commands might vary. Ellipses (...) in the response body represent data not shown in the example.

Example: GET method

Instructions and examples in this document use an IP address that is reserved for documentation, 192.0.2.5, as an example of the IP address for the switch. To access your switch, you must use the IP address or hostname of that switch.

- Get the list of all VLANS:

```
GET "https://192.0.2.5/rest/v1/system/vlans"
```

- Expand the list of URIs in the `vlans` collection by one level, which replaces the URI for the VLAN with the JSON data for that VLAN.

```
GET "https://192.0.2.5/rest/v1/system/vlans?depth=1"
```

- Use the `count` parameter to get the total number of VLANs:

```
GET "https://192.0.2.5/rest/v1/system/vlans?count"
```

- Use the `count` parameter with the `filter` parameter to get the total number of interfaces in a `down` administrative state:

```
GET "https://192.0.2.5/rest/v1/system/interfaces?count=true&filter=admin_state:down"
```

- Use the `filter` parameter with the value `type:static` to get a list of only the static VLANs:

```
GET "https://192.0.2.5/rest/v1/system/vlans?filter=type:static"
```

- Use the `filter` parameter to get the BGP routes that have 1.1.1.1 as a peer:

```
GET "https://192.0.2.5/rest/v1/system/vrfs/default/bgp_routes?filter=peer:1.1.1.1"
```

- Use the `attributes` parameter to get all ports but show only the attributes `name` and `ipv4_address`:

```
GET "https://192.0.2.5/rest/v1/system/ports?attributes=name,ipv4_address"
```

- Use the wildcard character to get a list of routes for all VRFs.

```
GET "https://192.0.2.5/rest/v1/system/vrfs/*/routes"
```

- Use the `selector` parameter to get all the configuration attributes of VLAN 100:

```
GET "https://192.0.2.5/rest/v1/system/vlans/100?selector=configuration"
```

- Use the `selector` parameter to get all the system attributes that are in the categories configuration and status:

```
GET "https://192.0.2.5/rest/v1/system?selector=category,status"
```

Example: Getting and deleting certificates using REST APIs

Getting a list of all certificates

It is important to note that the certificate resources do not support the use of internationalized strings. Since UTF8 is the only supported encoding, a Subject Alternative Name (SAN) must be used instead.

Example method and URI:

```
GET "https://192.0.2.5/rest/v1/certificates"
```

Example curl command:

```
$ curl --noproxy 192.0.2.5 -k GET \
-b /tmp/auth_cookie \
"https://192.0.2.5/rest/v1/certificates”
```

On successful completion, the switch returns response code 200 OK and a response body containing the certificate resource URLs indexed by the certificate name. For example:

```
{
  "my-cert-1": "/rest/v1/certificates/my-cert-1",
  "my-cert-2": "/rest/v1/certificates/my-cert-2"
}
```

Getting a certificate

Example method and URI:

```
GET "https://192.0.2.5/rest/v1/certificates/my-cert-2"
```

Example curl command:

```
$ curl --noproxy 192.0.2.5 -k GET \
-b /tmp/auth_cookie \
"https://192.0.2.5/rest/v1/certificates/my-cert-2"
```

On successful completion, the switch returns response code 200 OK and a response body containing the certificate.

For example:

```
'{
  "cert_name": "my-cert-2",
  "cert_type": "regular"
  "cert_status": "csr_pending",
  "key_type": "RSA",
  "key_size": 2048,
  "subject": {
    "common_name": "CX-8400",
    "country": "US",
    "locality: "el camino",
    "state": "CA",
    "org": "HPE",
    "org_unit": "Aruba"
    },
  "certificate": "<certificate-in-PEM-format>"
}'
```

Deleting a certificate

Example method and URI:

```
DELETE "https://192.0.2.5/rest/v1/certificates/my-cert-3"
```

Example curl command:

```
$ curl --noproxy 192.0.2.5 -k -X DELETE \
-b /tmp/auth_cookie \
"https://192.0.2.5/rest/v1/certificates/my-cert-3"
```

On successful completion, the switch returns response code 204.

Example: Generating a self-signed certificate using REST APIs

The following example generates a self-signed certificate.

Example method and URI:

```
POST "https://192.0.2.5/rest/v1/certificates"
```

Example request body:

```
{
...
  "certificate_name": "my-cert-1",
  "subject": {
    "country": "US",
    "state": "CA",
    "org": "HPE",
    "org_unit": "Aruba",
    "common_name": "CX-8400"},
  "key_type": "RSA",
  "key_size": 2048,
  "cert_type": "self-signed"
...
}
```

Example curl command:

```
$ curl --noproxy 192.0.2.5 -k -X POST \
-b /tmp/auth_cookie \
"https://192.0.2.5/rest/v1/certificates”
-d '{
...
  "certificate_name": "my-cert-1",
```

```
  "subject": {
    "country": "US",
    "state": "CA",
    "org": "HPE",
    "org_unit": "Aruba",
    "common_name": "CX-8400"},
  "key_type": "RSA",
  "key_size": 2048,
  "cert_type": "self-signed"
...
}'
```

On successful completion, the switch returns response code 201 Created.

Example: Getting and installing a signed leaf certificate using REST APIs

This example includes the step to create a trust anchor (TA) profile. If the TA profile had previously been configured, that step of the example would be skipped. The TA profile is used to validate the signed certificate when you import the certificate as part of the PUT request.

For more information about certificates and certificate management, see the *Security Guide*.

Procedure

1. Create a TA profile:
   a. From the certificate authority (CA), get a copy of the certificate against which you will validate leaf certificates.

      The certificate you validate leaf certificates against can be a root certificate or an intermediate certificate.

      The steps to get the leaf certificate depend on the CA and the operating system you use.
   b. Create a JSON object with a `certificate` key and a `name` key.

      For example:

      ```
      {
        "name": "<profile-name>",
        "certificate": "<root-ca-cert>"
      }
      ```

      - For the value of the `name` key, replace *<profile-name>* with the name of the TA profile you want to create.
      - For the value of the `certificate` key, replace *<root-ca-cert>* by pasting the copied certificate. After pasting, edit the text to ensure proper loading as a JSON object by doing the following:
        - Ensure the certificate headers and footers are treated as separate lines by adding `\n` characters after the header and before the footer.

          The following example shows the `\n` characters in bold.

          ```
          {
            "name": "myta",
            "certificate": "-----BEGIN CERTIFICATE-----\nMIIF2DCCA8CgAwIBAgIlCnL

          MA0GCSqGSIb3DQEBCwUAMHkxCzAJBgNVBAYTAkdCMRAwDgYDVQQIDAdFbmdsYW5kMRIwEAYDVQD
          Al
          ...
          ```

```
PKj0FmJ1+Qzw9Bcm6HiPTyxOVozMeRQzSQhTZVlh3OvBw/cUwTIqFJCe/afNQCqa9XnvTpJvP/
Q3z
...
S4L9sxrk/i3hKB88\n-----END CERTIFICATE-----"
}
```

- Ensure that any private key headers and footers are treated as separate lines by adding `\n` characters before and after them as needed.

  For example:

```
\n-----BEGIN PRIVATE KEY-----\n
MIIFDjBABgkqhkiG9wBBQ0wMzAbBgqkw0QwwDQIpJMN7sVGwCAggA
...
iKnXnUMpVPfLc74ty2S41DtH0X9gf6aa1jStg+7cND9XfGtjaV2CA
\n-----END PRIVATE KEY-----\n
\n-----BEGIN ENCRYPTED PRIVATE KEY-----\n
IJ6L/UhEtH523nUkdV6gvAgoYaD83PswToAGv5VS8OMFTPttrn5/K
...
OgSecqZsG6arbx0ESaYBir1c/6rPspcjbx283iD1MWOpeoS2aEmOX=
\n-----END ENCRYPTED PRIVATE KEY-----\n
```

c. Use the POST method to create the TA profile with the copied certificate. Include the JSON object in the request body:

Example method and URI:

```
POST "https://192.0.2.5/rest/v1/system/pki_ta_profiles"
```

Example curl commands:

```
$ curl --noproxy 192.0.2.5 -k -X POST \
-b /tmp/primary_auth_cookie \
-H 'Content-Type:application/json'
"https://192.0.2.5/rest/v10.04/system/pki_ta_profiles"
-d '{
  "name": "myta",
  "certificate": "-----BEGIN CERTIFICATE-----
\nMIIF2DCCA8CgAwIBAgIJANkWgud1lCnL

MA0GCSqGSIb3DQEBCwUAMHkxCzAJBgNVBAYTAkdCMRAwDgYDVQQIDAdFbmdsYW5kMRIwEAYD
VQQKDAl
...

PKj0FmJ1+Qzw9Bcm6HiPTyxOVozMeRQzSQhTZVlh3OvBw/cUwTIqFJCe/afNQCqa9XnvTpJv
P/Q3ze6
S4L9sxrk/i3hKB88\n-----END CERTIFICATE-----"
}'
```

On successful completion, the switch returns response code 201 Created.

2. Create a certificate with a pending certificate signing request (CSR).

For information about the required and optional items in the request body, see the JSON model for the `certificates` resource in the AOS-CX REST API Reference.

Example method and URI:

```
POST "https://192.0.2.5/rest/v1/certificates"
```

Example request body:

```
{
  "certificate_name": "my-cert-name",
  "subject": {
    "common_name": "CX-8400"
    "country": "US",
    "locality":"el camino",
```

```
    "state": "CA",
    "org": "HPE",
    "org_unit": "Aruba",
    },
  "key_type": "RSA",
  "key_size": 2048,
  "cert_type": "regular"
}
```

Example curl command:

```
$ curl --noproxy 192.0.2.5 -k -X POST \
-b /tmp/primary_auth_cookie \
-d '{
  "certificate_name": "my-cert-name",
  "subject": {
    "common_name": "CX-8400"
    "country": "US",
    "locality":"el camino",
    "state": "CA",
    "org": "HPE",
    "org_unit": "Aruba",
    },
  "key_type": "RSA",
  "key_size": 2048,
  "cert_type": "regular"
}'
"https://192.0.2.5/rest/v1/certificates"
```

On successful completion, the switch returns response code 201 Created.

3. Get the certificate you created in the previous step.

   Example method and URI:
   `GET "https://192.0.2.5/rest/v1/certificates/my-cert-name"`

   Example curl command:

   ```
   $ curl --noproxy 192.0.2.5 -k GET \
   -b /tmp/primary_auth_cookie \
   "https://192.0.2.5/rest/v1/certificates/my-cert-name"
   ```

   On successful completion, the switch returns response code 200 OK and a response body containing the CSR in PEM format.

4. Send the CSR to the CA for signing.

   The steps to send the CSR depend on the CA and the operating system you use.

   The CA returns the signed certificate in PEM format.

5. Import the signed certificate by using a PUT request to update the `my-cert-name` certificate with the signed certificate you received from the CA.

The imported certificate data must include all the intermediate CA certificates in the certificate chain leading to the certificate that was imported into the specified TA profile.

If you copy and paste the certificate into a JSON object, you must ensure that the certificate and private key headers and footers are processed as separate lines by editing the text to add `\n` characters as needed.

As part of the PUT request, the switch attempts to validate the certificate against the pool of all TA profiles installed on the switch. The certificate is accepted if it is validated with one of the TA profiles.

Example method and URI:

```
PUT "https://192.0.2.5/rest/v1/certificates/my-cert-name"
```

Example request body:

```
{
  "certificate": "-----BEGIN CERTIFICATE-----\n
MIIFRDCCAyygAwIBAgQP8nS2Vp15u0xXMdkDJzANBgkqhkiG9w0Bv
...
1NGNm3NG03GqPScs/TF9bVyFA5BOS5lmmkfRYK8D/kMTfRreSdxis
YQ1u1NqShps=
\n-----END CERTIFICATE-----\n
\n-----BEGIN ENCRYPTED PRIVATE KEY-----\n
MIIFDjBABgkqhkiG9wBBQ0wMzAbBgqkw0QwwDQIpJMN7sVGwCAggA
...
iKnXnUMpVPfLc74ty2S41DtH0X9gf6aa1jStg+7cND9XfGtjaV2+/
cb4=
\n-----END ENCRYPTED PRIVATE KEY-----"
}
```

Example curl commands:

```
$ curl --noproxy 192.0.2.5 -k -X PUT \
-b /tmp/primary_auth_cookie \
-d '{
  "certificate": "-----BEGIN CERTIFICATE-----\n
MIIFRDCCAyygAwIBAgQP8nS2Vp15u0xXMdkDJzANBgkqhkiG9w0Bv
...
1NGNm3NG03GqPScs/TF9bVyFA5BOS5lmmkfRYK8D/kMTfRreSdxis
YQ1u1NqShps=
\n-----END CERTIFICATE-----\n
\n-----BEGIN ENCRYPTED PRIVATE KEY-----\n
MIIFDjBABgkqhkiG9wBBQ0wMzAbBgqkw0QwwDQIpJMN7sVGwCAggA
...
iKnXnUMpVPfLc74ty2S41DtH0X9gf6aa1jStg+7cND9XfGtjaV2+/
cb4=
\n-----END ENCRYPTED PRIVATE KEY-----"
}'
"https://192.0.2.5/rest/v1/certificates/my-cert-name"
```

On successful completion, the switch returns response code 200 OK.

The certificate is installed and ready to be associated with switch features.

Example: Associating a leaf certificate with a switch feature using REST APIs

The following example associates the signed certificate `my-cert-name` with the HTTPS server switch feature. For complete information about the switch features to which you can associate a leaf certificate, see the *Security Guide*.

1. Get the configuration attributes of the `system` resource:

   Example method and URI:

   ```
   GET "https://192.0.2.5/rest/v1/system?selector=configuration"
   ```

   Example curl command:

   ```
   $ curl --noproxy 192.0.2.5 -k GET \
   -b /tmp/primary_auth_cookie \
   "https://192.0.2.5/rest/v1/system?selector=configuration"
   ```

On successful completion, the switch returns response code 200 and a JSON object containing the configuration attributes.

2. In the portion of the response body that defines the certificate name for the HTTPS server, change the value to: `my-cert-name`.

The certificate name associated with the HTTPS server is the value assigned to the `https-server` key, which is under the `certificate_association` key. By default, the certificate name is: `local-cert`.

The request body of a PUT request is permitted to include only the mutable configuration attributes. In the AOS-CX software releases to which this example applies, all the configuration attributes for the `system` resource are mutable attributes, so you do not need to edit the JSON object to remove the immutable attributes.

3. Using a PUT request, update the system resource with the edited JSON data as the request body.

   Example method and URI:
   `PUT "https://192.0.2.5/rest/v1/system"`

   Example request body:

```
{
    "aaa": {
...
    },
...
    "certificate_association": {
        "https-server": "my-cert-name",
        "syslog-client": "local-cert"
    },
...
}
```

   Example curl command:

```
$ curl --noproxy 192.0.2.5 -k -X PUT \
-b /tmp/primary_auth_cookie \
-d '{
    "aaa": {
...
    },
...
    "certificate_association": {
        "https-server": "my-cert-name",
        "syslog-client": "local-cert"
    },
...
}'
"https://192.0.2.5/rest/v1/system"
```

   On successful completion, the switch returns response code 200 OK.

Example: Configuration management using REST APIs

Downloading a configuration

Downloading the current configuration:

- Example method and URI:
  GET "https://192.0.2.5/rest/v1/fullconfigs/running-config"

- Example curl command:

```
$ curl --noproxy 192.0.2.5 -k GET \
-b /tmp/primary_auth_cookie \
"https://192.0.2.5/rest/v1/fullconfigs/running-config"
```

Downloading the startup configuration:

- Example method and URI:
  `GET "https://192.0.2.5/rest/v10.04/fullconfigs/startup-config"`
- Example curl command:

```
$ curl --noproxy 192.0.2.5 -k GET \
-b /tmp/primary_auth_cookie \
"https://192.0.2.5/rest/v1/fullconfigs/startup-config"
```

On successful completion, the switch returns response code 200 OK and a response body containing the entire configuration in JSON format.

Uploading a configuration

The following example shows uploading a configuration to become the running configuration. The running configuration is the only configuration that can be updated using this technique, however, you can copy other configurations. For more information about copying configurations, see Copying a configuration.

- Example method and URI:
  `PUT "https://192.0.2.5/rest/v10.04/fullconfigs/running-config"`

The request body must contain the configuration—in JSON format—to be uploaded.

- Example curl command:

```
$ curl --noproxy 192.0.2.5 -k -X PUT \
-b /tmp/auth_cookie \
"https://192.0.2.5/rest/v1/fullconfigs/running-config" \
-d '{
…
}'
```

The configuration being uploaded—represented as ellipsis but not shown in this example—is in JSON format in the body of the command (enclosed in braces).

On successful completion, the switch returns response code 200 OK.

Copying a configuration

To replace an existing configuration with another, use a REST PUT request to the destination configuration. Use the `from` query string parameter to specify the source configuration.

- At least one of the source or the destination configuration must be either `running-config` or `startup-config`. You cannot copy a checkpoint to a different checkpoint.
- If you specify a destination checkpoint that exists, an error is returned. You cannot overwrite an existing checkpoint.

The syntax of the method and URI is as follows:

```
PUT "https://<IPADDR>/rest/v1/fullconfigs/<destination-config>?
from=/rest/v1/fullconfigs/<source-config>"
```

Copying the running configuration to the startup configuration:

- Example method and URI:
  ```
  PUT "https://192.0.2.5/rest/v1/fullconfigs/startup-config?
  from=/rest/v1/fullconfigs/running-config"
  ```
- Example curl command:

```
$ curl --noproxy 192.0.2.5 -k -X PUT \
-b /tmp/auth_cookie -D-
"https://192.0.2.5/rest/v1/fullconfigs/startup-config?
from=/rest/v1/fullconfigs/running-config"
```

Copying the startup configuration to the running configuration:

- Example method and URI:
  ```
  PUT "https://192.0.2.5/rest/v1/fullconfigs/running-config?
  from=/rest/v1/fullconfigs/startup-config"
  ```
- Example curl command:

```
$ curl --noproxy 192.0.2.5 -k -X PUT \
-b /tmp/auth_cookie -D-
"https://192.0.2.5/rest/v1/fullconfigs/running-config?
from=/rest/v1/fullconfigs/startup-config"
```

Copying a checkpoint to the running configuration:

- Example method and URI:
  ```
  PUT "https://192.0.2.5/rest/v1/fullconfigs/running-config?
  from=/rest/v1/fullconfigs/MyCheckpoint"
  ```
- Example curl command:

```
$ curl --noproxy 192.0.2.5 -k -X PUT \
-b /tmp/auth_cookie -D-
"https://192.0.2.5/rest/v1/fullconfigs/running-config?
from=/rest/v1/fullconfigs/MyCheckpoint"
```

Copying the running configuration to a checkpoint:

- Example method and URI:
  ```
  PUT "https://192.0.2.5/rest/v1/fullconfigs/MyCheckpoint?
  from=/rest/v1/fullconfigs/running-config"
  ```
- Example curl command:

```
$ curl --noproxy 192.0.2.5 -k -X PUT \
-b /tmp/auth_cookie -D-
"https://192.0.2.5/rest/v1/fullconfigs/MyCheckpoint?
from=/rest/v1/fullconfigs/running-config"
```

Example: Firmware upgrade using REST APIs

Uploading a file as the secondary firmware image

In the following example, a curl command is used to upload the firmware image file from the local workstation to the switch, as the secondary firmware image. The `-F` option specifies that the POST method is used to upload the file.

Example method and URI:

```
POST "https://192.0.2.5/rest/v1/firmware?image=secondary"
```

The request body contains the switch firmware image file in binary format.

Example curl command:

```
$ curl --noproxy -k -b /tmp/auth_cookie \
-H 'Content-Type: application/json' \
-H 'Accept: application/json' \
-F "fileupload=@/myfirmwarefiles/myswitchfirmware_2020020905.swi" \
https://192.0.2.5/rest/v1/firmware?image=secondary
```

In the curl command, the POST request is handled as part of the `-F` option.

Booting the system using the secondary firmware image

Example method and URI:

```
POST "https://192.0.2.5/rest/v1/boot?image=secondary"
```

Example curl command:

```
$ curl --noproxy -k -X POST -b /tmp/auth_cookie \
-H 'Content-Type: application/json' \
-H 'Accept: application/json' \
"https://192.0.2.5/rest/v1/boot?image=secondary"
```

Example: Log operations using REST APIs

Event logs

A GET request to `/rest/v1/logs/event` URI returns all entries from all the event logs on the switch, including logs from internal processes.

The information returned by this request was not optimized for human readability. If you want to examine the log entries, Hewlett Packard Enterprise recommends that you use the Web UI. The Web UI also provides a method to export log entries.

In the following example, the `MESSAGE_ID` parameter filters the output to include event log messages only:

- `50c0fa81c2a545ec982a54293f1b1945` identifies event log messages from the active management module.
- `73d7a43eaf714f97bbdf2b251b21cade` identifies event log messages from the standby management module. Not all switches have a standby management module.

Example method and URI:

```
GET "https://192.0.2.5/rest/v1/logs/event?
limit=1000&
priority=4&
since=24%20hour%20ago&
MESSAGE_ID=50c0fa81c2a545ec982a54293f1b1945,73d7a43eaf714f97bbdf2b251b21cade"
```

Example curl command:

```
$ curl --noproxy 192.0.2.5 -k GET \
-b /tmp/primary_auth_cookie \
"https://192.0.2.5/rest/v1/logs/event?
limit=1000&
priority=4&
since=24%20hour%20ago&
MESSAGE_ID=50c0fa81c2a545ec982a54293f1b1945,73d7a43eaf714f97bbdf2b251b21cade"
```

Accounting (audit) logs

A GET request to the `/rest/v1/logs/audit` URI returns all entries from the accounting logs on the switch.

For a list of supported query parameters, see the AOS-CX REST API Reference.

Example method and URI:

```
GET "https://192.0.2.5/rest/v1/logs/audit?
since=24%20hour%20ago&
usergroup=administrators&
session=CLI"
```

Example curl command:

```
$ curl --noproxy 192.0.2.5 -k GET \
-b /tmp/primary_auth_cookie \
"https://192.0.2.5/rest/v1/logs/audit?
since=24%20hour%20ago&
usergroup=administrators&
session=CLI"
```

Example: Ping operations using REST APIs

This example gets ping statistics for the ping target.

Example method and URI:

```
GET "https://192.0.2.5/rest/v1/ping?
ping_target=192.0.2.10&
is_ipv4=true&
ping_data_size=100&
ping_time_out=2&
ping_repetitions=1&
ping_type_of_service=0&
include_time_stamp=false&
include_time_stamp_address=false&
record_route=false&
mgmt=false"
```

Example curl command:

```
$ curl --noproxy 192.0.2.5 -k GET \
-b /tmp/primary_auth_cookie \
"https://192.0.2.5/rest/v1/ping?
ping_target=192.0.2.10&
is_ipv4=true&
ping_data_size=100&
ping_time_out=2&
ping_repetitions=1&
ping_type_of_service=0&
include_time_stamp=false&
include_time_stamp_address=false&
```

```
record_route=false&
mgmt=false"
```

On successful completion, the switch returns response code 200 OK and a response body containing the output string produced by the ping operation.

Example: Traceroute operations using REST APIs

Example method and URI:

```
GET "https://192.0.2.5/rest/v1/traceroute?
ip=192.0.2.10&
is_ipv4=true&
timeout=3&
destination_port=33434&
max_ttl=30&
min_ttl=1&
probes=3&
mgmt=false"
```

Example curl command:

```
$ curl --noproxy 192.0.2.5 -k GET \
-b /tmp/primary_auth_cookie \
"https://192.0.2.5/rest/v1/traceroute?
ip=192.0.2.10&
is_ipv4=true&
timeout=3&
destination_port=33434&
max_ttl=30&
min_ttl=1&
probes=3&
mgmt=false"
```

On successful completion, the switch returns response code 200 OK and a response body containing the output string produced by the traceroute operation.

Example: User management using REST APIs

Creating a user

Example method and URI:

```
POST "https://192.0.2.5/rest/v1/system/users"
```

Example request body:

```
{
...
  "name": "myadmin",
  "password": "P@ssw0rd",
  "user_group": "/rest/v1/system/user_groups/administrators",
  "origin": "configuration"
...
}
```

Example curl command:

```
$ curl --noproxy -k -X POST \
-b /tmp/auth_cookie \
"https://192.0.2.5/rest/v1/system/users”
```

```
-d '{
...
  "name": "myadmin",
  "password": "P@ssw0rd",
  "user_group": "/rest/v1/system/user_groups/administrators",
  "origin": "configuration"
...
}'
```

On successful completion, the switch returns response code 201 Created.

Changing a password

Example method and URI:

```
PUT "https://192.0.2.5/rest/v1/system/users/myadmin"
```

Example request body:

```
{
  "password": "P@ssw0rd2g"
}
```

Example curl command:

```
$ curl --noproxy -k -X PUT \
-b /tmp/auth_cookie \
"https://192.0.2.5/rest/v1/system/users/myadmin”
-d '{
  "password": "P@ssw0rd2g"
}'
```

On successful completion, the switch returns response code 200 OK.

Deleting a user

Example method and URI:

```
DELETE "https://192.0.2.5/rest/v1/system/users/myadmin"
```

Example curl command:

```
$ curl --noproxy -k -X DELETE \
-b /tmp/auth_cookie \
"https://192.0.2.5/rest/v1/system/users/myadmin"
```

On successful completion, the switch returns response code 204 No Content.

Example: Creating an ACL with a port using REST APIs

This example shows creating the following ACL and port configuration on a switch at IP address 192.0.2.5:

```
interface 1/1/2
    no shutdown
    apply access-list ip ACLv4 out
access-list ip ACLv4
    10 permit tcp 10.0.100.101 eq 80 10.0.100.102 eq 8000
```

1. Creating the ACL.

```
$ curl --noproxy 192.0.2.5 -k -X POST \
-b /tmp/auth_cookie -d '{
"cfg_version": 0,
"list_type": "ipv4",
"name": "ACLv4"}'
"https://192.0.2.5/rest/v1/system/acls"
```

2. Creating an ACL entry.

```
$ curl --noproxy 192.0.2.5 -k -X POST \
-b /tmp/auth_cookie -d
'{
...
"action": "permit",
"dst_ip": "10.0.100.102/255.255.255.255",
"dst_l4_port_max": 8000,
"dst_l4_port_min": 8000,
"protocol": 6,
"sequence_number": 10,
"src_ip": "10.0.100.101/255.255.255.255",
"src_l4_port_max": 80,
"src_l4_port_min": 80
...
}'
"https://192.0.2.5/rest/v1/system/acls/ACLv4/ipv4/cfg_aces"
```

3. Getting the ACL configuration information to use in the next step. Ellipses (...) represent data not shown in the example.

```
$ curl --noproxy 192.0.2.5 -k GET \
-b /tmp/auth_cookie \
"https://192.0.2.5/rest/v1/system/acls/ACLv4/ipv4?selector=configuration"
{
...
  "cfg_aces": {},
  "cfg_version": 0
...
  "list_type": "ipv4",
  "name": "ACLv4"
...
}
```

4. Updating the ACL configuration using the return body received from the GET request performed in the previous step.

When you send a PUT request, the JSON request body must not contain immutable attributes. The AOS-CX REST API Reference model for the PUT method of the resource shows the mutable attributes. Any mutable attributes you do not include in the PUT request body are set to their defaults, which could be empty.

The AOS-CX REST API Reference JSON model for the PUT method of the `/system/acls/{id1}/{id2}` resource shows the following example:

```
{
  "cfg_aces": {
```

```
    "integer": "URL"
  },
  "cfg_version": 0
}
```

The following example shows the request to update the ACL configuration:

```
$ curl --noproxy 192.0.2.5 -k -X PUT \
-b /tmp/auth_cookie -d '{
"cfg_aces":{"10":"/rest/v1/system/acls/ACLv4/ipv4/cfg_aces/10"},
"cfg_version":1}' \
"https://192.0.2.5/rest/v1/system/acls/ACLv4/ipv4"
```

5. Creating port 1/1/2.

```
$ curl --noproxy 192.0.2.5 -k -X POST \
-b /tmp/auth_cookie -d '{
"name": "1/1/2",
"admin":"up",
"interfaces":["/rest/v1/system/interfaces/1%2F1%2F2"],
"vrf":"/rest/v1/system/vrfs/default"}' \
"https://192.0.2.5/rest/v1/system/ports"
```

6. Getting the configuration information for the interface.

   The GET response body includes only the configuration attributes that have been set.

```
$ curl --noproxy 192.0.2.5 -k GET \
-b /tmp/auth_cookie \

"https://192.0.2.5/rest/v1/system/interfaces/1%2F1%2F2?selector=configuratio
n"
{
...
  "options": {},
  "other_config": {},
  "udld_arubaos_compatibility_mode": "forward_then_verify",
  "udld_compatibility": "aruba_os",
  "udld_enable": false,
  "udld_interval": 7000,
  "udld_retries": 4,
  "udld_rfc5171_compatibility_mode": "normal",
  "user_config": {}
...
}
```

7. Verifying which configuration attributes are mutable and therefore can be included in the PUT request.

   When you send a PUT request, the JSON request body must not contain immutable attributes. The AOS-CX REST API Reference JSON model for the PUT method of the resource shows the mutable attributes. Any mutable attributes you do not include in the PUT request body are set to their defaults, which could be empty.

   The AOS-CX REST API Reference JSON model for the PUT method of the `/system/interfaces/{id}` resource shows the following example:

```
{
...
```

```
  "description": "string",
  "options": {},
  "other_config": {},
  "udld_arubaos_compatibility_mode": "string",
  "udld_compatibility": "string",
  "udld_enable": true,
  "udld_interval": 0,
  "udld_retries": 0,
  "udld_rfc5171_compatibility_mode": "string",
  "user_config": {}
...
}
```

8. Enabling the interface using all the attributes in the return body received from the GET request, modifying the `user_config` attribute to be: `"user_config":{"admin":"up"}`

```
$ curl --noproxy 192.0.2.5 -k -X PUT \
-b /tmp/auth_cookie -d '
{
...
  "options": {},
  "other_config": {},
  "udld_arubaos_compatibility_mode": "forward_then_verify",
  "udld_compatibility": "aruba_os",
  "udld_enable": false,
  "udld_interval": 7000,
  "udld_retries": 4,
  "udld_rfc5171_compatibility_mode": "normal",
  "user_config": {
    "admin": "up"
  }
...
}' \
"https://192.0.2.5/rest/v1/system/interfaces/1%2F1%2F2"
```

   In the preceding example, the following mutable attribute listed in the previous step was not included, so it is set to its default, which could be empty:
   `selftest_disable`

9. Getting the port configuration information to use in the next step.

   Ellipses (...) represent data not shown in the example.

```
$ curl --noproxy 192.0.2.5 -k GET \
-b /tmp/auth_cookie \
"https://192.0.2.5/rest/v1/system/ports/1%2F1%2F2?selector=configuration"
{
...
  "aclv4_out_cfg": {},
  "aclv4_out_cfg_version": {},
  "admin": {},
  "arp_timeout": 1800,
...
  "virtual_ip4_routers": {},
  "virtual_ip6_routers": {},
  "vlan_trunks": []
...
}
```

10. Adding the ACL information to the port using the return body received from the GET request performed in the previous step after verifying the values that are permitted in the JSON model for the PUT method. The modified values are shown in the following example.

    Ellipses (...) represent data not shown in the example.

```
$ curl --noproxy 192.0.2.5 -k -X PUT \
-b /tmp/auth_cookie -d '{
...
"admin":"up",
"interfaces":["/rest/v1/system/interfaces/1%2F1%2F2"],
"aclv4_out_cfg":"/rest/v1/system/acls/ACLv4/ipv4",
"aclv4_out_cfg_version":0,
...
}' -D- \
"https://192.0.2.5/rest/v1/system/ports/1%2F1%2F2"
```

Example: Creating a VLAN with a port using REST APIs

This example shows creating the following VLAN and port configuration on a switch at IP address 192.0.2.5:

```
vlan 2
    no shutdown
interface vlan 2
```

1. Creating the VLAN.

```
$ curl --noproxy 192.0.2.5 -k -X POST \
-b /tmp/auth_cookie -d
'{
...
"name":"vlan2",
"id":2,
"type":"static",
"admin":"up"
...
}' \
"https://192.0.2.5/rest/v1/system/vlans"
```

2. Creating a port and configure the VLAN information.

```
$ curl --noproxy 192.0.2.5 -k -X POST \
-b /tmp/auth_cookie -d
'{
...
"name": "1/1/2",
"admin":"up",
"interfaces":["/rest/v1/system/interfaces/1%2F1%2F2"],
"vlan_mode":"access",
"vlan_tag":"/rest/v1/system/vlans/2",
"routing":false
...
}' \
-D- "https://192.0.2.5/rest/v1/system/ports"
```

If Virtual Switching Extension (VSX) is enabled, you can access the REST API of a peer switch without having to separately log into or manage a session cookie from that peer switch.

To access a peer REST API from your connected switch, insert `/vsx-peer` in the URI path after the server URL and before the REST API and version identifier.

For example:
`https://192.0.2.5/vsx-peer/rest/v1/...`

The following uses of `/vsx-peer` in the URI path are not supported:

- You cannot specify the `login` resource. Requests to `/vsx-peer/rest/v1/login` are not required because logging in to one device automatically gives you access to the peer device.
- You cannot access the Web UI of a VSX peer switch. Setting the browser address to `https://<connected_switch_ip>/vsx-peer` is not supported.
- You cannot specify a VSX peer switch in the URIs in topic subscription messages in the real-time notifications framework. However, you can access the real-time notifications framework on the VSX peer switch by setting the connection address to the following:

  `wss://<connected_switch_ip>/vsx-peer/rest/v1/notification`

Please note the following points when using REST API with VSX.

- VSX must be enabled on both switches, and the interswitch link (ISL) must be up.
- REST API access must be enabled on the switch to which you are connected.
- For write access, the REST API access mode must be set to read-write on the switch to which you are connected.
- You must be logged in to the switch to which you are connected. For example, if you are connected to the primary VSX switch, you must be logged in to the primary switch.
- When configuration synchronization is enabled, supported configuration changes on the primary VSX switch are replicated on the secondary VSX switch. Changing the configuration of a secondary VSX switch might cause the configurations to be out of synchronization.
- Audit messages are logged on the peer switch, with the user information from the switch to which the user is connected.

Examples of curl commands

Getting the VSX status of the secondary VSX switch while connected to the primary VSX switch at IP address 192.0.2.5:

```
$ curl --noproxy "192.0.2.5" -k GET \
-b /tmp/primary_auth_cookie \
"https://192.0.2.5/vsx-peer/rest/v1/system/vsx?attributes=oper_status"
```

Getting the VSX status of the primary VSX switch while connected to the secondary VSX switch at IP address 192.0.2.6:

```
$ curl --noproxy "192.0.2.6" -k GET \
-b /tmp/secondary_auth_cookie \
"https://192.0.2.6/vsx-peer/rest/v1/system/vsx?attributes=oper_status"
```

Getting the names and IP addresses of interfaces on secondary VSX switch while connected to the primary VSX switch at IP address 192.0.2.5:

```
$ curl --noproxy "192.0.2.5" -k GET \
-b /tmp/primary_auth_cookie \
"https://192.0.2.5/vsx-
peer/rest/v1/system/interfaces?depth=1&attributes=name,ipv4_address"
```

For more information about VSX, see the *Virtual Switching Extension (VSX) Guide*.

Example: Interacting with a VSX peer switch

In the following examples, Virtual Switching Extension (VSX) is enabled, the primary VSX switch IP address is 192.0.2.5, and the secondary VSX switch IP address is 192.0.2.6.

Getting the list of all VLANS on the connected switch at IP address 192.0.2.5:

- Example method and URI:
  `GET "https://192.0.2.5/rest/v1/system/vlans"`
- Example curl command:

```
$ curl --noproxy 192.0.2.5 -k GET \
-b /tmp/primary_auth_cookie \
"https://192.0.2.5/rest/v1/system/vlans"
```

Getting the list of all VLANs on the peer VSX switch:

- Example method and URI:
  `GET "https://192.0.2.5/vsx-peer/rest/v1/system/vlans"`
- Example curl command:

```
$ curl --noproxy 192.0.2.5 -k GET \
-b /tmp/primary_auth_cookie \
"https://192.0.2.5/vsx-peer/rest/v1/system/vlans"
```

Getting the VSX status of the secondary VSX switch while connected to the primary VSX switch at IP address 192.0.2.5:

- Example method and URI:
  `GET "https://192.0.2.5/vsx-peer/rest/v1/system/vsx?attributes?oper_status"`
- Example curl command:

```
$ curl --noproxy 192.0.2.5 -k GET \
-b /tmp/primary_auth_cookie \
"https://192.0.2.5/vsx-peer/rest/v1/system/vsx?attributes?oper_status"
```

You can also get the VSX status of the primary VSX switch while connected to the secondary VSX switch.

Example: Upgrading to the latest version of VSX

To perform an upgrade using REST API, the minimum supported version of AOS-CX must be 10.07 or later.

Prerequisites

- The `admin` password must be configured on the VSX peers.

```
user admin password plaintext admin
```

- The REST API access mode must be set to read-write.

```
https-server rest access-mode read-write
```

- The running configuration must be saved before the upgrade.
- The upgrade image version must be validated and verified.
- The REST access must be enabled on the vrf that the REST commands are being sent from.

```
https-server vrf mgmt
```

In the following examples, Virtual Switching Extension (VSX) is enabled, the primary VSX switch IP address is *10.102.8.145*, the software update URL is `tftp://10.100.0.12/halon/GL_10_07_0001BC.swi`, and the VRF is `mgmt`.

To log in to VSX primary, use the following command:

```
curl -g -v -k -c /tmp/rest-cookie "https://10.102.8.145/rest/v1/login" -d
"username=admin&password=admin"
 --noproxy "*"
```

Upgrading VSX using normal mode

The following curl command downloads new software from the TFTP server and verifies the download. After a successful verification, the command installs the software to the alternative software bank of both the VSX primary and secondary switches. The command then reboots them in sequence, the VSX secondary switch followed by VSX primary switch. For example, if a switch has booted with the primary flash memory, then the command will install the software to secondary flash memory.

```
curl -g -v -k -b /tmp/rest-cookie -X PUT "https://10.102.8.145/rest/v1/system/vsx"
-H "Accept: application/json"
-d '{"device_role":"primary", "isl_port":"/rest/v1/system/ports/1%2F1%2F26",
"software_update_url": "tftp://10.100.0.12/halon/GL_10_07_0001.swi",
"software_update_vrf":"/rest/v1/system/vrfs/mgmt"}' --noproxy "*"
```

Upgrading VSX using pre-stage mode

The following curl command upgrades the VSX pairs using the specified boot bank on both the devices. Before running this command, copy the new software copy the new software to a USB flash drive that the switch is capable of booting from, then specify the USB as the target URL for the "software_update_

url", as shown in the example.

```
curl -g -v -k -b /tmp/rest-cookie -X PUT "https://10.102.8.145/rest/v1/system/vsx"
-H "Accept: application/json" -d '{"device_role":"primary",
"isl_port":"/rest/v1/system/ports/1%2F1%2F26",
"software_update_url": "usb://boot_bank=primary",
"software_update_vrf":"/rest/v1/system/vrfs/mgmt"}' --noproxy "*"
```

Aborting the VSX upgrade process

The following curl command aborts the VSX upgrade process. The value for abort request represents the number of times a software update process was requested to be aborted. The value for abort request must be incremented by 1. For example, if you are requesting an abort for the second time, the value will be 2. The abort operation takes effect only when the update operation is in progress.

```
curl -g -v -k -b /tmp/rest-cookie -X PUT "https://10.102.8.145/rest/v1/system/vsx"
-H "Accept: application/json" -d '{"device_role":"primary",
"isl_port":"/rest/v1/system/ports/1%2F1%2F26","software_update_abort_request":1}'
--noproxy "*"
```

Resetting VSX upgrade values

The following example curl command resets the VSX upgrade values:

```
curl -g -v -k -b /tmp/rest-cookie -X PUT "https://10.102.8.145/rest/v1/system/vsx"
 -H "Accept: application/json" -d '{"device_role":"primary",
"isl_port":"/rest/v1/system/ports/1%2F1%2F26"}' --noproxy "*"
```

For more information, see the Virtual Switching Extension (VSX) Guide.

Chapter 8
AOS-CX real-time notifications subsystem

The AOS-CX REST API, combined with built-in databases that provide configuration, state, statistical data, and time-series data for the features and protocols running in the switch, provides a flexible means for switch programmability. Each resource or collection of resources inside the switch is uniquely identified by its URI.

Clients can use the REST API to request information about resources. However, this polling ability does not address the specific use cases in which network management systems need to receive live data or real-time events from the switch. There is a need to have a live notification subsystem that provides the remote network management system with real-time information about any changes that occur in the switch. Timely information about changes is important for troubleshooting and statistical data analyses, as well as for the immediate reaction to real-time events.

The AOS-CX real-time notifications subsystem enables external clients to connect to the switch through a secure WebSocket Protocol connection and to receive real-time notifications about the switch resources and the configuration changes, state changes, and statistical information that interest them.

The WebSocket Protocol was selected based on latency, throughput, resource utilization, network overhead, and security requirements. Multiple clients and connections are supported.

AOS-CX notification messages use JSON encoding. The JSON encoding was designed to align with REST payloads, which enable clients to use combined REST and notification solutions.

The ability to subscribe to these push notifications about a variety of types of information about the switch, combined with the structured nature of the JSON data reported by the switch database, enables a form of network monitoring commonly called telemetry streaming.

Interested clients, known as subscribers, might include the following:

- Web clients such as the AOS-CX Web UI
- Network management systems
- Monitoring scripts

Secure WebSocket Protocol connections for notifications

You subscribe to and receive notifications from the switch through a secure WebSocket Protocol (`wss://`) connection.

A secure WebSocket Protocol connection is a secure, persistent, and full-duplex connection between a client and a server. Either the client or the server can send data in the form of messages at any time.

The handshake part of the WebSocket Protocol uses HTTPS, so there is no need to open a new port on the switch side, and there is no need to provide a new authentication mechanism. When you connect to a switch through a secure WebSocket Protocol connection, you pass the session cookie received from logging in to the REST API. Secure WebSocket Protocol connections to switches running AOS-CX software remain active until the connection is closed, even after the session cookie expires. Multiple clients and connections are supported.

For more information about the WebSocket Protocol see *RFC 6455: The WebSocket Protocol* at:

https://tools.ietf.org/html/rfc6455

Notification topics are switch resource URIs

When you subscribe to notifications, you subscribe to notifications about specific topics. A topic is the URI of a specific switch resource. That URI can contain a query string that specifies particular attributes of that resource.

For example, specifying the following URI as a topic results in notifications being sent when the administrative state or link state of any interface changes, but not when some other attribute of an interface changes:

```
/rest/v1/system/interfaces?depth=1&attributes=admin_state,link_state
```

The AOS-CX REST API Reference lists all the switch resources. You can use the GET method of the resource in the AOS-CX REST API Reference to determine the URI for that switch resource, including the query string to specify an attribute or list of attributes.

Rules for topic URIs

A topic is the URI of a switch resource:

- Not all switch resource URIs are supported as notification topics.

The **Implementation Notes** section of the GET method of the resource in the AOS-CX REST API Reference indicates if the resource is not supported by the notifications subsystem.

- Wildcard characters (`*`) are not supported.
- Specifying a resource on a peer VSX switch, by including `/vsx-peer` in the URIs for topic subscription messages, is not supported. To specify a peer switch, include `/vsx-peer` in the URL of the WSS connection. For example, to get notifications about VLANs on a peer, first open a connection to `wss://192.0.2.5/vsx-peer/rest/v1/notification` and then subscribe to `/rest/v1/system/vlans` as the topic name.
- You can specify a specific resource instance or a collection of resources.

Examples of specific resource instances:

  - `/rest/v1/system/vrfs/default`
  - `/rest/v1/system/vlans/1`

Examples of resource collections:

  - `/rest/v1/system/vrfs/default/bgp_routers`
  - `/rest/v1/system/vlans`

- The `depth` query parameter is supported, with a maximum value of `1`, only with resource collections. For example:
  - Correct: `/rest/v1/system/vlans?depth=1`
  - Incorrect: `/rest/v1/system/vlans/2?depth=1`
- The `attributes` query parameter is supported. You can specify a comma-separated list of attribute names in the query string for either resource collections or resource instances. If attributes are specified, then the subscriber receives notification messages only when the value of one of the specified attributes changes.

For example:

  - The following URI specifies the administrative state and link state of all interfaces on the switch:

    ```
    /rest/v1/system/interfaces?attributes=admin_state,link_state
    ```

  - The following URI specifies the names of the VLANs:

    ```
    /rest/v1/system/vlans?depth=1&attributes=name
    ```

The names of the attributes must match the names as documented in the AOS-CX REST API Reference for the GET method of the resource.

Notification security features

The notification feature uses secure WebSocket connections based on the TLS v1.2 protocol (Transport Layer Security version 1.2), which is the same protocol used for the REST HTTPS connections.

The switch uses self-signed certificates. To avoid certificate verification errors, disable certificate verification when establishing the connection.

AOS-CX real-time notifications subsystem reference summary

The following information is intended as a quick reference for experienced users. Values are not configurable unless noted otherwise.

Connection protocol

WebSocket secure (`wss://`)

Port

443

Message format

JSON

Message types

The following are the supported message types:

- `subscribe`
- `unsubscribe`
- `success`
- `error`
- `notification`

Authorization

Session cookie from successful HTTPS login request

Notification resource URI

`wss://<IP-ADDR>/rest/v1/notification`

<IP-ADDR> is the IP address of the switch.

For example:
`wss://192.0.2.5/rest/v1/notification`

Session idle timeout

None

Session hard timeout

None

Subscription persistence

Subscriptions are active only while the WebSocket secure connection is open.

Configuration maximums

- Maximum number of subscribers per switch: 50
- Maximum number of subscriptions per subscriber: 80
- Maximum number of topics in one subscription message: eight

Enabling the notifications subsystem on a switch

The AOS-CX real-time notifications subsystem relies on the REST API, so the REST API must be enabled on the switch and VRF from which you want to receive notifications.

HTTPS server must be enabled on the specified VRF. The VRF you specify determines from which network the HTTPS server can be accessed. You can enable access on multiple VRFs, including user-defined VRFs.

Procedure

Enable REST API access on the VRF from which you will access the switch.

Establishing a secure WebSocket connection through a web browser

Prerequisites

- Access to the switch REST API must be enabled. The REST API access mode can be either read-only or read/write.
- The web browser you use must support the secure WebSocket Protocol.

Procedure

1. Open a web browser page and log in to the switch Web UI or the REST API.

   The session cookie is managed by the browser and is shared among browser tabs.
2. From a different tab in the same browser, open the page that contains the WebSocket interface.

   For example, many browsers have a plugin for secure WebSocket connections.
3. Connect to the switch at the following URL:
   `wss://<IP-ADDR>/rest/v1/notification`

   `<IP-ADDR>` is the IP address of the switch.

   For example:
   `wss://192.0.2.5/rest/v1/notification`

After the connection is established, you can use the interface to send subscribe or unsubscribe messages and to view the responses and notification messages.

Establishing a secure WebSocket connection using a script

Prerequisites

Access to the switch REST API must be enabled. The REST API access mode can be either read-only or read/write.

Procedure

1. If you are using a script, you must include the actions to log in, get the session cookie, store the session cookie, and pass the session cookie with the secure WebSocket connection request.

   When you create the secure WebSocket connection, use the following URL:
   `wss://<IP-ADDR>/rest/v1/notification`
2. *<IP-ADDR>* is the IP address of the switch.

   For example:
   `wss://192.0.2.5/rest/v1/notification`
3. The exact methods to use to create connections and handle notification messages depend on the scripting language and library module you choose.

Subscribing to topics

Prerequisites

- You must have a secure WebSocket connection to the switch.
- Access to the switch REST API must be enabled. The REST API access mode can be either read-only or read/write.

Procedure

Using the WebSocket secure connection, send a subscribe message that contains the topics to which you want to subscribe and a poll interval hint, if any.

Some resource attributes—typically in the statistics category—are not populated until a client requests the information. The value of `hint` specifies how often—in seconds—the notification subsystem is to request information about the topics in the list.

For example:

```
{
  "type": "subscribe",
  "topics": [
    {
      "name": "/rest/v1/system/vrfs"
    },
    {
      "name": "/rest/v1/system/vlans/1?attributes=admin,oper_state_reason"
    }
  ],
  "hint": 5
}
```

If the subscriber already has a subscription to the specified topic, the following error is returned:

```
{
  "type":"error",
  "message":"The topic or combination of topics have been already subscribed."
}
```

If the URI in the topic name specifies a resource that is not in the configuration and state database, the following error is returned:

```
{"type":"error","message":"Object not found."}
```

Example of a message returned by a successful subscription attempt:

```
{
  "type": "success",
  "subscriber_name": "4bcf8uka90ki",
  "subscription_name": "ns83n58dky",
  "data": [
    {
      "topicname": "/rest/v1/system/vlans/1?attributes=admin,oper_state_reason",
      "resources": [
```

```
        {
          "operation": "",
          "uri": "/rest/v1/system/vlans/1",
          "values": {
            "admin": "up",
            "oper_state_reason": "no_member_port"
          }
        }
      ]
    },
    {
      "topicname": "/rest/v1/system/vrfs",
      "resources": [
        {
          "operation": "",
          "uri": "/rest/v1/system/vrfs/default",
          "values": {}
        },
        {
          "operation": "",
          "uri": "/rest/v1/system/vrfs/mgmt",
          "values": {}
        }
      ]
    }
  ]
}
```

Unsubscribing from topics

Prerequisites

- You must have a secure WebSocket connection to the switch.
- The switch must have REST API access enabled. The REST API access mode can be either read-only or read/write.

Procedure

Use the secure WebSocket connection to send an unsubscribe message that specifies the topic or topics from which you no longer want notifications.

Use a comma to separate topics in a list of topics.

You must be connected as the same subscriber that subscribed to the topic. For example, you must be using the same web browser session or be passing the same session cookie with the request.

For example, to unsubscribe notifications about the default VRF, send the following message through the WebSocket secure connection:

```
{
  "type": "unsubscribe",
  "topics": [
    {
      "name": "/rest/v1/system/vrfs/default"
    }
  ]
}
```

If the subscriber does not have a subscription to that topic, the following message is returned:

```
{
  "type": "error",
  "message": "subscription /rest/v10.04/system/vrfs doesn't exist"
```

```
  "data": null
}
```

The error can indicate that you have already unsubscribed, the connection was lost, or you attempted to unsubscribe from a different subscriber.

If the request is successful, the following message is returned:

```
{
  "type": "success",
  "message": "Successfully unsubscribe."
}
```

Parts of a subscribe message

A subscribe message is the message sent when a subscriber requests a subscription to a topic on a switch. The subscribe message is in JSON format.

Subscribe message example

```
{
  "type": "subscribe",
  "topics": [
    {
      "name": "/rest/v1/system/vrfs"
    },
    {
      "name": "/rest/v1/system/vlans/1?attributes=admin,oper_state_reason"
    }
  ],
  "hint": 5
}
```

Components of a subscribe message

`type`

Required. For a subscribe message, you must specify the following value: `subscribe`

`topics`

Required. The value is a comma-separated list of one or more topics in JSON format. A topic includes one component:

`name`

Required. The name of the topic, identified by the URI of the switch resource, including the optional query string.

`hint`

Optional. Some resource attributes—typically in the statistics category—are not populated until a client requests the information. The value of `hint` specifies how often—in seconds—the notification subsystem is to request information about the topics in the list. The same `hint` value applies to all the topics in the list.

If the same resource is a topic in multiple subscriptions that have different values for `hint`, the notification subsystem uses the smallest value.

Default: 10

Parts of a subscription success message

When a subscription request is successful, a subscription success message is returned. The subscription success message is in JSON format.

Example success message

```
{
  "type": "success",
  "subscriber_name": "4bcf8uka90ki",
  "subscription_name": "ns83n58dky",
  "data": [
    {
      "topicname": "/rest/v1/system/vlans/1?attributes=admin,oper_state_reason",
      "resources": [
        {
          "operation": "",
          "uri": "/rest/v1/system/vlans/1",
          "values": {
            "admin": "up",
            "oper_state_reason": "no_member_port"
          }
        }
      ]
    },
    {
      "topicname": "/rest/v1/system/vrfs",
      "resources": [
        {
          "operation": "",
          "uri": "/rest/v1/system/vrfs/default",
          "values": {}
        },
        {
          "operation": "",
          "uri": "/rest/v1/system/vrfs/mgmt",
          "values": {}
        }
      ]
    }
  ]
}
```

Components of subscription success message

`type`

Identifies the type of message. Success messages have the type: `success`

`subscriber_name`

Contains a unique identifier that represents the name of the subscriber.

`subscription_name`

Contains a unique identifier that represents the name of the specific subscription.

`data`

Contains a comma-separated list of one or more topics in JSON format.

Components of a topic

In a subscription success message, each topic in the data contains the following components:

`topicname`

Contains the name of the topic, identified by the URI of the switch resource, including the optional query string.

`resources`

Contains a comma-separated list of one or more resources in JSON format. When the URI of a topic is a resource collection, a topic includes multiple resources. In the example message, the `vrfs` resource includes two VRF instances: `default` and `mgmt`.

Each resource includes the following components:

`operation`

The value of `operation` is empty for success messages. This component is used for notification messages only.

`uri`

Contains the URI of the resource instance within the resource collection. If the `topicname` is a resource instance instead of a collection, `uri` matches the path portion of the URI in `topicname`.

`values`

Contains the names and current values of the attributes that were specified in the query string of `topicname`.

Parts of a notification message

A notification message is the message sent to the subscriber when there is a change to a switch resource that is the topic of a subscription. The notification message is in JSON format.

The content of a notification message depends on the type of change that occurred.

Notification message examples

For the following examples, assume that the following subscribe message was used:

```
{
  "topics": [
    {
      "name": "/rest/v1/system/vlans?depth=1&attributes=name"
    }
  ],
  "type": "subscribe"
}
```

The subscriber receives a notification when the name of any VLAN changes:

In the following example, VLAN7 has been added to the switch configuration:

```
{
  "data": [
    {
      "resources": [
        {
          "operation": "inserted",
          "uri": "/rest/v1/system/vlans/VLAN7",
          "values": {
            "name": "VLAN7"
          }
        }
      ],
      "topicname": "/rest/v1/system/vlans?depth=1&attributes=name"
    }
  ],
  "type": "notification"
}
```

In the following example, VLAN7 has been deleted from the configuration:

```
{
  "data": [
    {
      "resources": [
        {
          "operation": "deleted",
          "uri": "/rest/v1/system/vlans/VLAN7",
          "values": {}
        }
      ],
      "topicname": "/rest/v1/system/vlans?depth=1&attributes=name"
    }
  ],
  "type": "notification"
}
```

In the following example, the subscriber has subscribed to the following topic:
`/rest/v1/system/interfaces/1%2F1%2F2?attributes=name,admin_state`

If either the name or the administrative state of interface `1/1/2` changes, a notification message is sent. If attributes other than name or administrative state changes, no notification message is sent.

In the following example, the administrative state of the interface changed to `up`.

```
{
  "data": [
    {
      "resources": [
        {
          "operation": "modified",
          "uri": "/rest/v1/system/interfaces/1%2F1%2F2",
          "values": {
            "admin_state": "up"
          }
        }
      ],
      "topicname": "/rest/v1/system/interfaces/1%2F1%2F2?attributes=name,admin_state"
    }
  ],
  "type": "notification"
}
```

Components of a notification message

`type`
: Identifies the type of message. Notification messages have the type: `notification`

`data`
: Contains a comma-separated list of one or more topics in JSON format.

Components of a topic

In a notification message, each topic in the data contains the following components:

`topicname`
: Contains the name of the topic, identified by the URI of the switch resource, including the optional query string.

`resources`
: Contains a comma-separated list of one or more resources in JSON format. When the URI of a topic is a resource collection, a topic includes multiple resources.

  Each resource includes the following components:

`operation`
: For notification messages, operation is one of the following values:

`inserted`
: The resource or resource attribute was added to the configuration of the switch.

`deleted`
: The resource or resource attribute was deleted from the switch.

`modified`
: The resource or resource attribute changed.

`uri`
: Contains the URI of the resource instance within the resource collection. If the `topicname` is a resource instance instead of a collection, `uri` matches the path portion of the URI in `topicname`.

`values`
: The content of `values` depends on the operation:

- When the `operation` value is `deleted`, `values` is empty.
- When the `operation` value is `inserted`, `values` contains the current names and values of the attributes specified in the query portion of the `topicname`. If no query string was included in `topicname`, all attributes and values for that resource are included.
- When the `operation` value is `modified`, `values` contains the name and current value of the attribute in the query string that changed value:
  - If no query string was included in `topicname`, all attributes and values for that resource are included.
  - If multiple attributes are included in the query string of a topic and only some of those attribute values changed, only the changed attributes are included.
  - If an attribute that was not included in the query string changes, no notification message is sent because that attribute is not part of the subscription.

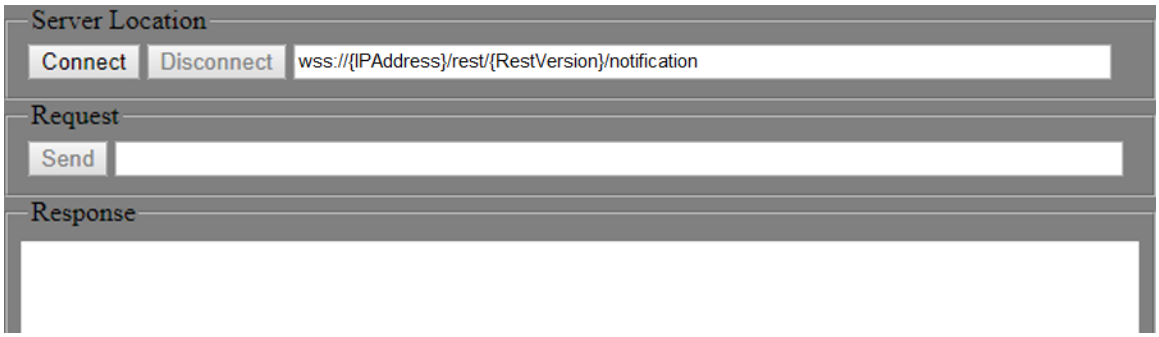
Example: Browser-based WebSocket connection

About the example

The following example, `websocket-client.html`, uses HTML and Javascript to create a webpage that you can use to establish a WSS connection and send and receive notification messages.

- Access to the switch REST API must be enabled on the VRF through which this browser will connect to the switch.
- Before you can use the HTML page, you must log in to the switch Web UI or REST API from a separate tab in the same web browser session. The browser shares the session cookie between tabs.
- When the browser page is open, in **Server Location**, substitute the switch IP address for `{IPAddress}` in `wss://{IPAddress}/rest/v1/notification`, then click **Connect**.
- Enter the subscription message in **Request** and click **Send**.
- Responses and notifications are shown in **Response**.

Example screen



Example HTML source

```
<!DOCTYPE html>
<html lang="en">
<head>
    <title>Web Socket Client Example</title>
    <script type="text/javascript">
        window.onload = function () {
            var conn;
```

```
            var log = document.getElementById("log");
            var msg = document.getElementById("msg");

            function appendLog(item) {
                var doScroll = log.scrollTop === log.scrollHeight -
log.clientHeight;
                log.appendChild(item);
                if (doScroll) {
                    log.scrollTop = log.scrollHeight - log.clientHeight;
                }
            }

            document.getElementById("connect").onclick = function () {
                var server = document.getElementById("wsURL");
                conn = new WebSocket(server.value);
                if (window["WebSocket"]) {
                    if (conn) {
                        conn.onopen = function (evt) {
                            document.getElementById("disconnect").disabled = false
                            document.getElementById("sendMsg").disabled = false
                            document.getElementById("connect").disabled = true
                            document.getElementById("status").innerHTML =
"Connection opened"
                        }
                        conn.onclose = function (evt) {
                            document.getElementById("status").innerHTML =
"Connection closed"
                            document.getElementById("connect").disabled = false
                        };
                        conn.onmessage = function (evt) {
                            var messages = evt.data.split('\n');
                            for (var i = 0; i < messages.length; i++) {
                                var item = document.createElement("pre");
                                item.innerText = messages[i];
                                appendLog(item);
                            }
                        }
                    }
                } else {
                    var item = document.createElement("pre");
                    item.innerHTML = "<b>Your browser does not support
WebSockets.</b>";
                    appendLog(item);
                }
            };

            document.getElementById("disconnect").onclick = function () {
                conn.close()
                document.getElementById("sendMsg").disabled = true
                document.getElementById("connect").disabled = false
                document.getElementById("disconnect").disabled = true
                document.getElementById("status").innerHTML = "Connection closed"
            };

            document.getElementById("form").onsubmit = function () {
                if (!conn) {
                    return false;
                }
                if (!msg.value) {
                    return false;
                }
                conn.send(msg.value);
```

```
                var item = document.createElement("pre");
                item.classList.add("subscribeMsg");
                item.innerHTML = msg.value;
                appendLog(item);
                return false;
            };
        };

    </script>
    <style type="text/css">
        html {
            overflow: hidden;
        }

        body {
            overflow: hidden;
            padding: 0;
            margin: 0;
            width: 100%;
            height: 100%;
            background: gray;
        }

        #log {
            background: white;
            margin: 0;
            padding: 0.5em 0.5em 0.5em 0.5em;
            top: 1.5em;
            left: 0.5em;
            right: 0.5em;
            bottom: 3em;
            overflow: auto;
            position: absolute;
            height: 530px;
        }

        #form {
            padding: 0 0.5em 0 0.5em;
            margin: 0;
            position: absolute;
            bottom: 3em;
            top: 5em;
            left: 8px;
            width: 100%;
            overflow: hidden;
        }

        #serverLocation {
            padding-top: 0.3em;
        }

        #requestSection {
            height: 38px;
        }

        #responseMsgSection {
            height: 570px;
            position: relative;
        }
    </style>
</head>
<body>
```

```
<fieldset id="serverLocation">
    <legend>Server Location</legend>
    <div>
        <input type="button" id="connect" value="Connect"/>
        <input type="button" id="disconnect" value="Disconnect" disabled/>
        <input type="text" id="wsURL" value="wss://
{IPAddress}/rest/v1/notification" size="64">
        <span id="status"></span>
    </div>
</fieldset>
<fieldset id="requestSection">
    <legend>Request</legend>
    <form id="form">
        <input type="submit" type="submit" value="Send" ; disabled/>
        <input type="text" id="msg" size="80"/>
    </form>
</fieldset>
<fieldset id="responseMsgSection">
    <legend>Response</legend>
    <div id="log"></div>
</fieldset>
</body>
</html>
```

Example: Getting information about current subscribers and subscriptions

To get information about the subscribers receiving notifications from a switch, you must use the REST API.

Instructions and examples in this document use an IP address that is reserved for documentation, 192.0.2.5, as an example of the IP address for the switch. To access your switch, you must use the IP address or hostname of that switch.

Prerequisites

You must be logged in to the switch REST API.

Procedure

- To get the list of current subscribers, send a GET request to the `notification_subscribers` resource.

For example:

```
GET "https://192.0.2.5/rest/v1/system/notification_subscribers"
```

The response body is a list of URIs. The identifier at the end of the URI string is the subscriber name.

For example:

```
[
  "rest/v1/system/notification_subscribers/z6901beisjgf",
  "rest/v1/system/notification_subscribers/18l9g87erb42"
]
```

- To get a list of all subscriptions of all subscribers, use the `depth=1` parameter when sending the GET request to the `notification_subscribers` resource.

For example:

```
GET "https://192.0.2.5/rest/v1/system/notification_subscribers?depth=1"
```

The response body contains the list of subscriptions for each subscriber.

In the following example,

- Subscriber `z6901beisjgf` has two subscriptions:
  - `5mzo50lgoo`
  - `pouswxt9m9`
- Subscriber `18l9g87erb42` has one subscription:
  - `dz95lljqwk`

```
[
  {
    "name": "z6901beisjgf",
    "notification_subscriptions": {
      "5mzo50lgoo": "rest/v1/system/notification_
subscribers/z6901beisjgf/notification_subscriptions/5mzo50lgoo",
      "pouswxt9m9": "rest/v1/system/notification_
subscribers/z6901beisjgf/notification_subscriptions/pouswxt9m9"
    },
    "type": "ws"
  },
  {
    "name": "18l9g87erb42",
    "notification_subscriptions": {
      "dz95lljqwk": "rest/v1/system/notification_
subscribers/18l9g87erb42/notification_subscriptions/dz95lljqwk"
    },
    "type": "ws"
  }
]
```

- To get a list of subscriptions that belong to a specific subscriber, send a GET request to the `notification_subscriptions` resource of the subscriber.

The following example gets the list of all the subscriptions of subscriber `z6901beisjgf`:

```
GET "https://192.0.2.5/rest/v1/system/notification_
subscribers/z6901beisjgf/notification_subscriptions"
```

The response body is a list of URIs. The identifier at the end of the URI string is the subscription name.

Example response body:

```
[
  "rest/v1/system/notification_subscribers/z6901beisjgf/notification_
subscriptions/5mzo50lgoo",
  "rest/v1/system/notification_subscribers/z6901beisjgf/notification_
subscriptions/pouswxt9m9"
]
```

- To get detailed information about a specific subscription, send a GET request to the `notification_subscriptions/{subscription-ID}` resource for that subscription.

The `notification_subscriptions` resource is a child resource of the specific subscriber:

`/system/notification_subscribers/{subscriber-id}/notification_subscriptions/{subscription-id}`

For example, to get information about subscription `5mzo50lgoo`, you must specify the subscriber name and the subscription name in the URI:

```
GET "https://192.0.2.5/rest/v1/system/notification_
subscribers/z6901beisjgf/notification_subscriptions/5mzo50lgoo"
```

Example response body:

```
{
  "5mzo50lgoo": {
    "resource": [
      "/rest/v1/system/ports?attributes=admin,vlan_mode,vlan_tag,vlan_
trunks,interfaces&depth=1"
    ]
  }
}
```

- To get detailed information about all subscriptions of specific subscriber, use the `depth=1` parameter when sending the GET request to the `notification_subscriptions` resource of that subscriber.

For example:

```
GET "https://192.0.2.5/rest/v1/system/notification_
subscribers/z6901beisjgf/notification_subscriptions?depth=1"
```

Example response body:

```
{
  "5mzo50lgoo": {
    "resource": [
      "/rest/v1/system/ports?attributes=admin,vlan_mode,vlan_tag,vlan_
trunks,interfaces&depth=1"
    ]
  },
  "pouswxt9m9": {
    "resource": [
      "/rest/v1/system/interfaces?attributes=type,hw_intf_info,link_state,link_
speed,error,other_config"
    ]
  }
}
```

General troubleshooting tips

Connectivity

Connectivity is often the first issue you encounter. Ensure that you have enabled https-server on the VRF you are trying to use.

- To connect to the REST API through the management (OOBM) port, REST API access must be enabled on the management VRF.
- To connect to the REST API through a data port, REST API access must be enabled on the default VRF or a user-created VRF that includes that data port.

Resources, attributes, and behaviors

- Resources, attributes, and behaviors might differ between different versions of the switch software. For example:

The `bridge` resource was eliminated from the URI path of REST v1 URIs beginning with AOS-CX version 10.03:

- Example of getting the list of VLANs for a switch running AOS-CX version 10.02:
  `GET "https://192.0.2.5/rest/v1/system/bridge/vlans"`
- Example of getting the list of VLANs for a switch running AOS-CX version 10.03:
  `GET "https://192.0.2.5/rest/v1/system/vlans"`

If you are getting errors when making requests to switches with different software versions, use the AOS-CX REST API Reference on each switch to compare the URI paths and attributes for the resource. You might need to alter your code to handle the different software versions.

- Resources, attributes, and behaviors might differ between different versions of the REST API, and a switch supports access through multiple versions of the REST API.

GET, PUT, POST, and DELETE methods

- Most resources do not allow POST, PUT, or DELETE methods and do not display those methods in the AOS-CX REST API Reference unless the REST access mode is set to `read-write`.
- The JSON model of a resource can vary by method used. The JSON data you receive from the GET method is not the same as the JSON data you can or must provide with the POST or PUT methods:
  - The GET method model contains all the attributes.
  - The POST method model contains only the configuration attributes.
  - The PUT method model contains only the **mutable** (changeable) configuration attributes. If you do not provide all the mutable attributes in the request body of the PUT request, those attributes you do not provide are set to their defaults, which could be empty. If you attempt to provide an immutable attribute in a PUT request, an error is returned.

- Use the GET method with the `selector=configuration` parameter to get only the configuration attributes of a resource. You can use the AOS-CX REST API Reference to view information about the supported methods and resource models.
- You can obtain additional platform-specific information through GET requests for product information attributes or subsystem collections. Aruba 8400 switch examples:
  - Example request:

    ```
    GET "https://192.0.2.5/rest/v1/system/subsystems"
    ```

    Example response body:

    ```
    [
      "/rest/v1/system/subsystems/chassis,base",
      "/rest/v1/system/subsystems/line_card,1%2F3",
      "/rest/v1/system/subsystems/management_module,1%2F5"
    ]
    ```

  - Example request:

    ```
    GET "https://192.0.2.5/rest/v1/system/subsystems/chassis,base?attributes=product_
    info"
    ```

    Example response body:

    ```
    {
      "product_info": {
        "base_mac_address": "00:00:5E:00:53:00",
        "device_version": "",
        "instance": "1",
        "number_of_macs": "512",
        "part_number": "JL375A",
        "product_description": "8400 8-slot Chassis/3xFan Trays/18xFans/Cable
    Manager/X462 Bundle",
        "product_name": "8400 Base Chassis/3xFT/18xFans/Cbl Mgr/X462 Bundle",
        "serial_number": "SG00A2A00A",
        "vendor": "Aruba"
      }
    }
    ```

- Aruba 8320 switch examples:
  Example request:

```
GET "https://192.0.2.5/rest/v1/system/subsystems
```

Example response body:

```
[
  "/rest/v1/system/subsystems/chassis/base",
  "/rest/v1/system/subsystems/line_card,1%2F1",
  "/rest/v1/system/subsystems/management_module,1%2F1"
]
```

Hardware and other features

- Different switches have different hardware and features. For example, the management module resource ID is 1/1 for some switches, and 1/4 or 1/5 for other switches. To get information about the switch model, use the GET method request with the URI for the `platform_name` system attribute.

For example:

```
GET "https://192.0.2.5/rest/v1/system?attributes=platform_name"
```

The following is an example of a response body for an Aruba 8320 switch:

```
{
  "platform_name": "8320"
}
```

The following is an example of a response body for an Aruba 8400 switch:

```
{
  "platform_name": "8400X"
}
```

- The words "port" and "interface" have meanings that are different from other network operating systems. In the AOS-CX operating system:
  - A port is the logical representation of a port.
  - An interface is the hardware representation of a port.
- You can enable debugging logs by using the `debug` command. The module name is `rest`. You can specify all severity log levels or a minimum severity log level.

Example specifying all severity log levels:

```
switch# debug rest all
```

Example specifying a minimum severity log level of `error`:

```
switch# debug rest all severity error
```

REST API response codes

The following table describes the different categories of the response codes.

Category	Description
2*xx*	Indicates that the request was accepted successfully.
4*xx*	Returns the client-side error response with the error message.
5*xx*	Returns the server-side error response with the error message.

The following are some response codes that you will see in the REST API.

Response code	Status	Description
200	OK	Returned from GET and PUT operations, and non-configuration API calls such as Login or Logout when the request is successfully completed.
201	Created	Returned from POST operations when a new resource was successfully created.

Response code	Status	Description
204	No Content	Returned from a PUT, POST, or DELETE operation when the request was successfully processed and there is no content to return.
400	Bad request	A problem with the request body, such as invalid syntax, incorrectly formatted JSON, or data violating a database constraint.
401	Unauthorized	No active session for this client (the login API has not been called) or too many sessions already created from this client.
403	Forbidden	The client session is valid, but does not have permissions to access the requested resource.
404	Not found	The resource does not exist, or the URI is incorrect for the desired resource. Can also occur when accessing the POST, PUT, or DELETE API while the REST access-mode is set to read-only.
500	Internal server error	An unexpected error has occurred in processing the request. View the logs on the device for details.
503	Service unavailable	The device is receiving more requests than it can process and is defensively rejecting requests to protect resources.

Error "'admin' password is not set"

Symptom

An attempt to enable the HTTPS server using the `https-server vrf` command fails and the following error is returned:

`Failed to enable https-server on VRF <VRF>. 'admin' password is not set`

Cause

The switch is shipped from the factory with a default user named `admin` without a password. The `admin` user must set a valid password before HTTPS servers can be enabled.

Action

From the global configuration context, set a valid password for the `admin` user.

For example:

```
switch(config)# user admin password
Changing password for user admin
Enter password:************
Confirm password:************
```

Error "certificate verify failed" returned from curl command

Symptom

A curl command to the switch URL fails with an error similar to the following:

`SSL3_GET_SERVER_CERTIFICATE:certificate verify failed`

Cause

The curl program could not verify the switch server certificate against the CA certificate bundle that comes with the curl installation, and you did not include the `-k` option in the curl command.

Action

Retry the command with the `-k` option included.

The switch HTTPS server uses self-signed certificates, which cannot be verified against a certificate authority. The `-k` option disables curl certificate validation.

For example:

```
$ curl -k --noproxy "192.0.2.5" GET /tmp/auth_cookie \
"https://192.0.2.5/rest/v1/system/vlans"
```

HTTP 400 error "Invalid Operation"

Symptom

A REST request returns response code 400 and the response body contains the following text string:
`Invalid operation`

Cause

The method used for this REST request is not supported for the specified resource. For example, the `Invalid operation` response is returned if you attempt a DELETE request on the `system` resource.

Action

Use a method supported by the resource.

The AOS-CX REST API Reference displays the methods supported by each resource.

HTTP 400 error "Value is not configurable"

Symptom

A PUT or POST request returns response code 400 and the response body contains the following text string:

`Value <value> is not configurable`

Cause

The JSON data in the POST or PUT request body contains non-configuration or immutable attributes.

Action

Retry the request with the correct JSON resource model for that PUT or POST method.

To determine the configuration attributes of a resource, you can send a GET request with the `selector=configuration` query parameter to the resource. Using the REST v10.04 API, you can also use the GET method with the `selector=writable` parameter to get only the mutable configuration attributes of the resource.

You can also use the AOS-CX REST API Reference to verify the JSON model of the PUT or POST method of the resource.

The category an attribute belongs to can depend on whether that instance of the resource is owned by the system or owned by a user. Configuration attributes can become status attributes in resource instances that are owned by the system. Status attributes can not be modified by users.

In addition, some configuration attributes cannot be changed after a resource is created. These immutable attributes cannot be included in a PUT request.

HTTP 400 error "Reference failure"

Symptom

A REST request returns response code 400 and the response body contains the following text string:

```
Reference failure
```

Cause

You attempted to delete a resource that is referenced by other resources. Typically, this error occurs for resources that have no clear parent in the resource hierarchy, such as ports. For example, the `Reference failure` response is returned if you attempt a DELETE request on a port.

Action

Remove all references to the resource.

After all references to a resource are removed, the resource is deleted automatically.

HTTP 401 error "Authorization Required"

Symptom

A REST request returns response code 401 and the response body contains the following text string:

```
Authorization Required
```

This response means that no valid session was found for the session token passed to the API.

Solution 1

Cause

The user attempting the request is not logged into the REST API for one of the following reasons:

- The user has not yet logged in.
- The user logged in but the session has expired.

Action

Log in to the REST API.

Solution 2

Cause

The user attempting the request is not logged in to the REST API because the user did not pass the correct session cookie to the API. Typically, incorrect session cookies are not a cause when accessing the REST API through a browser because the browser automatically handles the session cookie.

Action

1. Ensure that you save the session cookie returned from the login request.
2. Ensure that you pass the same cookie back to the switch with every REST API request, including the request to log out.

HTTP 401 error "Login failed: session limit reached"

Symptom

A REST request or Web UI login attempt returns response code 401 and the response body contains the following text string:

```
Login failed: session limit reached
```

Cause

A user attempted to log into the REST API or the Web UI, but that user already has the maximum number of concurrent sessions running.

Action

1. Log out from one of the existing sessions.

   Browsers share a single session cookie across multiple tabs or even windows. However, scripts that POST to the login resource and later do not POST to the logout resource can easily create the maximum number of concurrent sessions.
2. If the session cookie is lost and it is not possible to log out of the session, then wait for the session idle time limit to expire.

   When the session idle timeout expires, the session is terminated automatically.
3. If it is required to stop all HTTPS sessions on the switch instead of waiting for the session idle time limit to expire, you can stop all HTTPS sessions using the `https-server session close all` command.

   This command stops and starts the `hpe-restd` service, so using this command affects all existing REST sessions and Web UI sessions.

HTTP 403 error "Forbidden" on a write request

Symptom

A POST, PUT, or DELETE REST request returns response code 403 and the response body contains the following text string:

```
Forbidden
```

Cause

The user attempting the request is not a member of the `administrators` group.

Action

Log in to the REST API with a user name that has administrator rights as part of the `administrators` group.

The user must be a member of the predefined `administrators` group. POST requests to the `login` resource fail for members of a user-defined local user group.

HTTP 403 error "Forbidden" on a GET request

Symptom

A GET REST request returns response code 403 and the response body contains the following text string:

`Forbidden`

Cause

The user attempting the request is a member of the Auditors group, and the GET request specified a switch resource that users with auditor rights are not permitted to access.

Action

Log in to the REST API with a user name that has operator or administrator rights.

HTTP 404 error "Page not found" when accessing the switch URL

Symptom

The switch is operational and you are using the correct URL for the switch, but attempts to access the REST API or Web UI result in an HTTP 404 "Page not found" error.

Cause

REST API access is not enabled on the VRF that corresponds to the access port you are using. For example, you are attempting to access the REST API or Web UI from the management (OOBM) port, and access is not enabled on the `mgmt` VRF.

Action

Use the `https-server vrf` command to enable REST API access on the specified VRF.

For example:

```
switch(config)# https-server vrf mgmt
```

HTTP 404 error "Object not found" on object with "bridge/" in URI Path

Symptom

A request was made to a switch running AOS-CX version 10.03 or later using a REST v1 URI that contains `bridge/` in the URI path. The request returns response code 404 and the response body contains the following text string:

`Object not found`

Cause

The resource does not exist in the system. The URI in the request is incorrect.

The `bridge` collection was eliminated from the REST v1 API in AOS-CX version 10.03.

Action

Remove the following from the URI and retry the request:

```
bridge/
```

Example of getting the list of VLANs for a switch running AOS-CX version 10.02:

```
GET "https://192.0.2.5/rest/v1/system/bridge/vlans"
```

Example of getting the list of VLANs for a switch running AOS-CX version 10.03:

```
GET "https://192.0.2.5/rest/v1/system/vlans"
```

HTTP 404 error "Object not found" returned from a switch that supports multiple REST API versions (10.04 and later)

Symptom

A switch that supports multiple REST API versions returns response code 404 and the response body contains the following text string:

```
Object not found
```

Cause

The resource does not exist in the system. The URI in the request is incorrect for the version of the REST API specified in the request.

Action

Verify the URI of the resource and retry the request.

The schema for resources accessed through the REST v1 API can differ from the schema for the resources accessed through the REST v10.04 API.

For example, a REST request with the following URI will fail because the interfaces collection does not exist in the REST v1 API:

```
/rest/v1/system/interfaces/lag50
```

The correct URI for the `lag50` resource in the REST v1 API is the following:

```
/rest/v1/system/ports/lag50
```

HTTP 404 error "Object not found" when using a write method

Symptom

A PUT or DELETE request returns response code 404 and the response body contains the following text string:

```
Object not found
```

Cause

The resource does not exist in the system. The URI in the request is incorrect or the resource has not been added to the configuration.

Action

Verify the URI of the resource you are attempting to change or delete and retry the request.

HTTP 404 error "Page not found" when using a write method

Symptom

Using the GET method is successful, but attempting a POST, PUT, or DELETE method results in an HTTP 404 "Page not found" error.

Cause

The REST API access mode is set to `read-only`.

Action

Set the REST API access mode to `read-write`.

```
switch(config)# https-server rest access-mode read-write
```

Enabling the read-write mode on the REST API allows POST, PUT, and DELETE operations to be called on all configurable elements in the switch database.

Logout fails

Symptom

An attempt to log out of the REST API from a script or curl command fails.

Cause

The session cookie was not supplied or does not contain the correct session token.

Action

1. Repeat the command and send the correct session cookie or modify the script to send the correct session cookie.
2. If the session cookie has been lost and it is not possible to log out of the session, wait for the session idle time limit to expire.

   When the session idle timeout expires, the session is terminated automatically.

Accessing Aruba Support

Aruba Support Services	https://www.arubanetworks.com/support-services/
AOS-CX Switch Software Documentation Portal	https://www.arubanetworks.com/techdocs/AOS-CX/help_portal/Content/home.htm
Aruba Support Portal	https://asp.arubanetworks.com/
North America telephone	1-800-943-4526 (US & Canada Toll-Free Number) +1-408-754-1200 (Primary - Toll Number) +1-650-385-6582 (Backup - Toll Number - Use only when all other numbers are not working)
International telephone	https://www.arubanetworks.com/support-services/contact-support/

Be sure to collect the following information before contacting Support:

- Technical support registration number (if applicable)
- Product name, model or version, and serial number
- Operating system name and version
- Firmware version
- Error messages
- Product-specific reports and logs
- Add-on products or components
- Third-party products or components

Other useful sites

Other websites that can be used to find information:

Airheads social forums and Knowledge Base	https://community.arubanetworks.com/
AOS-CX Switch Software Documentation Portal	https://www.arubanetworks.com/techdocs/AOS-CX/help_portal/Content/home.htm
Aruba Hardware Documentation and Translations	https://www.arubanetworks.com/techdocs/hardware/DocumentationPortal/Content/home.htm

Portal	
Aruba software	https://asp.arubanetworks.com/downloads
Software licensing	https://lms.arubanetworks.com/
End-of-Life information	https://www.arubanetworks.com/support-services/end-of-life/
Aruba Developer Hub	https://developer.arubanetworks.com/

Accessing Updates

You can access updates from the Aruba Support Portal or the HPE My Networking Website.

Aruba Support Portal

https://asp.arubanetworks.com/downloads

If you are unable to find your product in the Aruba Support Portal, you may need to search My Networking, where older networking products can be found:

My Networking

https://www.hpe.com/networking/support

To view and update your entitlements, and to link your contracts and warranties with your profile, go to the Hewlett Packard Enterprise Support Center **More Information on Access to Support Materials** page:

https://support.hpe.com/portal/site/hpsc/aae/home/

Access to some updates might require product entitlement when accessed through the Hewlett Packard Enterprise Support Center. You must have an HP Passport set up with relevant entitlements.

Some software products provide a mechanism for accessing software updates through the product interface. Review your product documentation to identify the recommended software update method.

To subscribe to eNewsletters and alerts:

https://asp.arubanetworks.com/notifications/subscriptions (requires an active Aruba Support Portal (ASP) account to manage subscriptions). Security notices are viewable without an ASP account.

Warranty Information

To view warranty information for your product, go to https://www.arubanetworks.com/support-services/product-warranties/.

Regulatory Information

To view the regulatory information for your product, view the *Safety and Compliance Information for Server, Storage, Power, Networking, and Rack Products*, available at https://www.hpe.com/support/Safety-Compliance-EnterpriseProducts

Additional regulatory information

Aruba is committed to providing our customers with information about the chemical substances in our products as needed to comply with legal requirements, environmental data (company programs, product recycling, energy efficiency), and safety information and compliance data, (RoHS and WEEE). For more information, see https://www.arubanetworks.com/company/about-us/environmental-citizenship/.

Documentation Feedback

Aruba is committed to providing documentation that meets your needs. To help us improve the documentation, send any errors, suggestions, or comments to Documentation Feedback (docsfeedback-switching@hpe.com). When submitting your feedback, include the document title, part number, edition, and publication date located on the front cover of the document. For online help content, include the product name, product version, help edition, and publication date located on the legal notices page.