Lenovo Network

Python Programming Guide

For Lenovo Cloud Network Operating System 10.10

Lenovo™

Note: Before using this information and the product it supports, read the general information in the *Safety information and Environmental Notices* and *User Guide* documents on the Lenovo *Documentation* CD and the *Warranty Information* document that comes with the product.

First Edition (April 2019)

© Copyright Lenovo 2019
Portions © Copyright IBM Corporation 2014.

LIMITED AND RESTRICTED RIGHTS NOTICE: If data or software is delivered pursuant a General Services Administration "GSA" contract, use, reproduction, or disclosure is subject to restrictions set forth in Contract No. GS-35F-05925.

Lenovo and the Lenovo logo are trademarks of Lenovo in the United States, other countries, or both.

Contents

© Copyright Lenovo 2019

Preface

The *Lenovo Network Python Programming Guide for Cloud NOS 10.10* describes how to configure and use the Lenovo Cloud Network Operating System 10.10 software on the following Lenovo RackSwitches:

- Lenovo RackSwitch G8272. For documentation on installing the switch physically, see the *Lenovo RackSwitch G8272 Installation Guide*.
- Lenovo RackSwitch G8296. For documentation on installing the switch physically, see the *Lenovo RackSwitch G8296 Installation Guide*.
- Lenovo RackSwitch G8332. For documentation on installing the switch physically, see the *Lenovo RackSwitch G8332 Installation Guide*.
- Lenovo ThinkSystem NE1032 RackSwitch. For documentation on installing the switch physically, see the *Lenovo ThinkSystem NE1032 RackSwitch Installation Guide*.
- Lenovo ThinkSystem NE1032T RackSwitch. For documentation on installing the switch physically, see the *Lenovo ThinkSystem NE1032T RackSwitch Installation Guide*.
- Lenovo ThinkSystem NE1072T RackSwitch. For documentation on installing the switch physically, see the *Lenovo ThinkSystem NE1072T RackSwitch Installation Guide*.
- Lenovo ThinkSystem NE10032 RackSwitch. For documentation on installing the switch physically, see the *Lenovo ThinkSystem NE10032 RackSwitch Installation Guide*.
- Lenovo ThinkSystem NE2572 RackSwitch. For documentation on installing the switch physically, see the *Lenovo ThinkSystem NE2572 RackSwitch Installation Guide*.

Who Should Use This Guide

This guide is intended for network installers and system administrators engaged in configuring and maintaining a network. The administrator should be familiar with Ethernet concepts, IP addressing, Spanning Tree Protocol, and SNMP configuration parameters.

Additional References

Additional information about installing and configuring the switch is available in the following guides:

- *Lenovo Network Application Guide for Lenovo Cloud Network Operating System 10.10*
- *Lenovo Network Command Reference for Lenovo Cloud Network Operating System 10.10*
- *Lenovo Network Release Notes for Lenovo Cloud network Operating System 10.10*
- *Lenovo Network REST API Programming Guide for Lenovo Cloud Network Operating System 10.10*

Terminology

In every programming endeavor, terminology is used in a slightly different manner in different environments.

Following is a list of the terminology used in this guide.

Table 1. *Terminology Used in This Guide*

Term	Description
Function	Lists an action and associated arguments, for example: `python_get_vlan(`*vid*`)`
Function Arguments	Objects passed to a function when it is called inside a script or in the Python interpreter
N/OS Python API	Extensions to the Python library provided by Lenovo
Python scheduler	An engine to run scripts when specified events occurs.
Script Arguments	Strings passed to a script at run time

Typographic Conventions

The following table describes the typographic styles used in this book.

Table 2. *Typographic Conventions*

Typeface or Symbol	**Meaning**	**Example**
`ABC123`	This type is used for names of commands, files, and directories used within the text.	View the `readme.txt` file.
	It also depicts on-screen computer output and prompts.	`Switch#`
`ABC123`	This bold type appears in command examples. It shows text that must be typed in exactly as shown.	`Switch#` **`ping`**
<ABC123>	This italicized type appears in command examples as a parameter placeholder. Replace the indicated text with the appropriate real name or value when using the command. Do not type the brackets.	To establish a Telnet session, enter: `Switch#` **`telnet`** *<IP address>*
	This also shows book titles, special terms, or words to be emphasized.	Read your *User's Guide* thoroughly.
`{}`	Command items shown inside brackets are mandatory and cannot be excluded. Do not type the brackets.	`Switch#` **`cp {ftp|sftp}`**
`[]`	Command items shown inside brackets are optional and can be used or excluded as the situation demands. Do not type the brackets.	`Switch#` **`configure [device]`**
`|`	The vertical bar (**`|`**) is used in command examples to separate choices where multiple options exist. Select only one of the listed options. Do not type the vertical bar.	`Switch#` **`cp {ftp|sftp}`**
<AaBb123>	This block type depicts menus, buttons, and other controls that appear in graphical interfaces.	Click the **<Save>** button.

Chapter 1. Introduction to Python Scripting

The Lenovo Cloud Network Operating System (CNOS) version 10.10 Python function API is a set of libraries of API functions that are embedded into the Lenovo switch Command-Line Interface (CLI) to support script execution.

About Python

Python is a widely used general-purpose, high-level programming language. Its design philosophy emphasizes code readability, and its syntax allows programmers to express concepts in fewer lines of code than would be possible in other languages. The language provides constructs intended to enable clear programs on both a small and large scale. Like other dynamic languages, Python is often used as a scripting language, but is also used in a wide range of non-scripting contexts.

Python is supported by the Python Software Organization, which is open source with an active user community. Python provides comprehensive set of libraries that includes many built-in modules and the ability to write scripts and functional extensions. Organizations from NASA to gaming and data security companies use Python for development. Python version 2.7 is installed on the switch version 10.10.

About CNOS Python

Lenovo's CNOS Python API library extends the standard Python library with functions that allow you to write your own scripts to manage your switch.

CNOS Python comes with the following features:

- Automated switch provision and management
- The ability to perform switch monitoring tasks
- Automatic switch firmware update
- Automatic configuration file generation
- Notifications sent to users via email or system logger (syslog) messages

You can schedule CNOS Python scripts to run either at startup or when an event occurs. These scripts can send configuration and display commands to the switch, save variables, and send system log messages. Chapter 2, "Running Python Scripts via the ISCLI", contains information about how to run CNOS Python scripts in real time. Chapter 4, "Using the CNOS Python Scheduler", explains how to schedule a script to run when an event occurs.

Chapter 2. Running Python Scripts via the ISCLI

The most straightforward method to run a script on the switch is to execute it directly:

```
Switch# python <script filename> [arguments list]
```

The script will be run in the foreground. You can use **<Ctrl + C>** to stop the script execution.

For information about transferring scripts to the switch, see Chapter 3, “Managing Python Scripts“.

Running a Basic Script

The following is an example of a simple “Hello World!” script:

```
Switch# show script helloWorld.py

print "Hello world!"

Switch# python helloWorld.py

Hello world!
```

Running a Basic Script with Arguments

The following is an example of a basic script with arguments:

```
Switch# show script scriptWithArgs.py

import sys

for i in range(1, len(sys.argv)):
        print “Argument {0} is: {1}".format(i, sys.argv[i])

Switch# python scriptWithArgs.py 2 secondArgument 3rdArgument

Argument 1 is: 2
Argument 2 is: secondArgument
Argument 3 is: 3rdArgument
```

Entering and Exiting the Python Shell

You can enter the Python shell, the interactive mode of the Python interpreter, directly via the ISCLI **`python`** command. After entering the Python shell, you can get the online help for each Python API function and test it before calling it in your script.

Note: You must be a privileged administrator to use the Python shell.

To enter the Python shell, enter **`python`** at the switch command prompt.

```
Switch# python

>>>
```

To exit the Python shell, enter the following command or press **<Ctrl + D>**.

```
exit ()

Switch#
```

Chapter 3. Managing Python Scripts

Script files are saved in persistent storage on the switch, while the script log files are saved to volatile storage. The maximum storage for script files is 2.8 M bytes.

Lenovo Cloud Network Operating System (CNOS) for the switch provides the following managing actions on scripts:

- Downloading a Script from a TFTP Server
- Uploading a Script to a TFTP Server
- Editing a Script Directly on the Switch
- Deleting a Script
- Viewing A List of Script Files
- Viewing Script File Content
- Viewing Configured Scheduler Jobs

Downloading a Script from a TFTP Server

To download a TFTP script, use the following command:

```
Switch# cp tftp tftp://<server address>/<remote script> obs <local script> [vrf {<VRF instance name>|default|management}]
```

where:

Parameter	Description
server address	The IPv4 address of the TFTP server.
remote script	The path and filename of the remote script.
local script	The filename for the script on the switch.
VRF instance name	The Virtual Routing and Forwarding (VRF) instance name.
`default`	The default VRF instance.
`management`	The management VRF instance.

Uploading a Script to a TFTP Server

To upload a script to a TFTP server, use the following command:

```
Switch# cp obs <local script> tftp tftp://<server address>/<remote script> [vrf {<VRF
instance name>|default|management}]
```

where:

Parameter	Description
server address	The server address of the TFTP server.
local script	The filename for the script on the switch.
remote script	The path and filename of the remote script.
VRF instance name	The Virtual Routing and Forwarding (VRF) instance name.
default	The default VRF instance.
management	The management VRF instance.

Editing a Script Directly on the Switch

To create or edit a script directly on the switch, use the following command:

```
Switch# edit script <script filename>
```

where *script filename* is the name of your script.

Deleting a Script

To delete a script, use the following command:

```
Switch# no script <script filename>
```

where *script filename* is the name of your script.

To delete all Python scripts, use the following command:

```
Switch# no script all
```

Viewing A List of Script Files

To view a list of script files, use the following command:

```
Switch# show script
```

Viewing Script File Content

To view a script file, use the following command:

```
Switch# show script <script filename>
```

Viewing Configured Scheduler Jobs

To view the scheduler jobs configured on the switch, use the following command:

```
Switch# show script-job
```

Running jobs will be displayed in the running configuration display.

For more information about scheduling CNOS Python scripts, see Chapter 4, “Using the CNOS Python Scheduler“.

Chapter 4. Using the CNOS Python Scheduler

The Lenovo Cloud Network Operating System scheduler's main responsibility is to provide a programmatic mechanism to run Python scripts when specified events occur. These events are defined by switch administrators and can be triggered by a timer, which is aligned to the `cron` service.

You can schedule scripts to run as a response to an event using scheduler jobs and monitor the script execution.

© Copyright Lenovo 2019

Using the Python Scheduler

The CNOS Python scheduler is an engine that can run Python scripts at specified times or intervals, similar to the UNIX-based `cron` utility.

Creating a Scheduled Python Job

To create a job that runs a Python script at a scheduled time, use the command:

```
Switch(config)# script-job <Python script> [<arguments>] time {daily|hourly|
|monthly|reboot|weekly|yearly}
```

where:

Parameter	Description
Python script	The name of the Python script.
arguments	Any arguments the script needs.
`time`	Configure the job to run at specific times.
Crontab format	Run a script periodically, in the format of the UNIX `crontab` utility. See xx for more information on this argument.
`daily`	The scripts runs at the start of every day.
`hourly`	The script runs at the start of every hour.
`monthly`	The script runs at the start of every month.
`reboot`	The script runs when the switch is reloaded.
`weekly`	The script runs at the start of every week.
`yearly`	The script runs at the start of every year.

For example, to create a job to run the Python script `myScript.py` on a daily basis, enter:

```
Switch(config)# script-job myScript.py time daily
```

Using Crontab Format Arguments

The *Crontab format* date and time parameter uses the format:

<minute> <hour> <day of month> <month> <day of week>

where:

Parameter	**Description**
minute	Specifies the minutes of the hour. The *minute* parameter is an integer from 0 to 59.
hour	Specifies the hour. The *hour* parameter is an integer from 0 to 23.
day of month	Specifies the day of the month. The *day of month* is an integer from 1 to 31.
month	Specifies the month of the year. The *month* parameter is an integer from 1 to 12 or the three-letter abbreviation for the month, as follows: ● Jan ● Feb ● Mar ● Apr ● May ● Jun ● Jul ● Aug ● Sep ● Oct ● Nov ● Dec
day of week	Specifies the day of the week. The *day of week* parameter is an integer from 1 to 7 or the three-letter abbreviation for the day of the week, as follows: ● Mon ● Tue ● Wed ● Thu ● Fri ● Sat ● Sun

An asterisk in place of any of these fields means “use all values from first to last.”

Note: All time values must be surrounded by quotation marks.

For example, to run the Python script `myScript.py` every July the 4th at 10:55 A.M., regardless of the day of the week, use the following command:

```
Switch(config)# script-job myScript.py time “55 10 4 Jul *”
```

You can use ranges for the numeric values, separating them with commas or using hyphens for sequential ranges, such as “`1,2,5-9`” or “`0-4,8-12`”.

For example, to run the Python script `myScript.py` every day starting with July the 4th until July the 11th at 10:55 A.M. and 10:55 P.M., enter:

```
Switch(config)# script-job myScript.py time “55 10,22 4-11 7 *”
```

Note: Ranges or lists of month or day of week names are not allowed.

You can also specify step values or intervals using a slash (`/`). For example, to run the Python script `myScript.py` every July the 4th every two hours, enter:

```
Switch(config)# script-job myScript.py time “0 */2 4 Jul *”
```

Crontab format commands are executed when the minute, hour, and month of year fields match the current time, and when either the day of month or day of week match the current day.

Note: If you specify the day of a job’s execution by both day of month and day of week, the command will be run when either field matches the current date and time. For example, the Crontab format date “`30 4 1,15 * 5`” would cause a job to be run at 4:30 A.M. on the 1st and 15th of each month and every Friday.

Deleting a Job

To delete a job, use the following command:

```
Switch(config)# no script-job <Python filename>
```

Monitoring a Running Job

To monitor a running job, use the following command:

```
Switch# show script running
```

Stopping a Running Job

To stop a running job, use the following steps:

1. Check the list of running scripts:

```
Switch# show script running

Current running scripts:
1 myscript.py arg1 arg2
```

2. Copy the exact string from the list and use it as the argument for the following command:

```
Switch# stop running-script “<argument>”
```

Using the output from Step 1, the command would be:

```
Switch# stop running-script “myscript.py arg1 arg2”
```

Viewing Python Logs

To view a list of the log files created by OBS jobs, enter:

```
Switch# show script-log
```

To view an individual log file, enter:

```
Switch# show script-log <filename>
```

Creating Syslog Messages

Python scripts can send log messages to the system logger (syslog). These messages will be stored in the syslog repository and can be managed using the ISCLI commands.

System logs generated by a Python script have the same format as the current CNOS system log where the facility name is “`PYRUN`” and the mnemonic is “`OBS`”. The format is:

```
<timestamp> <hostname> %PYRUN-<SEVERITY>-OBS: [<LIB_NAME> | <THREAD_NAME>]
Description [@function:line]
```

For example:

```
2014-08-15T04:50:33+00:00 switch %PYRUN-6-OBS: Message=I am a testing log
from OBS
```

Chapter 5. Writing Python Scripts

This chapter describes script components, modules, and the API functions and arguments that you can use to create Python scripts to run on the switch. Python API functions extend the standard Python library to provide configuration, management, and monitoring abilities. These are located in several Python modules.

Script Components

The CNOS Python API contains the following modules:

- `systemApi`: Functions that open and close a Simple Management Interface (SMI) client connection.

  Note: You must import this module before importing anything else.
- `aaaApi`: Functions that manage the Authentication, Authorization and Accounting (AAA) switch configuration.
- `anyCastGwApi`: Functions that manage Anycast Gateway.
- `arpApi`: Functions that manage the Address Resolution Protocol (ARP) switch configuration.
- `aspApi`: Functions that manage ARP Suppression (ASP).
- `bfdApi`: Functions that manage the Bidirectional Forwarding Detection (BFD).
- `bgpApi`: Functions that manage the Border Gateway Protocol (BGP) switch configuration.
- `bootInfoApi`: Functions that manage the switch boot properties.
- `bootProfileApi`: Functions that manage switch boot profile properties.
- `dcbApi`: Functions that manage the Converged Enhanced Ethernet (CEE) switch configuration.
- `cliApi`: Functions that manage Command Line Interface configuration.
- `defaultIpAddressApi`: Functions that manage the Default IP Address configuration.
- `dhcpApi`: Functions that manage the Dynamic Host Configuration Protocol (DHCP) switch configuration.
- `dnsApi`: Functions that manage the Domain Name System (DNS) switch configuration.
- `dot1qEncapsApi`: Functions that manage Dot1Q encapsulation.
- `dot1xApi`: Functions that manage 802.1X authentication.
- `fdbApi`: Functions that manage the Forwarding Database (FDB) switch configuration.
- `fdbmApi`: Functions that manage MAC move loop detections.
- `fipsApi`: Functions that manage Federal Information Processing Standards (FIPS).
- `hostpCpyApi`: Functions that update image and configuration files via TFTP.
- `hscApi`: Functions that manage Hardware Switch Controller (HSC).
- `hwProfileApi`: Functions that manage hardware breakout profiles.
- `hostpRadiusApi`: Functions that manage the Remote Authentication Dial-In User Service (RADIUS) switch configuration.
- `hostpTacacsApi`: Functions that manage the Terminal Access Controller Access-Control System Plus (TACACS+) switch configuration.

- `igmpsnoopingApi`: Functions that manage Internet Group Management Protocol (IGMP) Snooping switch configuration.
- `ipApi`: Functions that manage the Internet Protocol (IP) switch configuration.
- `l2fApi`: Functions that manage Layer 2 Failover (L2F) configuration on the switch.
- `lacpApi`: Functions that manage the Link Aggregation Control Protocol (LACP) switch configuration.
- `lagApi`: Functions that manage the Link Aggregation Group (LAG) switch configuration.
- `ldapApi`: Functions that manage Lightweight Directory Access Protocol (LDAP).
- `lfdApi`: Functions that manage Link Flap Dampening (LFD).
- `lldpApi`: Functions that manage the Link Layer Discovery Protocol (LLDP) switch configuration.
- `moveNofifyApi`: Functions that manage MAC move notifications.
- `mpLoggingApi`: Functions that manage Mp Packet Logging.
- `mstpApi`: Functions that manage the Multiple Spanning Tree Protocol (MSTP) switch configuration.
- `natApi`: Functions that manage Network Address Translation (NAT).
- `nhopHealthcheckApi`: Functions that manage Nexthop health check.
- `ntpApi`: Functions that manage Network Type Protocol (NTP) authentication.
- `nwvApi`: Functions that manage Network Virtualization (NWV) authentication.
- `ospfApi`: Functions that manage the Open Shortest Path First (OSPF) switch configuration.
- `pbrApi`: Functions that manage Policy Based Routing (PBR).
- `platformApi`: Functions that manage the switch port configuration.
- `pvlanApi`: Functions that manage Private VLAN properties.
- `routeApi`: Functions that manage the switch static route configuration.
- `routemapApi`: Functions that manage routemaps.
- `secModeApi`: Functions that manage the switch security mode configuration.
- `subnetvlanApi`: Functions that manage Subnet VLAN configuration.
- `telemetryApi`: Functions that manage the Telemetry switch configuration.
- `ufpApi`: Functions that manage Unified Fabric Port (UFP).
- `vlagApi`: Functions that manage Virtual Link Aggregation Group (vLAG) information.
- `vlanApi`: Functions that configure VLAN properties.
- `vrfApi`: A function that manage Virtual Routing and Forwarding (VRF).
- `vrrpApi`: Functions that manage Virtual Router Redundancy Protocol (VRRP).

- `vxlanApi`: Functions that manage Virtual Extensible LAN (VXLAN) properties.
- `weightedEcmpApi`: Functions that manage the Equal Cost Multiple Paths (ECMP) switch configuration.

The following is a sample Python shell session:

```
Switch# python

>>> import systemApi
>>> systemApi.client_connect()
>>> systemApi.SystemInfo().get_systemInfo()

{'switch_type': 'Switch', 'fw_version': '10.4.1.0'}

>>> import lldpApi
>>> lldpApi.LldpNeighbor().python_lldp_get_all_neighbor()

[{'capability': 'BR', 'system name': 'Mars2', 'rx ttl': 120L, 'if_name':
'Ethernet1/8', 'system description': 'LENOVO RackSwitch SWITCH, LENOVO
NOS version 10.4.1.0'}]

>>> systemApi.client_disconnect()
>>> exit()

Switch#
```

Viewing Online Help

The Python API modules have built-in help. To obtain help on a particular module or class, enter:

```
>>> help(<module>[.<class>][.<function>])
```

For example, to get help on the `python_lldp_get_all_neighbor()` function from the previous example, enter the text shown in the following example:

```
>>> help(lldpApi.LldpNeighbor().python_lldp_get_all_neighbor)

Help on method python_lldp_get_all_neighbor in module lldpApi:

python_lldp_get_all_neighbor(self) method of lldpApi.LldpNeighbor
instance
    API's description: Get neighbor information of all ports
    Mandatory arguments: None
    Optional arguments: None
    Output: list of dictionaries of LLDP neighbor
    [
        {
            if_name(Str),
            capability(Int),
            rx ttl(Int),
            system name(Str),
            system description(Str)
        }
    ]
```

Python API Function Arguments

Python API functions have mandatory and optional arguments. Mandatory arguments must be set with correct types. Optional arguments will use their default values.

Python API functions can verify user arguments. The API functions can detect if mandatory arguments are missing or are in the incorrect type of mandatory arguments. If argument verification fails, it will report the error and not execute the API function.

Python API functions return useful information on either success or failure. For example: configuration API functions return `True` if the command is successful, and `False` if the command fails, and displays an error message. Query API functions return information from the switch.

All Python API functions use keyword arguments.

Script Examples

This section contains a set of sample Lenovo Python API scripts.

ARP Configuration Example

This script demonstrates how to use the Python API to create and delete an Address Resolution Protocol (ARP) entry.

```
import sys

#Import python object APIs from modules
from systemApi import *
from ipApi import *
from arpApi import *

#Create class instance
ipobj = IP()
arpobj = ARP()

#Calling client_connect to establish the SMI client-server connection
client_connect()

#Make sure the interface is one routed interface
ipobj.unset_bridge_port('Ethernet1/1')

#Calling set_ip_addr defined in module ipApi to set IP address
ipobj.set_ip_addr('Ethernet1/1', '10.0.0.1/8', 0)

#Calling set_ip_arp defined in module arpApi to create ARP entry
arpobj.set_ip_arp('10.0.0.100','0000.0000.0100', 'Ethernet1/1')
arpobj.set_ip_arp('10.0.0.101','0000.0000.0101', 'Ethernet1/1')

#Calling get_all_static_arp_entry to check if ARP entry is created
successfully
print arpobj.get_all_static_arp_entry('Ethernet1/1')

#Calling delete_ip_arp defined in module arpApi to delete ARP entry
arpobj.delete_ip_arp('10.0.0.100', 'Ethernet1/1')
arpobj.delete_ip_arp('10.0.0.101', 'Ethernet1/1')

#Calling get_all_static_arp_entry to check if ARP entry is created
successfully
print arpobj.get_all_static_arp_entry('Ethernet1/1')

#Calling client_disconnect to disconnect the SMI client-server connection
client_disconnect()
```

IP Configuration Example

The following script demonstrates how to use the Python API to configure IP interfaces.

```
#Import modules
import systemApi, ipApi

#Calling client_connect to establish the SMI client-server connection
systemApi.client_connect()

print ' #Configure port  '
print ipApi.IP().unset_bridge_port('Ethernet1/1')
print '\n'

print ' #Configure IP on a routed interface  '
print ipApi.IP().set_ip_addr('Ethernet1/1','11.0.0.10/16', 0 )
print '\n'

print ' #Verify IP interface'
print ipApi.IP().get_ipinfo('Ethernet1/1')
print '\n'

print ' #Verify IP interface'
print ipApi.IP().set_if_flagup('Ethernet1/11')
print '\n'

print ' #Configure IP on a routed interface  '
print ipApi.IP().set_ip_addr('Ethernet1/1','10.0.0.10/16', 0 )
print '\n'

print ' #Verify IP interface'
print ipApi.IP().get_ipinfo('Ethernet1/1')
print '\n'

print ' #Calling client_disconnect to disconnect the SMI client-server
connection'
systemApi.client_disconnect()
```

LAG Configuration Example

The following script demonstrates how to use the Python API to create, view, update, and delete a Link Aggregation Group.

```
#Import modules
import systemApi, lagApi

#Calling client_connect to establish the SMI client-server connection
systemApi.client_connect()

print ' #Create LAG '
print lagApi.LAG().python_create_lag_id({'lag_id': 1, 'interfaces' :
[{'lacp_prio': 32768, 'lacp_timeout': 'long', 'lag_mode':'lacp_active',
'if_name': 'Ethernet1/11'}, {'lacp_prio': 32768, 'lacp_timeout': 'long',
'lag_mode':'lacp_active', 'if_name': 'Ethernet1/12'}] })
print '\n'

print ' #Verify LAG information'
print lagApi.LAG().python_get_lag_id(1)
print '\n'

print ' #Update LAG '
print lagApi.LAG().python_update_lag_id_details({'lag_id': 1,
'interfaces' :  [{'lacp_prio': 100, 'lacp_timeout': 'long',
'lag_mode':'lacp_active', 'if_name': 'Ethernet1/11'}, {'lacp_prio': 100,
'lacp_timeout': 'long', 'lag_mode':'lacp_active', 'if_name':
'Ethernet1/12'}] })
print '\n'

print ' #Verify LAG information'
print lagApi.LAG().python_get_lag()
print '\n'

print ' #Delete LAG '
print lagApi.LAG().python_delete_lag_id(1)
print '\n'

print ' #Calling client_disconnect to disconnect the SMI client-server
connection'
systemApi.client_disconnect()
```

LLDP Configuration Example

The following script demonstrates how to use the Python API to administer Link Layer Discovery Protocol (LLDP).

```
#Import modules
import systemApi, lldpApi

#Calling client_connect to establish the SMI client-server connection
systemApi.client_connect()

print ' #Verify lldp reinit delay'
print lldpApi.LldpSystem().python_lldp_get_reinit_delay()
print '\n'

print ' #Verify lldp tx interval'
print lldpApi.LldpSystem().python_lldp_get_msg_tx_interval()
print '\n'

print ' #Verify lldp tx delay'
print lldpApi.LldpSystem().python_lldp_get_tx_delay()
print '\n'

print ' #Set lldp reinit delay'
print lldpApi.LldpSystem().python_lldp_set_reinit_delay(15)
print '\n'

print ' #Set lldp tx interval'
print lldpApi.LldpSystem().python_lldp_set_msg_tx_interval(2000)
print '\n'

print ' #Set lldp tx delay'
print lldpApi.LldpSystem().python_lldp_set_tx_delay(3)
print '\n'

print ' #Get lldp neighbor'
print lldpApi.LldpNeighbor().python_lldp_get_all_neighbor()
print '\n'

print ' #Calling client_disconnect to disconnect the SMI client-server
connection'
systemApi.client_disconnect()
```

VLAN Configuration Example

This script demonstrates how to use the Python API to administer VLANs.

```
#Import modules
import systemApi, vlanApi

#Calling client_connect to establish the SMI client-server connection
systemApi.client_connect()

print ' #Verify vlan status - default'
print('Check vlan status by calling the "python_get_vlan" method with no
argument')
print vlanApi.VlanSystem().python_get_vlan()
print '\n'

print('Check vlan 1-default')
print vlanApi.VlanSystem().python_get_vlan(1)
print '\n'

print ' #Create vlan 10 - test vlan'
print vlanApi.VlanSystem().python_create_vlan({'vlan_name':'TEST',
'vlan_id':10, 'admin_state': 'up'})

print ' #Verify that vlan 10 was created '
print vlanApi.VlanSystem().python_get_vlan(10)
print '\n'

print ' #Update vlan 10 - new_name vlan '
print vlanApi.VlanSystem().python_update_vlan_name(10,'new_name')
print '\n'

print ' #Verify vlan 10 '
print vlanApi.VlanSystem().python_get_vlan(10)
print '\n'

print ' #Update vlan 10 - admin_state down '
print vlanApi.VlanSystem().python_update_vlan_admin_state(10,'down')
print '\n'

print ' #Verify vlan 10 '
print vlanApi.VlanSystem().python_get_vlan(10)
print '\n'

print ' #Update vlan properties for a given interface'
print
vlanApi.VlanEthIf().python_update_vlan_properties({'if_name':'Ethernet1/1
1','bridgeport_mode':'access', 'pvid':1,'vlans':[1]})
print '\n'

print ' #Call get_vlan_properties for a given interface'
print vlanApi.VlanEthIf().python_get_vlan_properties('Ethernet1/1')
print '\n'

print ' #Delete vlan 10 - test vlan '
print vlanApi.VlanSystem().python_delete_vlan(10)
print '\n'

print ' #Calling client_disconnect to disconnect the SMI client-server
connection'
systemApi.client_disconnect()
```

Chapter 6. The CNOS Python API

This chapter explains the contents of all the modules included in the Lenovo Cloud Network Operating System:

AAA Module

This module manages the Authentication, Authorization and Accounting (AAA) configuration on the switch.

To use this module, in the Python file or in the Python interpreter, enter:

```
import hostpAAAApi
```

class AAA()

This class gets and sets AAA configurations.

get_accounting_def()

Displays the current accounting configuration.

Syntax

`get_accounting_def``()`

Returns

The current accounting configuration (string); `"group"`, followed by a list of up to eight AAA groups (optionally followed by `"local"`).

set_accounting_def()

Configures accounting on the switch.

Syntax

`set_accounting_def``(`*input_str, groups*`)`

where:

Parameter	**Description**
input_str	The AAA method type (string). Valid values: `"group"`, `"local"`.
groups	(Optional) The list of AAA groups (string). Up to eight AAA groups can be configured, separated by space. List of groups can be followed by `"local"`. **Note:** To configure the list of AAA groups, the variable *input_str* must be set to `"group"`.

Returns

Boolean (`True` on success, otherwise `False`).

get_authorization_cmds_def()

Displays the current User EXEC command mode authorization configuration.

Syntax

`get_authorization_cmds_def``()`

Returns

The current authorization configuration (string); `"group"`, followed by a list of up to eight AAA groups (optionally followed by `"local"`).

set_authorization_cmds_def()

Enables or disables User EXEC command mode authorization.

Syntax

`set_authorization_cmds_def``(`*input_str, groups*`)`

where:

Parameter	**Description**
input_str	The AAA method type (string). Valid values: `"group"`, `"local"`.
groups	(Optional) The list of AAA groups (string). Up to eight AAA groups can be configured, separated by space. List of groups can be followed by `"local"`. **Note:** To configure the list of AAA groups, the variable *input_str* must be set to `"group"`.

Returns

Boolean (`True` on success, otherwise `False`).

get_authorization_conf_cmds_def()

Displays the current configuration command mode authorization configuration.

Syntax

`get_authorization_conf_cmds_def``()`

Returns

The current authorization configuration (string); `"group"`, followed by a list of up to eight AAA groups (optionally followed by `"local"`).

set_authorization_conf_cmds_def()

Enables or disables configuration command mode authorization.

Syntax

`set_authorization_conf_cmds_def``(`*input_str, groups*`)`

where:

Parameter	**Description**
input_str	The AAA method type (string). Valid values: `"group"`, `"local"`.
groups	(Optional) The list of AAA groups (string). Up to eight AAA groups can be configured, separated by space. List of groups can be followed by `"local"`. **Note:** To configure the list of AAA groups, the variable *input_str* must be set to `"group"`.

Returns

Boolean (`True` on success, otherwise `False`).

get_authentication_login_con()

Displays the current console user login authentication configuration.

Syntax

`get_authentication_login_con``()`

Returns

The current console user login authentication configuration (string); `"group"`, followed by a list of up to eight AAA groups (optionally followed by `"local"` or `none`).

set_authentication_login_con()

Enables or disables console user login authentication.

Syntax

`set_authentication_login_con``(`*input_str, groups*`)`

where:

Parameter	**Description**
input_str	The AAA method type (string). Valid values: `"group"`, `"local"`.
groups	(Optional) The list of AAA groups (string). Up to eight AAA groups can be configured, separated by space. List of groups can be followed by `"local"` or `none`. **Note:** To configure the list of AAA groups, the variable *input_str* must be set to `"group"`.

Returns

Boolean (`True` on success, otherwise `False`).

get_authentication_login_def()

Displays the current remote user login authentication configuration.

Syntax

`get_authentication_login_def``()`

Returns

The current remote user login authentication configuration (string); `"group"`, followed by a list of up to eight AAA groups (optionally followed by `"local"` or `none`).

set_authentication_login_def()

Enables or disables remote user login authentication used for remote protocol connections such as SSH or Telnet.

Syntax

`set_authentication_login_def(`*input_str, groups*`)`

where:

Parameter	**Description**
input_str	The AAA method type (string). Valid values: `"group"`, `"local"`, or `none`.
groups	(Optional) The list of AAA groups (string). Up to eight AAA groups can be configured, separated by space. List of groups can be followed by `"local"` or `none`. **Note:** To configure the list of AAA groups, the variable *input_str* must be set to `"group"`.

Returns

Boolean (`True` on success, otherwise `False`).

get_authentication_login_err_enable()

Checks if error messages are displayed when users fail to authenticate.

Syntax

`get_authentication_login_err_enable``()`

Returns

The status of the error messages (string). Valid values: `"enable"` if error messages are displayed, `"disable"` if error message are not displayed.

set_authentication_login_err_enable()

Enables the display of error messages when users fail to authenticate.

Syntax

`set_authentication_login_err_enable``()`

Returns

Boolean (`True` on success, otherwise `False`).

unset_authentication_login_err_enable()

Disables the display of error messages when users fail to authenticate.

Syntax

`unset_authentication_login_err_enable``()`

Returns

Boolean (`True` on success, otherwise `False`).

get_maxfail_attempts()

Displays the maximum number of unsuccessful authentication attempts before a user is locked out.

Syntax

`get_maxfail_attempts``()`

Returns

The maximum number of unsuccessful authentication attempts.
An integer from 1-25.

set_maxfail_attempts()

Configures the maximum number of unsuccessful authentication attempts before a user is locked out.

Syntax

`set_maxfail_attempts``(`*maxfail_attempts*`)`

where:

Parameter	**Description**
maxfail_attempts	The maximum number of unsuccessful authentication attempts before a user is locked out. An integer from 1-25.

Returns

Boolean (`True` on success, otherwise `False`).

get_user_default_role()

Checks if users are allowed to login even if the TACACS+ server does not provide a default role.

Syntax

`get_user_default_role``()`

Returns

The status of the default user role configuration (string).
Valid values: `"enable"`, `"disable"`.

set_user_default_role()

Enables users to login even if the TACACS+ server does not provide a role. The default role is network-operator.

Syntax

`set_user_default_role``()`

Returns

Boolean (`True` on success, otherwise `False`).

unset_user_default_role()

Disables users to login even if the TACACS+ server does not provide a role. If this option is disabled then users without a role provided by the TACACS+ server will be unable to login.

Syntax

`unset_user_default_role``()`

Returns

Boolean (`True` on success, otherwise `False`).

get_groups()

Displays information about the configured AAA groups.

Syntax

```
get_groups()
```

Returns

A dictionary with information about the configured AAA groups:

Parameter	Description
group_name	The name of the AAA group (string).
type	The type of the AAA group (string). Valid values: `"TACACS+"`, `"RADIUS"`, or `"LDAP"`.

AnyCast Gateway Module

This module manages Anycast Gateway.

To use this module, in the Python file or in the Python interpreter, enter:

```
import anyCastGwApi
```

class AnycastGateway()

This class provides functions that manage anycast gateway.

get_anycast_gateway_address ()

Gets the anycast gateway MAC address.

Syntax

`get_anycast_gateway_address``()`

Returns

A dictionary containing anycast gateway MAC addresses:

Parameter	Description
anycast_gw_mac	The anycast gateway MAC address (string).

set_anycast_gateway_address ()

Sets the anycast gateway MAC address.

Syntax

`set_anycast_gateway_address``(`*mac_address*`)`

where:

Parameter	Description
mac_address	The anycast gateway MAC address (string).

Returns

Boolean (`True` on success, otherwise `False`).

get_anycast_gateway_in_use()

Gets if anycast gateway is in use.

Syntax

`get_anycast_gateway_in_use``()`

Returns

Boolean (`True` on success, otherwise `False`).

set_anycast_gateway_interface ()

Sets anycast gateway forwarding on a specified interface.

Syntax

`set_anycast_gateway_interface``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The interface name (string).

Returns

Boolean (`True` on success, otherwise `False`).

unset_anycast_gateway_interface ()

Disables anycast gateway forwarding on a specified interface.

Syntax

`unset_anycast_gateway_interface``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The interface name (string).

Returns

Boolean (`True` on success, otherwise `False`).

get_anycast_gateway_interface ()

Gets the anycast gateway configuration of a specified interface.

Syntax

`get_anycast_gateway_interface``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The interface name (string).

Returns

A dictionary containing interface and anycast gateway forwarding status:

Parameter	**Description**
interface	The interface name (string).
anycast_gw_status	The anycast gateway status (string). Valid values: `“enabled”`, `“disabled”`.

ARP Module

This module manages Address Resolution Protocol (ARP).

To use this module, in the Python file or in the Python interpreter, enter:

```
import arpApi
```

class ARP()

This class provides functions for managing ARP.

get_all_static_arp_entry()

Gets all static ARP entries.

Syntax

`get_all_static_arp_entry(`*if_name*`)`

where:

Parameter	Description
if_name	(Optional) The Ethernet interface name (string). **Note:** If specified, the interface must exist.

Returns

The IP address, MAC address, and interface name for all or the specified ARP entry:

Parameter	Description
if_name	The Ethernet interface name (string).
ip_addr	The IP address (string).
mac_addr	The MAC address. A string in the following format: "`xxxx.xxxx,xxxx`".

get_one_static_arp_entry()

Gets one static ARP entry for the specified interface.

Syntax

`get_one_static_arp_entry``(`*if_name, ip_addr*`)`

where:

Parameter	**Description**
if_name	The Ethernet interface name (string). **Note:** The interface must exist.
ip_addr	The IP address (string).

Returns

The static ARP entry, with the following parameters:

Parameter	**Description**
if_name	The Ethernet interface name (string).
ip_addr	The IP address (string).
mac_addr	The MAC address. A string in the following format: “`xxxx.xxxx,xxxx`”.

set_ip_arp()

Creates a static proxy ARP entry.

Syntax

`set_ip_arp``(`*ip_addr, mac_addr, if_name*`)`

where:

Parameter	**Description**
ip_addr	The IP address (string).
mac_addr	The MAC address. A string in the following format: “`xxxx.xxxx,xxxx`”.
if_name	The Ethernet interface name (string). **Note:** The interface must exist.

Returns

Boolean (`True` on success, otherwise `False`).

delete_ip_arp()

Deletes an ARP entry.

Syntax

`delete_ip_arp``(`*ip_addr, if_name*`)`

where:

Parameter	Description
ip_addr	The IP address (string).
if_name	The Ethernet interface name (string). **Note:** The interface must exist.

Returns

Boolean (`True` on success, otherwise `False`).

get_arp_sys_pro()

Gets the global ARP properties of the system.

Syntax

`get_arp_sys_pro``()`

Returns

The global ARP entry age out time:

Parameter	Description
ageout_time	The global ARP entry age out time, in seconds. An integer from 60-28800. Default value: `1500`.

set_arp_sys_pro()

Sets the global ARP properties of the system.

Syntax

`set_arp_sys_pro``(`*ageout_time*`)`

where:

Parameter	Description
ageout_time	The global ARP entry age out time, in seconds. An integer from 60-28800. Default value: `1500`.

Returns

Boolean (`True` on success, otherwise `False`).

get_arp_sys_interfaces()

Gets ARP properties for all interfaces or for the specified interface.

Syntax

`get_arp_sys_interfaces``(`*if_name*`)`

where:

Parameter	**Description**
if_name	(Optional) The Ethernet interface name (string). **Note:** If specified, the interface must exist.

Returns

ARP properties for all interfaces or for the specified interface:

Parameter	**Description**
if_name	The Ethernet interface name (string).
ageout_time	The global ARP entry age out time, in seconds. An integer from 60-28800. Default value: `1500`.

set_arp_sys_pro_interface()

Sets the ARP properties of the specified interface.

Syntax

`set_arp_sys_pro_interface``(`*if_name, ageout_time*`)`

where:

Parameter	**Description**
if_name	The Ethernet interface name (string). **Note:** If specified, the interface must exist.
ageout_time	The global ARP entry age out time, in seconds. An integer from 60-28800. Default value: `1500`.

Returns

Boolean (`True` on success, otherwise `False`).

get_arp_refresh()

Checks if ARP Refresh is enabled on the switch.

Syntax

`get_arp_refresh``()`

Returns

A dictionary containing the variable state (string).
Valid values: `”disabled”`, `”enabled”`.

set_arp_refresh()

Enables or disables ARP Refresh on the switch. With ARP Refresh enabled, ARP requests are sent after the timeout period for an ARP entry expires.

Syntax

`set_arp_refresh``(`*enable*`)`

where:

Parameter	**Description**
enable	The status of ARP refresh (string). Valid values: `”enabled”`, `”disabled”`.

Returns

Boolean (`True` on success, otherwise `False`).

ASP Module

This module manages ARP Suppression (ASP).

To use this module, in the Python file or in the Python interpreter, enter:

`import aspApi`

class ARP_Suppression()

This class provides functions that manage ARP suppression.

get_asp_status()

Gets the ARP suppression status.

Syntax

`get_asp_status``(`*vnid*`)`

where:

Parameter	Description
vnid	(Optional) The VXLAN ID. An integer from 1-16777214. **Note:** ARP suppression can be enabled for a specified VNID. If *vnid* is not specified, the API gets the ASP status for all VNIDs with ASP enabled.

Returns

A dictionary with ASP status:

Parameter	Description
vnid	The VXLAN ID. An integer from 1-16777214.
status	The ARP suppression status (string). Valid values: `“enabled”`, `“disabled”`. Default value: `“disabled”`.

set_asp_status()

Sets the ARP suppression status.

Syntax

`set_asp_status``(`*vnid, status*`)`

where:

Parameter	Description
vnid	The VXLAN ID. An integer from 1-16777214.
status	The ARP suppression status (string). Valid values: `"enable"`, `"disable"`. Default value: `"disable"`.

Returns

Boolean (`True` on success, otherwise `False`).

class ARP_Suppression_Entry()

This class provides functions that manage ARP suppression MAC entries.

get_asp_entry()

Gets the ARP suppression MAC entries for all VNIDs or for a specified VNID.

Syntax

`get_asp_entry``(`*vnid*`)`

where:

Parameter	Description
vnid	(Optional) The VXLAN ID. An integer from 1-16777214. **Note:** If *vnid* is not specified, the API gets the ASP MAC entries for all VNIDs with ASP enabled.

Returns

A dictionary with ASP entries:

Parameter	Description
vnid	The VXLAN ID. An integer from 1-16777214.
vm_ip	The VM IP address (string). Valid values: an IPv4 address.
vm_mac	The VM MAC address (string).

Parameter	Description
vtep_ip	The VTEP IP address (string). Valid values: an IPv4 address.
flag	The entry flag (string). Valid values: `"Static"`, `"Remote via BGP-EVPN"`, `"Local via Access port"`, `"Local via NPAD"`, `"Remote via NPAD"`, `"Remote via Network port"`.
age	The age of the ASP entry (string).

delete_asp_entry()

Deletes the ARP suppression MAC entries for all VNIDs or for a specified VNID.

Syntax

`delete_asp_entry``(`*vnid*`)`

where:

Parameter	Description
vnid	(Optional) The VXLAN ID. An integer from 1-16777214. **Note:** If *vnid* is not specified, the API deletes the ASP supression MAC entries for all VNIDs.

Returns

Boolean (`True` on success, otherwise `False`).

class ARP_Suppression_Vnet()

This class provides functions that manage ARP suppression virtual network information.

get_asp_vnet()

Gets the ARP suppression virtual network information.

Syntax

`get_asp_vnet(`*vnid***`)`**

where:

Parameter	Description
vnid	(Optional) The VXLAN ID. An integer from 1-16777214. **Note:** If *vnid* is not specified, the API gets ASP virtual network information for all VNIDs.

Returns

A dictionary with ASP virtual network information:

Parameter	Description
vnid	The VXLAN ID. An integer from 1-16777214.
status	The ARP suppression status (string). Valid values: `“enabled”`, `“disabled”`. Default value: `“disabled”`.
vrf_name	The VRF name (string).
interface	The interface name (string). Valid values: the Ethernet or LAG interface name. Default: `NULL`.
vlan	The VLAN ID. An integer from 1-4096. Default: `NULL`.

BFD Module

This module manages Bidirectional Forwarding Detection (BFD).

To use this module, in the Python file or in the Python interpreter, enter:

`import bfdApi`

class BFD()

This class provides functions to retrieve BFD neighbors information.

get_bfd_nbr_all()

Gets the BFD neighbors information.

Syntax

`get_bfd_nbr_all``(`*details, appl*`)`

where:

Parameter	**Description**
details	(Optional) Retrieves detailed BFD neighbors information (string). Valid value: `"details"`.
appl	(Optional) The application name. Filters BFD neighbors by registered protocols (string). Valid values: `"bgp"`, `"bfd"`, `"ospf"`, `"rib"`, `"nwv"`.

Returns

A list of dictionaries with neighbors information:

Parameter	**Description**
loc_addr	The BFD session source IP address (string). A valid IP address.
rem_addr	The BFD session destination IP address (string). A valid IP address.
loc_disc	The unique number used by the local system to identify that BFD session. An integer from 1-2147483647.
rem_disc	The unique number used by the remote system to identify that BFD session. An integer from 1-2147483647.
RH/RS	The Remote Heard/Remote State (string). Valid values: `"UP"`, `"DOWN"`, `"ADMIN_DOWN"`.
holdown	The session is declared down if no BFD packet is received in holdown (ms). An integer from 150-2997.

Parameter	**Description**
mult	The number of times a packet is missed before BFD declares the neighbor down. An integer from 3-50.
sess_state	The BFD session state (string). Valid values: `"UP"`, `"DOWN"`, `"ADMIN_DOWN"`.
if_name	The interface on which the BFD session is active (string).
echo_mode	Whether the BFD session has echo mode enabled or not (string). Valid values: `"enabled"`, `"disabled"`.
gtsm	Whether if BFD session has GTSM enabled or not (string). Valid values: `"enabled"`, `"disabled"`.
gtsm_ttl	Displays BFD GTSM TTL value. An integer from 1-255.
minTxInt	The rate in microseconds at which BFD control packets will be sent to BFD neighbors. An integer from 50-999.
minRxInt	The rate in microseconds at which BFD control packets will be expected. An integer from 50-999.
negotiated_minRxInt	The negotiated rate at which BFD control packets will be received from BFD neighbor. An integer from 50-999.
negotiated_multiplier	The number of times a packet is missed before BFD declares the session down, in microseconds. An integer from 3-50.
rx_Count	The number of received BFD packets. A positive integer.
tx_Count	The number of sent BFD packets. A positive integer.
registered_protocols	The protocol for which the BFD session is active (string). one of `"OSPF"`, `"BGP"`, `"RIB"`, `"BFD"`, `"NWV"`.
uptime	How long the BFD session has been up. A string in the following format: `"00:00:00"`.

get_bfd_nbr_by_ip()

Gets the BFD neighbors information based on the IP address.

Note: You can enter only the *source_ip* or *dest_ip*, not both.

Syntax

`get_bfd_nbr_by_ip``(`*source_ip, dest_ip*`)`

where:

Parameter	**Description**
source_ip	The BFD session source IP address (string). A valid IP address.
dest_ip	The BFD session destination IP address (string). A valid IP address.

Returns

A list of dictionaries with neighbors information:

Parameter	**Description**
loc_addr	The BFD session source IP address (string). A valid IP address.
rem_addr	The BFD session destination IP address (string). A valid IP address.
loc_disc	The unique number used by the local system to identify that BFD session. An integer from 1-2147483647.
rem_disc	The unique number used by the remote system to identify that BFD session. An integer from 1-2147483647.
RH/RS	The Remote Heard/Remote State (string). Valid values: `”UP”`, `”DOWN”`, `”ADMIN_DOWN”`.
holdown	The session is declared down if no BFD packet is received in holdown (ms). An integer from 150-2997.
mult	The number of times a packet is missed before BFD declares the neighbor down. An integer from 3-50.
sess_state	The BFD session state (string). Valid values: `”UP”`, `”DOWN”`, `”ADMIN_DOWN”`.
if_name	The interface on which the BFD session is active (string).

class BFD_CFG()

This class provides functions that manage BFD global configurations.

bfd_slow_timer_set()

Sets the BFD global slow timer value.

Syntax

`bfd_slow_timer_set``(`*slow_timer_val*`)`

where:

Parameter	**Description**
slow_timer_val	(The desired rate at which BFD will send control packets when the BFD session is down or when the BFD echo feature is enabled. An integer from 1000-30000.

Returns

Boolean (`True` on success, otherwise `False`).

bfd_slow_timer_unset()

Unsets the BFD global slow timer value.

Syntax

`bfd_slow_timer_unset``()`

Returns

Boolean (`True` on success, otherwise `False`).

bfd_gtsm_enable()

Enables BFD global Generalized TTL Security Mechanism (GTSM).

Syntax

`bfd_gtsm_enable``()`

Returns

Boolean (`True` on success, otherwise `False`).

bfd_gtsm_disable()

Disables BFD global GTSM.

Syntax

`bfd_gtsm_disable``()`

Returns

Boolean (`True` on success, otherwise `False`).

bfd_gtsm_ttl_set()

Sets the BFD global GTSM TLL value.

Syntax

`bfd_gtsm_ttl_set``(`*gtsm_ttl_value*`)`

where:

Parameter	**Description**
gtsm_ttl_value	Sets the desired BFD GTSM TLL. An integer from 1-255.

Returns

Boolean (`True` on success, otherwise `False`).

bfd_gtsm_ttl_unset()

Unsets the BFD global GTSM TLL value.

Syntax

`bfd_gtsm_ttl_unset``()`

Returns

Boolean (`True` on success, otherwise `False`).

class BFD_IF_CFG()

This class provides functions that create/delete static BFD sessions and configure BFD on a specified interface.

bfd_enable()

Enables BFD feature on a specified interface.

Syntax

`bfd_enable``(`*if_name, addr_family*`)`

where:

Parameter	Description
if_name	The interface name (string).
addr_family	The address family (string). Valid values: `"ipv4"`, `"ipv6"`.

Returns

Boolean (`True` on success, otherwise `False`).

bfd_disable()

Disables BFD feature on a specific interface.

Syntax

`bfd_disable``(`*if_name, addr_family*`)`

where:

Parameter	Description
if_name	The interface name (string).
addr_family	The address family (string). Valid values: `"ipv4"`, `"ipv6"`.

Returns

Boolean (`True` on success, otherwise `False`).

bfd_echo_enable()

Enables BFD echo feature on a specified interface.

Syntax

`bfd_echo_enable``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The interface name (string).

Returns

Boolean (`True` on success, otherwise `False`).

bfd_echo_disable()

Disables BFD echo feature on a specified interface.

Syntax

`bfd_echo_disable``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The interface name (string).

Returns

Boolean (`True` on success, otherwise `False`).

bfd_interval_set()

Sets the BFD intervals.

Syntax

`bfd_interval_set``(`*min_tx, min_rx, multiplier, if_name, addr_family*`)`

where:

Parameter	**Description**
min_tx	The desired rate at which BFD will be able to send packets to the BFD neighbors. An integer from 59-999.
min_rx	The desired rate at which BFD will be able to receive packets from the BFD neighbors. An integer from 50-999.
multiplier	The desired number of times a packet can be missed before BFD will declare the neighbor down. An integer from 3-50.
if_name	(Optional) The interface name (string). **Note:** If the interface name is not specified, BFD global intervals are set.
addr_family	(Optional) The address family (string). Valid values: `"ipv4"`, `"ipv6"`. **Note:** If the address family is not specified, all addresses families will be considered.

Returns

Boolean (`True` on success, otherwise `False`).

bfd_interval_unset()

Unsets the BFD intervals.

Syntax

`bfd_interval_unset``(`*if_name, addr_family*`)`

where:

Parameter	**Description**
if_name	(Optional) The interface name (string). **Note:** If the interface name is not specified, all BFD global intervals will be unset.
addr_family	(Optional) The address family (string). Valid values: ”`ipv4`”, ”`ipv6`”.

Returns

Boolean (`True` on success, otherwise `False`).

bfd_auth_set()

Configures BFD authentication on a specified interface.

Syntax

`bfd_auth_set``(`*if_name, auth_type, auth_key_chain, auth_key_id, auth_key*`)`

where:

Parameter	**Description**
if_name	The interface name (string).
auth_type	SHA256 authentication type (string). Valid values: ”`simple`”, ”`keyed-md5`”, ”`keyed-sha1`”, ”`keyed-sha256`”, ”`meticulous-keyed-md5`”, ”`meticulous-keyed-sha1`”, ”`meticulous-keyed-sha256`”.
auth_key_chain	The authentication key chain (string).
auth_key_id	The authentication key ID. An integer from 0-255.
auth_key	The authentication key string.

Returns

Boolean (`True` on success, otherwise `False`).

bfd_auth_unset()

Unsets BFD authentication on a specified interface.

Syntax

`bfd_auth_unset``(`*if_name*`)`

where:

Parameter	Description
if_name	The interface name (string).

Returns

Boolean (`True` on success, otherwise `False`).

bfd_add_nbr()

Creates a static BFD session.

Syntax

`bfd_add_nbr``(`*addr_family, source_ip, dest_ip, if_name, multihop, admin_down, non-persistent*`)`

where:

Parameter	Description
addr_family	The address family (string). Valid values: `"ipv4"`, `"ipv6"`.
source_ip	The BFD session source IP address (string). A valid IPv4 or IPv6 address.
dest_ip	The BFD session destination IP address (string). A valid IPv4 or IPv6 address.
if_name	The interface name (string).
multihop	(Optional) Set BFD session type as multi-hop (boolean). Valid values: `True`, `False`. Default value: `False`.
admin_down	Set BFD session state to `ADMIN DOWN` (boolean). Valid values: `True`, `False`. Default value: `False`.
non_persistent	Set the BFD session as non-persistent (boolean). Valid values: `True`, `False`. Default value: `False`.

Returns

Boolean (`True` on success, otherwise `False`).

bfd_del_nbr()

Deletes a BFD session.

Syntax

`bfd_del_nbr``(`*if_name, loc_disc*`)`

where:

Parameter	**Description**
if_name	The interface name (string).
loc_disc	The local discriminator. Unique number used by the local system to identify the BFD session. An integer from 0-2147483647.

Returns

Boolean (`True` on success, otherwise `False`).

class BFD_MH_CFG()

This class provides functions to manage BFD multi-hop sessions.

bfd_mh_interval_set()

Configures the intervals for multi-hop BFD sessions.

Syntax

`bfd_mh_interval_set``(`*min_tx, min_rx, multiplier, ip_addr, addr_family*`)`

where:

Parameter	**Description**
min_tx	The desired rate at which BFD will be able to send packets to the BFD neighbors. An integer from 59-999.
min_rx	The desired rate at which BFD will be able to receive packets from the BFD neighbors. An integer from 50-999.
multiplier	The desired number of times a packet can be missed before BFD will declare the neighbor down. An integer from 3-50.
ip_addr	The IP address of the BFD neighbor (string). A valid IP address.
addr_family	The address family (string). Valid values: `"ipv4"`, `"ipv6"`.

Returns

Boolean (`True` on success, otherwise `False`).

bfd_mh_interval_unset()

Unsets the intervals for multi-hop BFD sessions.

Syntax

`bfd_mh_interval_unset``(`*ip_addr, addr_family*`)`

where:

Parameter	**Description**
ip_addr	The IP address of the BFD neighbor (string). A valid IP address.
addr_family	The address family (string). Valid values: `"ipv4"`, `"ipv6"`.

Returns

Boolean (`True` on success, otherwise `False`).

bfd_mh_auth_set()

Sets authentication for multi-hop BFD sessions.

Syntax

`bfd_mh_auth_set``(`*ip_addr, addr_family, auth_type, auth_key_chain, auth_key_id, auth_key*`)`

where:

Parameter	**Description**
ip_addr	The IP address of the BFD neighbor (string). A valid IP address.
addr_family	The address family (string). Valid values: `"ipv4"`, `"ipv6"`.
auth_type	SHA256 authentication type (string). one of `"simple"`, `"keyed-md5"`, `"keyed-sha1"`, `"keyed-sha256"`, `"meticulous-keyed-md5"`, `"meticulous-keyed-sha1"`, `"meticulous-keyed-sha256"`.
auth_key_chain	The authentication key chain (string).
auth_key_id	The authentication key ID. An integer from 0-255.
auth_key	The authentication key string.

Returns

Boolean (`True` on success, otherwise `False`).

bfd_mh_auth_unset()

Unsets authentication for multi-hop BFD sessions.

Syntax

`bfd_mh_auth_unset``(`*ip_addr, addr_family*`)`

where:

Parameter	**Description**
ip_addr	The IP address of the BFD neighbor (string). A valid IP address.
addr_family	The address family (string). Valid values: `"ipv4"`, `"ipv6"`.

Returns

Boolean (`True` on success, otherwise `False`).

BGP Module

This module manages Border Gateway Protocol (BGP) configurations.

To use this module, in the Python file or in the Python interpreter, enter:

`import bgpApi`

class BGP()

This class provides functions to get and set BGP configurations.

python_bgp_get_global_statistics()

Gets BGP global statistics.

Syntax

`python_bgp_get_global_statistics``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, “`default`”, “`all`”. Default value: “`default`”.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing BGP global statistics:

Parameter	**Description**
vrf_name	Virtual Routing and Forwarding name (string).
stats	A dictionary containing global statistics.
in_msgs	Received message number. A positive integer.
out_msgs	Send message number. A positive integer.
bytes_in	Bytes received. A positive integer.
bytes_out	Bytes sent. A positive integer.
open_in	Open message input count. A positive integer.
open_out	Open message output count. A positive integer.
update_in	Update message input count. A positive integer.
update_out	Update message ouput count. A positive integer.

Parameter	**Description**
keepalive_in	Keepalive input count. A positive integer.
keepalive_out	Keepalive output count. A positive integer.
notify_in	Notify input count. A positive integer.
notify_out	Notify output count. A positive integer.
refresh_in	Route Refresh input count. A positive integer.
refresh_out	Route Refresh output count. A positive integer.
dynamic_cap_in	Dynamic Capability input count. A positive integer.
dynamic_cap_out	Dynamic Capability output count. A positive integer.

python_bgp_clear_global_statistics()

Gets BGP global statistics.

Syntax

`python_bgp_clear_global_statistics`(*vrf_name*)

where:

Parameter	**Description**
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, “`default`”, “`all`”. Default value: “`default`”.

Returns

Boolean (`True` on success, otherwise `False`).

python_show_bgp_peer_adj_routes()

Shows BGP neighbor received and advertised routes.

Syntax

`python_show_bgp_peer_adj_routes``(`*adjin, peer_ip, vrf_name, af_name, subaf_name*`)`

where:

Parameter	**Description**
adjin	Number used to select adjacent routes. Valid values: ● `1` - received adjacent routes ● `0` - advertised adjacent routes
peer_ip	Neighbor IP address (string). A valid IPv4 or IPv6 address.
af_name	(Optional) Address family name (string). Valid values: `“ipv4”`, `”ipv6”`. Default value: `“ipv4”`.
vrf_name	(Optional) VRF name (string). Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.
subaf_name	(Optional) Subaddress family name (string). Valid values: `”unicast”`, `”multicast”`. Default value: `”unicast”`.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing BGP neighbor received and advertised routes:

Parameter	**Description**
status	Router status (string). Valid values: ● `“s”` - suppressed ● `“d”` - damped ● `“h”` - history ● `“*”` - valid ● `“>”` - best ● `“i”` - internal
network	Route destination IP address (string). A valid IPv4 or IPv6 address.
mask_len	Route mask length. An integer from 0-32.
nexthop	Route next hop (string). A valid IP address.
metric	Route Multi-Exit Discriminator attribute. An integer from 0-4294967295.

Parameter	Description
local_pref	Route local preference attribute. An integer from 0-4294967295.
aspath	Route AS path attribute (string). An AS path VTY string.
aspath4B	Route 4B AS path (string). An AS path VTY string.
weight	Route weight attribute. An integer from 0-65535.
origin	Route origin attribute (string). Valid values: ● `“i”` - IGP ● `“e”` - EGP ● `“?”` - incomplete

python_bgp_get_status()

Shows whether BGP is enabled or disabled globally.

Syntax

`python_bgp_get_status``(`*vrf_name*`)`

where:

Parameter	Description
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.

Returns

The BGP global status. Valid values: `“enable”`, `”disable”`, or `False` if an error occurred.

python_bgp_get_router_id()

Gets the BGP router ID.

Syntax

`python_bgp_get_router_id``(`*vrf_name*`)`

where:

Parameter	Description
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.

Returns

The BGP router ID (string). A valid IP address, or `False` if an error occurred.

python_bgp_get_as_number()

Gets the BGP AS number.

Syntax

`python_bgp_get_as_number``()`

Returns

The BGP AS number. An integer from 0-4294967295. Default value: `0`.

python_bgp_get_hold_down_timer()

Gets the BGP hold down interval.

Syntax

`python_bgp_get_hold_down_timer``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.

Returns

The hold down timer value, in MS. An integer from 1-3600. Default value: `180`, or `False` if an error occurred.

python_bgp_get_keep_alive_timer()

Gets the BGP keep alive interval.

Syntax

`python_bgp_get_keep_alive_timer``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.

Returns

The keep alive timer value, in MS. An integer from 1-3600. Default value: `60`, or `False` if an error occurred.

python_bgp_get_enforce_first_as()

Shows whether BGP global enforce-first-AS is enabled or disabled.

Syntax

`python_bgp_get_enforce_first_as``(`*vrf_name*`)`

where:

Parameter	Description
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.

Returns

The BGP global enforce-first-AS status. Valid values: `”enable”`, `”disable”`, or `False` if an error occurred.

python_bgp_get_fast_external_failover()

Shows whether BGP global fast-external-failover is enabled or disabled.

Syntax

`python_bgp_get_fast_external_failover``(`*vrf_name*`)`

where:

Parameter	Description
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.

Returns

The BGP global fast-external-failover status. Valid values: `”enable”`, `”disable”`, or `False` if an error occurred.

python_bgp_get_log_neighbor_changes()

Shows whether BGP global log-neighbor-changes is enabled or disabled.

Syntax

`python_bgp_get_log_neighbor_changes``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.

Returns

The BGP global log-neighbor-changes status.
Valid values: `”enable”`, `”disable”`, or `False` if an error occurred.

python_bgp_get_as_local_cnt()

Gets the BGP Autonomous System (AS) local count.

Syntax

`python_bgp_get_as_local_cnt``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.

Returns

The BGP local AS count: an integer from 0-64. Default value: `0`, or `False` if an error occurred.

python_bgp_get_maxas_limit()

Gets the BGP maximum AS limit.

Syntax

`python_bgp_get_maxas_limit``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.

Returns

The maximum number of Autonomous Systems. An integer from 0-2000. Default value: `0`, or `False` if an error occurred.

python_bgp_get_synchronization()

Shows whether BGP global synchronization is enabled or disabled.

Syntax

`python_bgp_get_synchronization``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.

Returns

The BGP global synchronization status (string). Valid values: `“enable”`, `“disable”`, or `False` if an error occurred.

python_bgp_get_bestpath_cfg()

Gets BGP best path configuration.

Syntax

`python_bgp_get_bestpath_cfg``(`*vrf_name*`)`

where:

Parameter	Description
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing the best path configuration:

Parameter	Description
always-compare-med	Allow comparing MED from different neighbors (string). Valid values: `”enable”`, `”disable”`.
compare-routeid	Compare route IDs for identical EBGP paths (string). Valid values: `”enable”`, `”disable”`.
as-path-ignore	Ignore as-path length in selecting a route (string). Valid values: `”enable”`, `”disable”`.
tie-break-on-age	Whether to prefer the old route when `“compare-route-id”` is not set (string). Valid values: `”enable”`, `”disable”`.
compare-confed-aspath	Allow comparing confederation AS path length (string). Valid values: `”enable”`, `”disable”`.
dont-compare-originator-id	Don't compare originator IDs for BGP (string). Valid values: `”enable”`, `”disable”`.
med-confed	Compare MED among confederation paths (string). Valid values: `”enable”`, `”disable”`.
med-missing-as-worst	Treat missing MED as the least preferred one (string). Valid values: `”enable”`, `”disable”`.
med-remove-recv-med	Whether to remove received MED attribute (string). Valid values: `”enable”`, `”disable”`.
med-remove-send-med	Whether to remove send MED attribute (string). Valid values: `”enable”`, `”disable”`.

Parameter	Description
as-path multipath-relax	Relax AS-Path restriction when choosing multipaths (string). Valid values: `"enable"`, `"disable"`.
med-non-deterministic	Best MED path among paths not selected from same AS (string). Valid values: `"enable"`, `"disable"`.

python_bgp_get_confed_id()

Gets the BGP confederation identifier.

Syntax

`python_bgp_get_confed_id``(`*vrf_name*`)`

where:

Parameter	Description
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `"default"`, `"all"`. Default value: `"default"`.

Returns

The BGP routing domain confederation AS. An integer from 0-65535, or `False` if an error occurred.

python_bgp_get_confederation_peers()

Gets the BGP confederation peers.

Syntax

`python_bgp_get_confederation_peers``(`*vrf_name*`)`

where:

Parameter	Description
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `"default"`, `"all"`. Default value: `"default"`.

Returns

The number of peer autonomous systems in the BGP confederation.
An integer from 1-65535, or `False` if an error occurred.

python_bgp_get_graceful_helper_status()

Shows whether BGP graceful helper is enabled or disabled.

Syntax

`python_bgp_get_graceful_helper_status``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.

Returns

The BGP graceful helper status (string). Valid values: `“enable”`, `”disable”`, or `False` if an error occurred.

python_bgp_get_graceful_stalepath_time()

Gets the BGP stale path time.

Syntax

`python_bgp_get_graceful_stalepath_time``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.

Returns

The delay value, in seconds, to remove BGP routes marked as stale.
An integer from 1-3600, or `False` if an error occurred.

python_bgp_get_cluster_id()

Gets the BGP route reflector cluster ID.

Syntax

`python_bgp_get_cluster_id`(*vrf_name*)

where:

Parameter	**Description**
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing the following parameters:

Parameter	**Description**
vrf_name	The VRF name (string).
cluster_id	The route reflector cluster ID. An integer from 1-4294967295, or a valid IP address (string).

python_show_ip_bgp()

Gets BGP Routing Information Base (RIB) information.

Syntax

`python_show_ip_bgp``(`*af_name, vrf_name*`)`

where:

Parameter	Description
af_name	(Optional) Address family name (string). Valid values: `“ipv4”`, `”ipv6”`, or `”l2vpn”`.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing RIB information:

Parameter	Description
status	Route status code (string). Valid values: ● `“s”` - suppressed ● `“d”` - damped ● `“h”` - history ● `“*”` - valid ● `“>”` - best ● `“i”` – internal
network	Route destination IP address (string). A valid IPv4 or IPv6 address. For L2VPN, the network field displays: ● EVPN type-1 prefix: `“[1]:[ESI]:[EthTag]”` ● EVPN type-2 prefix: `“[2]:[ESI]:[EthTag]:[MAClen]:[MAC]”` ● EVPN type-3 prefix: `“[3]:[EthTag]:[IPlen]:[OrigIP] “` ● EVPN type-5 prefix: `“[5]:[EthTag]:[IPlen]:[IPPrefix]”`
nextHopGlobal	Route nexthop IPv6 address (string). Not used for IPv4.
nextHopLocal	Route nexthop (string). A valid IPv4 or IPv6 address.
weight	Route weight attribute. An integer from 0-65535.
pathInfo	Route path information (string). A valid AS path VTY string.

Parameter	**Description**
aspath4B	Route 4B AS path (string). A valid AS path VTY string.
aspath	Route AS path attribute (string). A valid AS path VTY string.
origin	Route origin attribute (string). Valid values: ● `“i”` - IGP ● `“e”` - EGP ● `“?”` - incomplete
medvalue	Multi-exit discriminator value if the MED attribute is missing and missing-as-worst is set. An integer from 0-4294967294.
med	Multi-exit discriminator value. An integer from 0-4294967294.

python_show_ip_bgp_network()

Gets detailed information about a BGP route.

Syntax

`python_show_ip_bgp_network``(`*ip_address, netmask, af_name, vrf_name*`)`

where:

Parameter	**Description**
ip_address	The BGP network IP address (string). A valid IPv4 or IPv6 address.
netmask	Network mask: ● IPv4: An integer from 0-32 ● IPv6: An integer from 0-128
af_name	(Optional) Address family name (string). Valid values: `“ipv4”`, `”ipv6”`.
vrf_name	(Optional) VRF name (string). Valid values: the VRF name, `“default”`, `“all”`. Default value: `“default”`.

Returns

Multiple possible return values:

- `False` if an error occurred

- a dictionary containing BGP route information:

Parameter	**Description**
table_entry_for	Route IP address/mask (string). A valid IP address and net mask.
best	Whether this is the best path (string). Valid values: “Yes”, ”No”.
advertised	Whether the route is advertised to any peers (string). Valid values: “Yes”, ”No”.
adv_ebgp	Whether the route is advertised to an EBGP peer (string). Valid values: “Yes”, ”No”.
adv_out_locas	Whether the route is advertised outside the local AS (string). Valid values: “Yes”, ”No”.
agg_supress	Whether advertisements are suppressed by an aggregate (string). Valid values: “Yes”, ”No”.
adv_non_peer_group	IP address advertised to non peer-group peers (string).
adv_peer_groups	IP address advertised to peer groups (string).
not_adv	Not advertised to any peer (string). **Note:** This value only appears if “true”.
as_path_str	Route path information (string). A valid AS path VTY string.
aggregator_as	Aggregator AS number. A positive integer
aggregator_as4	Aggregator 4-byte AS number. A positive integer
aggregator_address	Aggregator address (string).
rec_from_rr_client	Received from route reflector client (string). Valid value: “Yes”. **Note:** This value only appears if it has been set.
suppressed	Suppressed due to dampening (string). Valid value: “Yes”. **Note:** This value only appears if it has been set.
history_entry	History entry (string). Valid value: “Yes”. **Note:** This value only appears if it has been set.
nexthop_address	Route nexthop (string). A valid IPv4 or IPv6 address.
peer	Peer address (string)
inaccessible	Whether the RIB is can be accessed (string). Valid values: “Yes”, “No”.
igpmetric	IGP metric value (string). Valid value: “No”. **Note:** This value only appears if it is “No”.

Parameter	**Description**
from_peer	Whether the from peer address can be accessed (string). Valid value: “`No`”. **Note:** This value only appears if it is “`No`”.
orig_id	Whether the originator ID can be accessed (string). **Note:** This value only appears if it is “`No`”.
next_hop_local_ip	Whether the next-hop IP address can be accessed (string). A valid IP address or “`No`”. **Note:** The value “`No`” only appears if it is inaccessible.
origin	Originating protocol. A string up to 25 characters long.
metric	Metric (string). Valid values: one of the metric values, “`removed`”.
local_pref	Local preference value. Only appears if set. A positive integer.
weight	Route weight attribute. An integer from 0-65535. **Note:** This value only appears if it is set.
label	Label. Only appears if set. A positive integer.
valid	Whether the path is valid (string). Valid value: “`Yes`”. **Note:** This value only appears if the path is valid.
stale	Whether the state is stale (string). Valid value: “`Yes`”. **Note:** This value only appears if the state is stale.
type	The route type. A string up to 30 characters long.
multipath_candidate	Whether this is a multipath candidate (string). Valid values: “`Yes`”, ”`No`”.
installed	Whether installed (string). Valid values: “`Yes`”, ”`No`”.
synchronized	Whether synchronized (string). Valid values: “`Yes`”, ”`No`”.
atomic_aggregate	Whether this is an atomic aggregate (string). Valid values: “`Yes`”, ”`No`”. **Note:** This value only appears if it is “`Yes`”.
community	Community string. A string up to 128 characters long.
extended community	Extended community string. A string up to 128 characters long.
originator	The originator route ID. A string up to 64 characters long.
cluster_id	Cluster ID. A string up to 128 characters long.

Parameter	Description
reuse_info	Reuse information. A string up to 32 characters long.
last update	Last update time. A string up to 256 characters long.

python_show_l2vpn_bgp_network()

Shows the BGP routing table.

Syntax

`python_show_l2vpn_bgp_network``(`*keyword, value, vni, rd*`)`

where:

Parameter	Description
keyword	The route type (string). Valid vlaues: `“mac”`, `”esi”`.
value	A MAC or an Ethernet segment ID. A string with an ESI value, or a MAC value in the following format: `“EEEE.EEEE.EEEE”`.
vni	The Virtual Network Identifier. An integer from 1-16777214.
rd	(Optional) The route distinguisher. A string in the following format: `”IP_address:nn”`.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing the route list of BGP for a certain address family:

Parameter	Description
table_entry_for	Route IP address/mask (string). A valid IP address and net mask.
best	Wether this is the best pats (string). Valid values: `”Yes”`, `”No”`.
advertised	Whether the route is advertised to any peers (string). Valid values: `”Yes”`, `”No”`.
adv_ebgp	Whether the route is advertised to an EBGP peer (string). Valid values: `“Yes”`, `”No”`.
adv_out_locas	Whether the route is advertised outside the local AS (string). Valid values: `“Yes”`, `”No”`.
agg_supress	Whether the advertisements are suppressed by an aggregate. (string). Valid values: `”Yes”`, `”No”`.
adv_non_peer_group	The non peer-group name.

Parameter	**Description**
adv_peer_group	The peer-group name (string).
not_adv	Not advertised to any peer (string).
as_path_str	Route AS path attribute. An AS path VTY string.
aggregator_as	Aggregator AS number.
aggregator_as4	Aggregator 4-byte AS number.
aggregator_address	Aggregator address.
rec_from_rr_client	Received from route reflector client (string). Valid values: `"Yes"`, `"No"`.
suppressed	Suppressed due to dampening (string). Valid values: `"Yes"`, `"No"`.
history_entry	History entry (string). Valid values: `"Yes"`, `"No"`.
nexthop_address	Route next-hop. A valid IP address.
peer	Peer address (string).
inaccessible	Whether the RIB is can be accessed (string). Valid values: `"Yes"`, `"No"`.
igpmetric	The IGP metric value.
from_peer	The peer address.
orig_id	The originator ID.
next_hop_local_ip	The next-hop IP address. A valid IP address.
origin	Originating protocol. A string up to 25 characters long.
metric	Whether one of the metric values is removed (string). Valid values: `"Yes"`, `"No"`.
local_pref	Local preference value.
weight	Route weight attribute. An integer from 0-65535.
label	The label. A positive integer.
valid	Whether the paths is valid (string). Valid values: `"Yes"`, `"No"`.
stale	Whether the state is stale (string). Valid values: `"Yes"`, `"No"`.
type	The route type. A string up to 30 characters long.
multipath_candidate	Whether this is a multipath candidate (string). Valid values: `"Yes"`, `"No"`.

Parameter	**Description**
installed	Whether installed (string). Valid values: `"Yes"`, `"No"`.
synchronized	Whether synchronized (string). Valid values: `"Yes"`, `"No"`.
atomic_aggregate	Whether this is an atomic aggregate (string). Valid values: `"Yes"`, `"No"`.
community	The community string.
extended community	The extended community string.
originator	The originator ID. A string up to 46 characters long.
cluster id	The cluster ID. A string up to 128 characters long.
reuse_info	Reuse information. A string up to 32 characters long.
last update	Last update time. A string up to 256 characters long.

python_show_ip_bgp_summary()

Gets IP, IPv6, L2VPN BGP summary information.

Syntax

`python_show_ip_bgp_summary``(`*af_name, vrf_name, subaf_name*`)`

where:

Parameter	**Description**
af_name	(Optional) Address family name (string). Valid values: `"ipv4"`, `"ipv6"` or `"l2vpn"`. Default value: `"ipv4"`.
vrf_name	(Optional) VRF name (string). Valid values: the VRF name, `"default"`, `"all"`. Default value: `"default"`.
subaf_name	(Optional) Subsequent Address Family Identifier name (string). Valid values: `"unicast"`, `"epvn"`. Default value: `"unicast"`.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing BGP route information:

Parameter	**Description**
route-id	Router ID (string). A valid IPv4 address.
table version	The table version. A positive integer.

Parameter	**Description**
neighbor	The neighbor IP address (string). A valid IPv4 or IPv6 address.
peer version	Peer version. A positive integer.
as	The Autonomous System (as). A positive integer.
peer as	Peer AS. A positive integer.
open in	Number of received open messages. A positive integer.
open out	Number of sent open messages. A positive integer.
update in	Number of received updates. A positive integer.
update out	Number of sent updates. A positive integer.
keepalive in	Number of received keepalives. A positive integer.
keepalive out	Number of sent keepalive messages. A positive integer.
notify in	Notify input count. A positive integer.
notify out	Notify output count. A positive integer.
refresh in	Number of received route refresh. A positive integer.
refresh out	Number of sent route refresh messages. A positive integer.
dynamic cap in	Dynamic capabilities input count. A positive integer.
dynamic cap out	Dynamic capabilities output count. A positive integer.
uptime	The BGP process uptime, in the following format: “HH:MM:SS”.
state	The BGP process state. A positive integer.

python_show_ip_bgp_neighbors()

Gets BGP neighbor details.

Syntax

`python_show_ip_bgp_neighbors``(`*ip_address, vrf_name*`)`

where:

Parameter	**Description**
ip_address	(Optional) Neighbor IP address (string). A valid IPv4 or IPv6 address. If no IP address is supplied, this function displays neighbor information for all IPv4 and IPv6 neighbors.
vrf_name	(Optional) VRF name (string). Valid values: the VRF name, `”default”`, `”all”`. Default value: `”default”`.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing BGP route information:

Parameter	**Description**
neighbor	Neighbor address (string).
vrfname	VRF name (string).
remote_as	AS number. An integer from 1-65535.
local_as	Local AS number. An integer from 1-65535.
af_name	Address family (string). Valid values: `"ipv4"`, `"ipv6"`, `"l2vpn"`.
af_name_2nd	Second address family (string). Valid values: `"ipv4"`, `"ipv6"`, `"l2vpn"`.
table version	Table version. A positive integer.
neighbor version	Neighbor version. A positive integer.
index val	Neighbor index value. A positive integer.
index offset	Index offset. A positive integer.
index mask	Index mask. A positive integer.
link type	Link type (string). Valid values: `”internal”`, `”external”`.
version	Version. A positive integer.
description	Description. A string up to 150 characters long.

Parameter	**Description**
remote router-ID	Remote router ID (string).
admin	Admin state. A string up to 40 characters long.
ifbound	Whether the interface is bound (string). Valid values: `“No interface binding”`, `“Interface bound”`.
state	Neighbor state (string). Valid values: `"Idle"`, `"Connect"`, `"Active"`, `"OpenSent"`, `"OpenConfirm"`, `"Established"`, `"Invalid"`.
dyncap_adv	Dynamic capability advertised, only if advertised (string). Valid value: `"Dynamic cap advertised"`.
dyncap_rec	Dynamic capability received, only if received (string). Valid value: `"Dynamic cap received"`.
refresh_adv	Refresh capability advertised, only if advertised (string). Valid value: `"Refresh advertised"`.
refresh_new_rec	Refresh New received, only if received (string). Valid value: `"Refresh New received"`.
refresh_old_rec	Refresh Old received, only if received (string). Valid value: `"Refresh Old received"`.
ext_asn_adv	Extended ASN capability advertised (string). Valid value: `"Extended ASN Capability advertised"`.
ext_asn_rec	Extended ASN capability received (string). Valid value: `"Extended ASN Capability received"`.
afc_adv	Address family unicast sent (string). Valid value: `"Address family Unicast sent"`.
afc_recv	Address family unicast received (string). Valid value: `"Address family Unicast received"`.
afc_vpn_adv	Address family VPN sent (string). Valid value: `"Address family VPN sent"`.
afc_vpn_recv	Address family VPN received (string). Valid value: `"Address family VPN received"`.
afc_mcast_adv	Address family multicast sent (string). Valid value: `"Address family Multicast sent"`.
afc_mcast_recv	Address family multicast received (string). Valid value: `"Address family Multicast received"`.
uptime	Uptime in the following format: `HH:MM:SS`.
peer-group name	Peer IP address. A string up to 32 characters long.

Parameter	**Description**
holdtime	Holdtime. A positive integer.
keepalive	Keepalive time. A positive integer.
conf holdtime	Configured holdtime. A positive integer.
conf keepalive	Configured keepalive time. A positive integer.
recvMsg	Number of received messages. A positive integer.
recvNotf	Number of received notifications. A positive integer.
recvQueue	Received messages queue count. A positive integer.
sentMsg	Number of sent messages. A positive integer.
sentNotf	Number of sent notifications. A positive integer.
sentQueue	Sent messages queue count. A positive integer.
refresh_in	Number of route refresh messages received. A positive integer.
refresh_out	Number of route refresh messages sent. A positive integer.
routeadv	Number of router advertisements. A positive integer.
update_if	Update interface. A string up to 150 characters long.
update_source	Update source address. A string up to 150 characters long.
established	Established count. A positive integer.
dropped	Dropped count. A positive integer.
prefix overflow	Whether there is a prefix overflow (string). Valid values: `”Yes”`, `”No”`.
ttl	Time to live value. A positive integer.
local address	Local neighbor IP address (string).
local port	Local port number. A positive integer.
remote address	Remote peer IP address (string).
remote port	Remote port number. A positive integer.
nextHopAddress	Next-hop address (string).
nextHopLocalV6	Next-hop address (link local). A string.
nextHopGlobalV6	Next-hop address (global). A string.
shared network	Shared network. A string up to 128 characters long.
next conn retry	Number of retries. A positive integer.
err notif	Whether there was an error notification (string). Valid values: `“sent”`, `“received”`.

Parameter	Description
last_reset_time	Last reset time in the following format: `HH:MM:SS`.
err code	Code string. A string up to 32 characters long.
err subcode	Subcode string. A string up to 32 characters long.
rmap_name	Default originating route map. A positive integer.

python_bgp_get_af_distance_config()

Gets BGP distance information.

Syntax

`python_bgp_get_af_distance_config`(*af_name, saf_name, vrf_name*)

where:

Parameter	Description
af_name	Address family name (string). Valid values: `"ipv4"`, `"ipv6"`.
saf_name	Subsequent Address Family Identifier name (string). Valid values: `"unicast"`, `"multicast"`.
vrf_name	(Optional) VRF name. Valid values: the VRF name, `"default"`, `"all"`. Default value: `"default"`.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing BGP distance information:

Parameter	Description
vrf_name	VRF name (string). Valid values: the VRF name, `"default"`, `"all"`. Default value: `"default"`.
distance_ebgp	Distance for routes external to the AS. An integer from 0-255.
distance_ibgp	Distance for routes internal to the AS. An integer from 0-255.
distance_local	Distance for routes local to the AS. An integer from 0-255.

python_bgp_get_af_global_config()

Gets BGP global configuration information.

Syntax

`python_bgp_get_af_global_config`(*af_name, saf_name, vrf_name*)

where:

Parameter	Description
af_name	Address family name (string). Valid values: `”ipv4”`, `”ipv6”`, `”l2vpn”`.
saf_name	Subsequent Address Family Identifier name (string). Valid values: `“unicast”`, `“evpn”`.
vrf_name	(Optional) VRF name (string). Valid values: the VRF name, `”default”`, `”all”`. Default value: `”default”`.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing BGP global configuration information:

Parameter	Description
vrf_name	VRF name (string). Valid values: the VRF name, `”default”`, `”all”`. Default value: `”default”`.
cc_reflection	Client-to-client reflect (string). Valid values: `”enable”`, `”disable”`.
synchronization	Perform IGP synchronization (string). Valid values: `”enable”`, `”disable”`.
network_ synchronization	Perform IGP synchronization on network routes (string). Valid values: `”enable”`, `”disable”`.

python_bgp_get_af_maximum_paths_config()

Gets BGP multipath ECMP number configuration information.

Syntax

`python_bgp_get_af_maximum_paths_config``(`*af_name, saf_name, vrf_name*`)`

where:

Parameter	**Description**
af_name	Address family name (string). Valid values: `"ipv4"`, `"ipv6"`.
saf_name	Subsequent Address Family Identifier name (string). Valid values: `"unicast"`, `"multicast"`.
vrf_name	(Optional) VRF name. Valid values: the VRF name, `"default"`, `"all"`. Default value: `"default"`.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing BGP global configuration information:

Parameter	**Description**
vrf_name	VRF name (string). Valid values: the VRF name, `"default"`, `"all"`. Default value: `"default"`.
ibgp_max_number	IBGP multipath maximum ECMP number. An integer from 1-32.
ebgp_max_number	EBGP multipath maximum ECMP number. An integer from 1-32.

python_bgp_get_af_nexthop_trigger_delay_config()

Gets the BGP nexthop trigger-delay configuration.

Syntax

`python_bgp_get_af_nexthop_trigger_delay_config`(*af_name, saf_name*)

where:

Parameter	Description
af_name	Address family name (string). Valid values: `“ipv4”`, `”ipv6”`.
saf_name	Subsequent Address Family Identifier name (string). Valid values: `“unicast”`, `“multicast”`.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing BGP nexthop trigger-delay configuration information:

Parameter	Description
critical	Nexthop changes affecting reachability. An integer from 1-4294967295.
non-critical	Nexthop changes affecting metric. An integer from 1-4294967295.

python_bgp_get_af_aggregate_config()

Gets the BGP aggregate configuration.

Syntax

`python_bgp_get_af_aggregate_config`(*af_name, saf_name, vrf_name*)

where:

Parameter	**Description**
af_name	Address family name (string). Valid values: `“ipv4”`, `”ipv6”`. Default value; both.
saf_name	Subsequent Address Family Identifier name (string). Valid values: `“unicast”`, `“multicast”`.
vrf_name	(Optional) VRF name. Valid values: the VRF name, `”default”`, `”all”`. Default value: `”default”`.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing BGP aggregate information:

Parameter	**Description**
vrf_name	VRF name (string). Valid values: the VRF name, `”default”`, `”all”`. Default value: `”default”`.
prefix	Aggregate prefix (string). An IP address in the following format: ● `“A.B.C.D/M”` ● `“X:X::X:X/M”`
type	Aggregate type (string). Valid values: ● `“as_set”` - Generate AS set path information. ● `“summary_only”` - Filter more specific routes from updates. ● `“as_set_summary_only”` - Both `“as-set”` and `“summary-only”`.

python_bgp_get_af_dampening_config()

Gets the BGP dampening configuration.

Syntax

`python_bgp_get_af_dampening_config``(`*af_name, saf_name, vrf_name*`)`

`where:`

Parameter	**Description**
af_name	Address family name (string). Valid values: `“ipv4”`, `”ipv6”`.
saf_name	Subsequent Address Family Identifier name (string). Valid values: `“unicast”`, `“multicast”`.
vrf_name	(Optional) VRF name. Valid values: the VRF name, `”default”`, `”all”`. Default value: `”default”`.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing BGP dampening information:

Parameter	**Description**
vrf_name	VRF name (string). Valid values: the VRF name, `”default”`, `”all”`. Default value: `”default”`.
half_life	Reachability Half-life time, in minutes, for the penalty. An integer from 1-45.
reuse_penalty	Value to start reusing a route. An integer from 1-20000.
max_suppress	Maximum duration, in minutes, to suppress a stable route. An integer from 1-255.
unreach_half_life	Unreachability half life time, in minutes. An integer from 1-255.
rmap_name	Route-map name. A string up to 64 characters long.

python_bgp_get_af_network_config()

Gets the BGP network configuration.

Syntax

`python_bgp_get_af_network_config``(`*af_name, saf_name, vrf_name*`)`

where:

Parameter	**Description**
af_name	Address family name (string). Valid values: `”ipv4”`, `”ipv6”`.
saf_name	Subsequent Address Family Identifier name (string). Valid values: `“unicast”`, `“multicast”`.
vrf_name	(Optional) VRF name. Valid values: the VRF name, `”default”`, `”all”`. Default value: `”default”`.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing BGP network information:

Parameter	**Description**
vrf_name	VRF name (string). Valid values: the VRF name, `”default”`, `”all”`. Default value: `”default”`.
prefix	Aggregate prefix (string). An IP address in the following format: ● `“A.B.C.D/M”` ● `“X:X::X:X/M”`
backdoor	Whether a BGP backdoor route is specified (string). Valid values: `“enable”`, `”disable”`.
rmap_name	Route map name. A string up to 63 characters long.

python_bgp_get_af_redistribute_config()

Gets the BGP redistribute configuration.

Syntax

`python_bgp_get_af_redistribute_config``(`*af_name, saf_name, vrf_name*`)`

where:

Parameter	Description
af_name	Address family name (string). Valid values: `“ipv4”`, `”ipv6”` or `”l2vpn”`.
saf_name	Subsequent Address Family Identifier name (string). Valid values: `“unicast”`, `”evpn”`.
vrf_name	(Optional) VRF name. Valid values: the VRF name, `”default”`, `”all”`. Default value: `”default”`.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing BGP redistribute configuration information:

Parameter	Description
vrf_name	VRF name (string). Valid values: the VRF name, `”default”`, `”all”`. Default value: `”default”`.
redist_direct	Whether redistribute direct is enabled (string). Valid values: `“enable”`, `”disable”`.
direct_rmap_name	Route map name for redistribute direct. A string up to 63 characters long.
redist_host_info	Whether redistribute host information is enabled (string). Valid values: `“enable”`, `”disable”`.
redist_ospf	Whether redistribute OSPF is enabled (string). Valid values: `“enable”`, `”disable”`.
ospf_rmap_name	Route map name for redistribute OSPF (string). A string up to 63 characters long.
redist_static	Whether redistribute static is enabled (string). Valid values: `“enable”`, `”disable”`.
static_rmap_name	Route map name for redistribute static (string). A string up to 63 characters long.

python_bgp_put_af_redistribute_config()

Configures the BGP redistribute configuration.

Syntax

`python_bgp_put_af_redistribute_config``(`*af_name, saf_name, redist_direct, direct_rmap_name, redist_ospf, ospf_rmap_name, redist_static, static_rmap_name, redist_host_info, vrf_name*`)`

where:

Parameter	**Description**
af_name	Address family name (string). Valid values: `"ipv4"`, `"ipv6"` or `"l2vpn"`.
saf_name	Subsequent Address Family name (string). Valid values: `"unicast"`, `"evpn"`.
redist_direct	(Optional) Whether redistribute direct is enabled (string). Valid values: `"enable"`, `"disable"`.
direct_rmap_name	(Optional) Route map name for redistribute direct. A string up to 63 characters long.
redist_ospf	(Optional) Whether redistribute OSPF is enabled (string). Valid values: `"enable"`, `"disable"`. Default value: `none`.
ospf_rmap_name	(Optional) Route map name for redistribute OSPF. A string up to 63 characters long.
redist_static	(Optional) Whether redistribute static is enabled (string). Valid values: `"enable"`, `"disable"`.
static_rmap_name	(Optional) Route map name for redistribute static. A string up to 63 characters long.
redist_host_info	(Optional) Whether redistribute host info is enabled (string). Valid values: `"enable"`, `"disable"`.
vrf_name	(Optional) VRF name (string). Valid values: the VRF name, `"default"`. Default value: `"default"`..

Returns

Boolean (`True` on success, otherwise `False`).

python_show_ip_bgp_neighbor_stats()

Gets BGP neighbor details.

Syntax

`python_show_ip_bgp_neighbor_stats`(*peer_ip, arg, vrf_name*)

where:

Parameter	Description
peer-ip	Neighbor IP address (string). Valid values: one or more valid IPv4 or IPv6 addresses.
arg	The type of statistics (string). Valid values: `“keepalive”`, `“notification”`, `“open”`, `“update”`, `“recv_msgs”`, `“send_msgs”`.
vrf_name	(Optional) VRF name (string). Valid values: the VRF name, `”default”`. Default value: `”default”`..

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing BGP neighbor detail information, based on the requested parameter

Parameter	Description
keepalive in	Keepalive input count. A positive integer.
keepalive out	Keepalive output count. A positive integer.
received notification	Notify input count. A positive integer.
sent notification	Notify output count. A positive integer.
received open	Open message input count. A positive integer.
sent open	Open message output count. A positive integer.
received updates	Update message input count. A positive integer.
sent updates	Update message output count. A positive integer.
received messages	Received messages. A positive integer.
sent messages	Sent messages. A positive integer.

python_show_ip_bgp_neighbors_cfg()

Displays BGP neighbor configuration information.

Syntax

`python_show_ip_bgp_neighbors_cfg``(`*ip_address, vrf_name*`)`

where:

Parameter	**Description**
ip_address	(Optional) The IP address of the BGP neighbor (string).
vrf_name	(Optional) The VRF instance for the BGP neighbor (string). Default value: `“default”`.

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary showing BGP neighbor information:

Parameter	**Description**
neighbor	The IP address of the BGP neighbor (string).
vrf_name	The VRF instance of the BGP neighbor (string).
remote_as	The current remote AS number. An integer from 1 - 4294967295.
local_as	The current local AS number. An integer from 1 - 4294967295.
address_family	The BGP neighbor address family (string). Valid values: `”ipv4”`, `”ipv6”`.
advertisement_interval	The configured minimum time interval, in seconds, between consecutive BGP updates. An integer from 1-65535.
bfd	The status of BFD (string). Valid values: `“enabled”`, `”disabled”`, `”multihop enabled”`.
connectio_ retry_time	The connection retry time, in seconds. An integer from 1-65535.
description	The BGP neighbor description (string).
disallow_infinite_ holdtime	Whether the configuration of infinite hold-time is disallowed (string). Valid values: `“Yes”`, `”No”`.
do_not_capability_ negotiate	Whether capability negotiations are disabled (string). Valid values: `“Yes”`, `”No”`.

Parameter	**Description**
advertise_dynamic_capability	Whether dynamic capability advertisements are enabled (string). Valid values: `“Yes”`, `”No”`.
egbp_multihop	The number of EBGP multi-hops. An integer from 1-255.
remote_private_as	Whether the removal of private AS numbers from outbound routes updates is enabled (string). Valid values: `“Yes”`, `”No”`.
maximum_peers	The maximum number of peers configured for the prefix of the BGP neighbor. A string from 1-96.
password	The encrypted password for the BGP neighbor (string).
shutdown	Whether the BGP neighbor is shut down (string). Valid values: `“Yes”`, `”No”`.
peer_holdtime	The time interval, in seconds, the switch awaits before transitioning the BGP neighbor to IDLE state, if the switch doesn’t receive an update or keep-alive message from the neighbor. An integer from 0-3600.
peer_keepalive	The time interval, in seconds, the switch awaits before sending another keep-alive message to the BGP neighbor. An integer from 0-3600.
connection_mode_passive	Whether the initiations of TCP sessions with the BGP neighbor are disabled (string). Displays whether the BGP neighbor is shut down. Valid values: `“Yes”`, `”No”`.
ttl_security_hops	The minimum number of TTL router hops an IP packet must have to not be discarded. An integer from 1-254.
update_source	The source of the BGP session and updates. A string containing ethernet port, VLAN, and loopback interfaces information.
weight	The default weight of routes incoming from the BGP neighbor. An integer from 1-65535.
allow_as_in	Whether AS paths with the local AS number are accepted by the switch (string). Valid values: `“Yes”`, `”No”`.
default_originate	Whether a default route to the BGP neighbor is configured (string). Valid values: `“Yes”`, `”No”`.
default_originate_map	The name of the route map for the default route (string).
prefix_list_in	The prefix list for routes incoming from the BGP neighbor (string).
prefix_list_out	The prefix list for routes outgoing to the BGP neighbor (string).

Parameter	**Description**
maximum_prefix	The maximum number of prefixes that can be received from the BGP neighbor. An integer from 1-15872.
maximum_prefix_ warning	Whether warning messages are generated only when the maximum prefix limit is exceeded (string). Valid values: `“Yes”`, `”No”`.
maximum_prefix_ threshold_percent	The percentage of the maximum prefix limit at which the switch starts to generate a warning message. An integer from 1-100.
next_hop_self	Whether next-hop calculations for the BGP neighbor are disabled (string). Valid values: `“Yes”`, `”No”`.
filter_list_in	The AS path ACL for routes incoming from the BGP neighbor (string).
filter_list_out	The AS path ACL for routes outgoing to the BGP neighbor (string).
route_map_in	The applied route map for routes incoming from the BGP neighbor (string).
route_map_out	The applied route map for routes outgoing to the BGP neighbor (string).
route_reflector_client	Whether the BGP neighbor is configured as a route reflector client (string). Valid values: `“Yes”`, `”No”`.
send_community	Whether community attributes are sent to the BGP neighbor (string). Valid values: `“Yes”`, `”No”`.
send_community_ extended	Whether extended community attributes are sent to the BGP neighbor (string). Valid values: `“Yes”`, `”No”`.
soft_reconfiguration_ inbound	Whether the switch is configured to store BGP neighbor updates (string). Valid values: `“Yes”`, `”No”`.
unsuppress_map	The name of the route map configured to selectively unsuppress suppressed routes (string).

python_put_ip_bgp_neighbors_cfg()

Configures a BGP neighbor.

Syntax

`python_put_ip_bgp_neighbors_cfg`(*ip_address, dict_neigh, vrf_name*)

where:

Parameter	**Description**
ip_address	The IP address of the BGP neighbor (string).
dict_neigh	The dictionary containing the BGP neighbor configuration details.
vrf_name	(Optional) The VRF instance for the BGP neighbor (string).

The *dict_neigh* dictionary contains the following configuration details:

Parameter	**Description**
address family	The neighbor address family (string). Valid values: `“ipv4 unicast”`, `”ipv6 unicast”`, `”l2vpn evpn”`. **Note:** For `“l2vpn evpn”`, only the following parameters are allowed: *address family, remote as, allow as in, route reflector client, send community extended*.
remote as	The current remote AS number. An integer from 1-4294967295.
local as	The current local AS number. An integer from 1-4294967295.
advertisement interval	The configured minimum time interval, in seconds, between consecutive BGP updates. An integer from 0-65535.
bfd	The status of BFD (string). Valid vlaues: `“enabled”`, `”disabled”`, `”multihop enabled”`.
connection retry time	The connection retry time, in seconds. An integer from 0-65535.
description	The BGP neighbor description (string).
disallow infinite holdtime	Whether the configuration of infinite hold-time is disallowed (string). Valid values: `“yes”`, `”no”`.
do not capability negotiate	Whether capability negotiations are disabled (string). Valid values: `“yes”`, `”no”`.
advertise dynamic capability	Whether dynamic capability advertisements are enabled (string). Valid values: `“yes”`, `”no”`.

Parameter	**Description**
EBGP multihop	The number of EBGP multi-hops. An integer from 1-255.
remote private as	Whether the removal of private AS numbers from outbound routes updates is enabled (string). Valid values: “yes”, ”no”.
maximum peers	The maximum number of peers configured for the prefix of the BGP neighbor. A string from 1-96.
password	The encrypted password for the BGP neighbor (string).
unencrypt-password	Whether the password is unencrypted (string). Valid values: “yes”, ”no”.
shutdown	Whether the BGP neighbor is shut down (string). Valid values: “yes”, ”no”.
peer holdtime	The time interval, in seconds, the switch awaits before transitioning the BGP neighbor to IDLE state, if the switch doesn’t receive an update or keep-alive message from the neighbor. An integer from 0-3600.
peer keepalive	The time interval, in seconds, the switch awaits before sending another keep-alive message to the BGP neighbor. An integer from 0-3600.
connection-mode passive	Whether the initiations of TCP sessions with the BGP neighbor are disabled (string). Displays whether the BGP neighbor is shut down. Valid values: “yes”, ”no”.
ttl security hops	The minimum number of TTL router hops an IP packet must have to not be discarded. An integer from 1-254.
update-source	The source of the BGP session and updates. A string containing ethernet port, VLAN, and loopback interfaces information.
weight	The default weight of routes incoming from the BGP neighbor. An integer from 0-65535.
allow as in	Whether AS paths with the local AS number are accepted by the switch (string). An integer from 1-10.
default originate	Whether a default route to the BGP neighbor is configured (string). Valid values: “yes”, ”no”.
default originate rmap	The name of the route map for the default route (string).
prefix-list in	The prefix list for routes incoming from the BGP neighbor (string).
prefix-list out	The prefix list for routes outgoing to the BGP neighbor (string).

Parameter	**Description**
maximum-prefix	The maximum number of prefixes that can be received from the BGP neighbor. An integer from 1-15872.
maximum-prefix warning	Whether warning messages are generated only when the maximum prefix limit is exceeded (string). Valid values: `“yes”`, `”no”`.
maximum-prefix threshold percent	The percentage of the maximum prefix limit at which the switch starts to generate a warning message. An integer from 1-100.
next-hop-self	Whether next-hop calculations for the BGP neighbor are disabled (string). Valid values: `“yes”`, `”no”`.
filter-list in	The AS path ACL for routes incoming from the BGP neighbor (string).
filter-list out	The AS path ACL for routes outgoing to the BGP neighbor (string).
route-map in	The applied route map for routes incoming from the BGP neighbor (string).
route-map out	The applied route map for routes outgoing to the BGP neighbor (string).
route reflector client	Whether the BGP neighbor is configured as a route reflector client (string). Valid values: `“yes”`, `”no”`.
send community	Whether community attributes are sent to the BGP neighbor (string). Valid values: `“yes”`, `”no”`.
send community extended	Whether extended community attributes are sent to the BGP neighbor (string). Valid values: `“yes”`, `”no”`.
soft reconfiguration inbound	Whether the switch is configured to store BGP neighbor updates (string). Valid values: `“yes”`, `”no”`.
unsuppress-map	The name of the route map configured to selectively unsuppress suppressed routes (string).

Returns

Boolean (`True` on success, otherwise `False`).

python_bgp_set_unnumbered()

Globally enables BGP unnumbered on the switch.

Syntax

`python_bgp_set_unnumbered``(`*as_number, set_bfd, vrf_name*`)`

where:

Parameter	**Description**
as_number	The BGP AS number. An integer from 1-4294967295.
set_bfd	(Optional) BFD option (string). Valid values: `”enabled”`, `”disabled”`. Default value: `”disabled”.`
vrf_name	(Optional) The VRF name (string). Default value: `"default"`.

Returns

Boolean (`True` on success, otherwise `False`).

python_bgp_unset_unnumbered()

Globally disables BGP unnumbered on the switch.

Syntax

`python_bgp_unset_unnumbered``(`*as_number, vrf_name*`)`

where:

Parameter	**Description**
as_number	The BGP AS number. An integer from 1-4294967295.
vrf_name	(Optional) The VRF name (string). Default value: `"default"`.

Returns

Boolean (`True` on success, otherwise `False`).

python_bgp_get_dscp()

Gets the BGP DSCP marking value, or `False` if an error occurred.

Syntax

`python_bgp_get_dscp``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	(Optional) VRF name (string). Valid values: `“data”`, `”default”`. Default value: `”default”`.

Returns

The DSCP marking value. An integer from 0-63.

python_bgp_put_dscp()

Sets the BGP DSCP marking value.

Syntax

`python_bgp_put_dscp``(`*dscp, vrf_name*`)`

where:

Parameter	**Description**
dscp	The DSCP marking value. An integer from 0-63.
vrf_name	(Optional) VRF name (string). Valid values: `“data”`, `”default”`. Default value: `”default”`.

Returns

Boolean (`True` on success, otherwise `False`).

Boot Information Module

This module gets and sets switch boot properties.

To use this module, in the Python file or in the Python interpreter, enter:

```
import bootInfoApi
```

class BootInfo()

This class provides functions to manage boot properties.

get_boot()

Gets detailed boot information.

Syntax

```
get_boot()
```

Returns

A dictionary containing boot information:

Parameter	**Description**
ztp	Zero Touch Provisioning (ZTP) status (string). Valid values: `“Enable”`, `”Forcedly Enabled”`, `”Forcedly Disabled”`. Default value: `“Enable”`.
active image	Active image information. A string containing the version and time downloaded.
standby image	Standby image information. A string containing the version and time downloaded.
uboot image	Uboot image information. A string containing the version and time downloaded.
ONIE	ONIE image information. Valid values: `empty` or a string containing the version and time downloaded.
boot software	Boot image setting (string). Valid values: `“active”`, `”standby”`.
scheduled reboot	The next scheduled reboot. Valid values: `none` or a string containing the date and time of the next scheduled reboot.
port mode	The port mode (string). Default value: `“default mode”`.

get_boot_ztp()

Gets detailed Zero Touch Provisioning (ZTP) boot information.

Syntax

`get_boot_ztp``()`

Returns

A dictionary containing ZTP boot information:

Parameter	**Description**
ztp	Zero touch feature status (string). Valid values: `“Enable”`, `”Forcedly Enabled”`, `”Forcedly Disabled”`. Default value: `“Enable”`.

set_boot_ztp()

Sets detailed Zero Touch Provisioning (ZTP) boot information.

Syntax

`set_boot_ztp``(`*state*`)`

where:

Parameter	**Description**
state	Zero touch feature status (string). Valid values: `“Enable”`, `”Forcedly Enabled”`, `”Forcedly Disabled”`.

Returns

Boolean (`True` on success, otherwise `False`).

get_boot_image()

Gets boot image status.

Syntax

`get_boot_image``()`

Returns

A dictionary containing boot software status:

Parameter	**Description**
boot software	Boot image status (string). Valid values: `“active”`, `”standby”`.

set_boot_image()

Sets next boot image.

Syntax

`set_boot_image``(`*image*`)`

where:

Parameter	Description
image	Next boot image (string). Valid values: `“active”`, `”standby”`.

Returns

Boolean (`True` on success, otherwise `False`).

BootProfile Module

This module gets and sets switch boot profile properties.

To use this module, in the Python file or in the Python interpreter, enter:

`import bootProfileApi`

class BootProfile()

This class provides functions to manage boot profile properties.

get_boot_profile()

Gets detailed boot profile information.

Syntax

`get_boot_profile``(`*index, detailed*`)`

where:

Parameter	Description
index	(Optional) The boot profile ID (integer or string). Valid values: • `0` - for all profiles • `1-N`- for a single profile • `"current"` - for the running profile • `"configured"` - for the configured profile for next unit boot Default value: `0`.
detailed	(Optional) Get detailed information about the boot profiles (boolean). Valid values: `True`, `False`. Default value: `False`.

Returns

A dictionary containing boot profile information:

Parameter	Description
cur_mode	The ID of the running boot profile (string). Valid values: 1-N.
conf_mode	The ID of the running boot profile (string). Valid values: 1-N.
mac_addr	The number of available MAC addresses (integer).
ipv4_uni	The number of IPv4 unicast addresses (integer).
ipv6_uni	The number of IPv6 unicast addresses (integer).

Parameter	Description
ipv4_multi	The number of IPv4 multicast addresses (integer).
ipv6_multi	The number of IPv6 multicast addresses (integer).
ipv4	The number of IPv4 unicast routes (integer).
ipv6	The number of IPv6 unicast routes (integer).

set_boot_profile()

Sets the boot profile that will be used after the next unit boot.

Syntax

`set_boot_profile``(`*index*`)`

where:

Parameter	Description
index	The boot profile ID (integer). Valid values: 1-N.

Returns

Boolean (`True` on success, otherwise `False`).

CEE Module

This module manages the Converged Enhanced Ethernet (CEE) configuration on the switch.

To use this module, in the Python file or in the Python interpreter, enter:

`import dcbApi`

class DCB()

This class provides functions to get and set Data Center Bridging (DCB) configurations.

python_dcbx_get_interface_state()

Displays the Data Center Bridging Capability Exchange protocol (DCBX) configuration for a specified switch interface.

Syntax

`python_dcbx_get_interface_state``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The name of the switch interface (string). For example: “`Ethernet1/12`”.

Returns

A dictionary showing DCBX interface information:

Parameter	**Description**
dcbx state	The status of DCBX on the interface (string). Valid values: “`enable`”, “`disable`”.
if_name	The name of the switch interface (string). For example: “`Ethernet1/12`”.
pfc advt	The status of Priority Flow Control (PFC) local configuration advertisement to the DCBX peer (string). Valid values: “`on`”, ”`off`”.
est advt	The status of Enhanced Transmission Selection (ETS) local configuration advertisement to the DCBX peer (string). Valid values: “`on`”, ”`off`”.
app advt	The status of application protocol local configuration advertisement to the DCBX peer (string). Valid values: “`on`”, ”`off`”.

python_dcbx_set_state()

Configures DCBX on a switch interface.

Syntax

`python_dcbx_set_state``(`*if_name, enadis_string*`)`

where:

Parameter	**Description**
if_name	The name of the switch interface (string). For example: `“Ethernet1/12”`.
enadis_string	The status of the DCBX (string). Valid values: `“enable”`, `“disable”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_dcbx_pfc_set_advt()

Configures PFC local configuration advertisement on a switch interface.

Syntax

`python_dcbx_pfc_set_advt``(`*if_name, enadis_string*`)`

where:

Parameter	**Description**
if_name	The name of the switch interface (string). For example: `“Ethernet1/12”`.
enadis_string	The status of the PFC local configuration advertisement (string). Valid values: `“on”`, `”off”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_dcbx_ets_set_advt()

Configures ETS local configuration advertisement on a switch interface.

Syntax

`python_dcbx_ets_set_advt``(`*if_name, enadis_string*`)`

where:

Parameter	Description
if_name	The name of the switch interface (string). For example: `“Ethernet1/12”`.
enadis_string	The status of the ETS local configuration advertisement (string). Valid values: `“on”`, `”off”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_dcbx_app_set_advt()

Configures application protocol local configuration advertisement on a switch interface.

Syntax

`python_dcbx_app_set_advt``(`*if_name, enadis_string*`)`

where:

Parameter	Description
if_name	The name of the switch interface (string). For example: `“Ethernet1/12”`.
enadis_string	The status of the application protocol local configuration advertisement (string). Valid values: `“on”`, `”off”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_dcbx_get_ctrl()

Displays the DCBX control state machine for a switch interface.

Syntax

`python_dcbx_get_ctrl`(*if_name*)

where:

Parameter	Description
if_name	The name of the switch interface (string). For example: “`Ethernet1/12`”.

Returns

A dictionary showing DCBX control state machine interface information:

Parameter	Description
dcbx version	The version of DCBX (string). Valid values: “`DCBX IEEE 802.1Qaz (v2.5)`” or ”`DCBX CEE (v1.01)`”.
if_name	The name of the switch interface (string). For example: “`Ethernet1/12`”.

python_dcbx_get_dcbxstate()

Displays the status of DCBX for a switch interface.

Syntax

`python_dcbx_get_dcbxstate`(*if_name*)

where:

Parameter	Description
if_name	The name of the switch interface (string). For example: “`Ethernet1/12`”.

Returns

Boolean (`True` on success, otherwise `False`).

python_dcbx_pfc_get_interface()

Displays the PFC configuration for a switch interface.

Syntax

`python_dcbx_pfc_get_interface``(`*if_name, cfgtype*`)`

where:

Parameter	**Description**
if_name	The name of the switch interface (string). For example: “`Ethernet1/12`”.
cfgtype	The type of PFC configuration (string). Valid values: “`admin`”, ”`operation`”, ”`remote`”.

Returns

A dictionary showing PFC interface configuration:

Parameter	**Description**
state	The status of PFC for the interface (string). Valid values: “`on`”, ”`off`”.
willing	Whether the switch is “willing” to learn PFC configurations from a DCBX peer (string). Valid values: “`on`”, ”`off`”.
advt	The status of PFC local configuration advertisement (string). Valid values: “`on`”, ”`off`”.
syncd	The status of PFC information synchronization (string). Valid values: “`on`”, ”`off`”.
priority_map	The priorities enabled on the interface (string).

python_dcbx_ets_get_interface()

Displays the ETS configuration for a switch interface.

Syntax

`python_dcbx_ets_get_interface``(`*if_name, cfgtype*`)`

where:

Parameter	Description
if_name	The name of the switch interface (string). For example: `“Ethernet1/12”`.
cfgtype	The type of PFC configuration (string). Valid values: `“admin”, ”operation”, ”remote”`.

Returns

A dictionary showing PFC interface configuration:

Parameter	Description
state	The status of ETS for the interface (string). Valid values: `“on”, ”off”`.
advt	Whether the switch is “willing” to learn ETS configurations from a DCBX peer (string). Valid values: `“on”, ”off”`.
willing	The status of ETS local configuration advertisement (string). Valid values: `“on”, ”off”`.
syncd	The status of ETS information synchronization (string). Valid values: `“on”, ”off”`.
tcg	The Traffic Class Group configuration. A list of priority groups, bandwidth percentage allocations, and assigned priorities
bandwidth	The bandwidth percentage allocated to a priority group. An integer from 1-100.
pgid	The ID of the priority group. An integer from 0-7 or 15.
priority	The priorities enabled for a priority group (string).

python_dcbx_app_get_interface()

Displays the application protocol configuration for a switch interface.

Syntax

`python_dcbx_app_get_interface``(`*if_name, crgtype*`)`

where:

Parameter	Description
if_name	The name of the switch interface (string). For example: `“Ethernet1/12”`.
cfgtype	The type of application protocol configuration (string). Valid values: `“admin”`, `”operation”`, `”remote”`.

Returns

A dictionary showing application protocol interface configuration:

Parameter	Description
state	The status of application protocol for the interface (string). Valid values: `“on”`, `”off”`.
willing	Whether the switch is “willing” to learn application protocol configurations from a DCBX peer (string). Valid values: `“on”`, `”off”`.
advt	The status of application protocol local configuration advertisement (string). Valid values: `“on”`, `”off”`.

python_dcbx_app_get_protocol_list_interface()

Displays the application protocol list for a switch interface.

Syntax

`python_dcbx_app_get_protocol_list_interface``(`*if_name, cfgtype*`)`

where:

Parameter	**Description**
if_name	The name of the switch interface (string). For example: `“Ethernet1/12”`.
cfgtype	The type of application protocol configuration (string). Valid values: `“admin”`, `”operation”`, `”remote”`.

Returns

A dictionary showing the application protocol list for the specified interface:

Parameter	**Description**
priority	The priority enabled for the protocol. An integer from 0-7.
protocol	The type of the protocol (string). Valid values: `“ethertype”`, `”udp”`, `”tcp”`.
protostr	The name of the protocol (string).

class CEE*()*

The functions in this class set and get CEE configurations.

python_cee_get_status()

Displays the CEE configuration of the switch.

Syntax

`python_cee_get_status``()`

Returns

A dictionary showing the CEE configuration:

Parameter	**Description**
status	The status of CEE on the switch (string). Valid values: `“on”`, `”off”`.

python_cee_set_status()

Globally enables or disables CEE on the switch.

Syntax

`python_cee_set_status``(`*cee_cfg*`)`

where:

Parameter	Description
cee_cfg	The status of CEE on the switch (string). Valid values: `“on”`, `”off”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_cee_is_platform_supported()

Gets the CEE Platform support status.

Syntax

`python_cee_is_platform_supported``()`

Returns

`Boolean` (`True` if the platform supports CEE, otherwise `False`).

class PFC*()*

This class provides functions to get and set PFC configurations.

python_cee_pfc_get_state()

Displays the status of PFC on the switch.

Syntax

`python_cee_pfc_get_state``()`

Returns

The status of PFC on the switch (string). Valid values: `“on”`, `”off”`.

python_cee_pfc_set_state()

Globally enables or disables PFC on the switch.

Syntax

`python_cee_pfc_set_state``(`*pfc_state*`)`

where:

Parameter	Description
pfc_state	The status of PFC (string). Valid values: `“on”`, `”off”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_cee_pfc_get_priority_map()

Displays the list of configured PFC priorities.

Syntax

`python_cee_pfc_get_priority_map``()`

Returns

A string showing the enabled list of PFC priorities.

python_cee_pfc_set_priority_map()

Configures the PFC priority flow mapping.

Note: This function overwrites existing PFC priority configurations.

Syntax

`python_cee_pfc_set_priority_map``(`*new_priority_map*`)`

where:

Parameter	Description
new_priority_map	The PFC priority flow map. A list of enabled priorities, which are integers between 0 and 7.

Returns

Boolean (`True` on success, otherwise `False`).

python_cee_pfc_interface_get_state()

Displays the status of PFC for a switch interface.

Syntax

`python_cee_pfc_interface_get_state`(*if_name*)

where:

Parameter	Description
if_name	The name of the switch interface (string). For example: "`Ethernet1/12`".

Returns

The status of PFC on the interface (string). Valid values: "`on`", "`off`".

python_cee_pfc_interface_set_state()

Enables or disables PFC on a switch interface.

Syntax

`python_cee_pfc_interface_set_state`(*if_name, pfc_state*)

where:

Parameter	Description
if_name	The name of the switch interface (string). For example: "`Ethernet1/12`".
pfc_state	The status of PFC on the specified interface (string). Valid values: "`on`", "`off`".

Returns

Boolean (`True` on success, otherwise `False`).

python_cee_pfc_interface_get_counters()

Displays the PFC statistics for a switch interface.

Syntax

`python_cee_pfc_interface_get_counters``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The name of the switch interface (string). For example: "`Ethernet1/12`".

Returns

A dictionary showing PFC interface statistics:

Parameter	**Description**
pfc_received	The number of received PFC packets (integer).
pfc_sent	The number of sent PFC packets (integer).
if_name	The name of the switch interface (string). For example: "`Ethernet1/12`".

class ETS()

This class provides functions to get and set ETS configurations.

python_cee_ets_get_config()

Displays the ETS configuration on the switch.

Syntax

`python_cee_ets_get_config``()`

Returns

A dictionary showing the ETS configuration:

Parameter	**Description**
bandwidth	The bandwidth percentage allocated to a priority group. An integer from 0-100.
pgid	The ID of the priority group. An integer from 0-7, or 15.
priority_pgid_mapping	The priorities mapped to the priority group. A list of priorities, which are integers between 0 and 7.

python_cee_ets_set_config()

Configures ETS.

Syntax

`python_cee_ets_set_config``(`*tcg_list*`)`

where *tcg_list* is a dictionary that must contain all eight priority groups (0-7 and 15) and the following variables:

Parameter	**Description**
bandwidth	The bandwidth percentage allocated to a priority group. An integer from 0-7, or 15.
pgid	The ID of the priority group. An integer from 0-7, or 15.
priority_pgid_mapping	The priorities mapped to the priority group. A list of priorities, which are integers between 0 and 7.

Returns

Boolean (`True` on success, otherwise `False`).

class APP()

This class provides functions to get and set application protocol configurations.

python_cee_app_get_protocol_list()

Displays the application protocol configuration.

Syntax

`python_cee_app_get_protocol_list``()`

Returns

A dictionary showing the application protocol configuration:

Parameter	**Description**
priority	The priority enabled for the protocol. An integer from 0-7.
protocol	The type of the protocol (string);. Valid values: `“ethertype”`, `“udp”`, `”tcp”`.
protoid	The name of the protocol (string).
config_name	The name of the application protocol configuration (string).

python_cee_app_post_protocol()

Creates an application protocol configuration.

Syntax

`python_cee_app_post_protocol``(`*priority, protoid, config_name, protocol=None*`)`

where:

Parameter	**Description**
priority	The priority enabled for the protocol. An integer from 0-7.
protoid	The name of the protocol (string).
config_name	The name of the application protocol configuration (string).
protocol	The type of the protocol (string). Valid values: `“ethertype”`, `“udp”`, `”tcp”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_cee_app_del_protocol()

Deletes an application protocol configuration.

Syntax

`python_cee_app_del_protocol``(`*config_name*`)`

where:

Parameter	**Description**
config_name	The name of the application protocol configuration (string).

Returns

Boolean (`True` on success, otherwise `False`).

CLI Module

This module manages Command Line Interface (CLI).

To use this module, in the Python file or in the Python interpreter, enter:

`import cliApi`

class Pylib()

This class provides APIs used to execute CLI commands on a switch.

getcmds()

Gets supported command list.

Syntax

`getcmds``()`

Returns

The list of supported CLI commands.

Note: While in CNOS CLI the vertical bar is used as an output modifier, in the returned command list it is used to match a single regular expression out of several possible regular expressions. To distinguish from the two, use an escape character. For example: `show running-config \| include hostname` translates in CNOS CLI as `show running-config | include hostname`.

runcmds()

Runs a list of commands.

Syntax

`runcmds``(`*cmds*`)`

where:

Parameter	**Description**
cmds	A list containing the supported CLI commands as string.

Returns

Multiple possible return values:

- A dictionary of command output on success
- The error description string: “`Argument type invalid`” or “`Command not supported: CMD_STRING`”.

DefaultIpAddress Module

This module manages Default IP Address.

To use this module, in the Python file or in the Python interpreter, enter:

```
import defaultIpAddressApi
```

class DefaultIpAddress()

This class provides functions to set/unset the default IP address on the management interface.

set_default_ip_address()

Sets the default IP address on the management interface.

Syntax

```
set_default_ip_address()
```

Returns

Boolean (`True` on success, otherwise `False`).

unset_default_ip_address()

Unsets the default IP address on the management interface.

Syntax

```
unset_default_ip_address()
```

Returns

Boolean (`True` on success, otherwise `False`).

get_default_ip_address()

Gets the default IP address status on the management interface.

Syntax

`get_default_ip_address``()`

Returns

A dictionary containing IP address status details:

Parameter	**Description**
interface	The management interface (string). Valid value: `”mgmt0”`.
state	The state of the default IP address feature on the management interface (string). Valid values: `“unset”, ”set”, ”installed”`.

DHCP Module

The module manages Dynamic Host Configuration Protocol (DHCP) information.

To use this module, in the Python file or in the Python interpreter, enter:

```
import dhcpApi
```

class DHCP_Client()

This class provides functions to get and set DHCP configurations.

get_dhcp_feature()

Determines whether the DHCP client is enabled or disabled.

Syntax

```
get_dhcp_feature()
```

Returns

Whether the DHCP client is enabled or disabled:

Parameter	**Description**
ena_dhcp_feature	Whether the DHCP client is enabled (string). Valid values: "`yes`", "`no`". Default value: "`yes`".

set_dhcp_feature()

Enables the DHCP client.

Syntax

```
set_dhcp_feature()
```

Returns

Boolean (`True` on success, otherwise `False`).

unset_dhcp_feature()

Disables the DHCP client.

Syntax

```
unset_dhcp_feature()
```

Returns

Boolean (`True` on success, otherwise `False`).

get_dhcpinfo()

Gets the DHCP client configuration.

Syntax

`get_dhcpinfo``(`*if_name*`)`

where:

Parameter	**Description**
if_name	(Optional) The interface port name (string).

Returns

One or more lists containing the DHCP configurations for all interfaces or for the specified interface:

Parameter	**Description**
if_name	Interface name (string).
ena_v4_client	Whether or not the DHCPv4 client is enabled on the interface. Valid values: `“yes”`, `”no”`. Default value: `”no”`.
ena_v6_client	Whether or not the DHCPv6 client is enabled on the interface. Valid values: `“yes”`, `”no”`. Default value: `”no”`.
req_hostname	Whether or not the request for host name option is enabled on an interface.
req_ntp_server	Whether or not the request for ntp-server option is enabled on the interface.
req_log_server	Whether or not the request for Log server option is enabled on the interface. Valid values: `“yes”`, `”no”`. Default value: `”no”`.
class_id	The name of the vendor class-identifier. A string up to 64 characters long.

set_dhcp_client()

Enables the DHCP client on the specified interface.

Syntax

`set_dhcp_client``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The interface port name (string).

Returns

Boolean (`True` on success, otherwise `False`).

unset_dhcp_client()

Disables the DHCP client on the specified interface.

Syntax

`unset_dhcp_client``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The interface port name (string).

Returns

Boolean (`True` on success, otherwise `False`).

set_dhcpv6_client()

Enables the DHCPv6 client on the specified interface.

Syntax

`set_dhcpv6_client``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The interface port name (string).

Returns

Boolean (`True` on success, otherwise `False`).

unset_dhcpv6_client()

Disables the DHCPv6 client on the specified interface.

Syntax

`unset_dhcpv6_client``(`*if_name*`)`

where:

Parameter	Description
if_name	The interface port name (string).

Returns

Boolean (`True` on success, otherwise `False`).

set_dhcp_option_hostname()

Enables the DHCP hostname option on the specified interface.

Syntax

`set_dhcp_option_hostname``(`*if_name*`)`

where:

Parameter	Description
if_name	The interface port name (string).

Returns

Boolean (`True` on success, otherwise `False`).

unset_dhcp_option_hostname()

Disables the DHCP hostname option on the specified interface.

Syntax

`unset_dhcp_option_hostname``(`*if_name*`)`

where:

Parameter	Description
if_name	The interface port name (string).

Returns

Boolean (`True` on success, otherwise `False`).

set_dhcp_option_ntp_server()

Enables the NTP server option on the DHCP client of the specified interface.

Syntax

`set_dhcp_option_ntp_server``(`*if_name*`)`

where:

Parameter	Description
if_name	The interface port name (string).

Returns

Boolean (`True` on success, otherwise `False`).

unset_dhcp_option_ntp_server()

Disables the NTP server option on the DHCP client of the specified interface.

Syntax

`unset_dhcp_option_ntp_server``(`*if_name*`)`

where:

Parameter	Description
if_name	The interface port name (string).

Returns

Boolean (`True` on success, otherwise `False`).

set_dhcp_option_log_server()

Enables the log server on the DHCP client on the specified interface.

Syntax

`set_dhcp_option_log_server``(`*if_name*`)`

where:

Parameter	Description
if_name	The interface port name (string).

Returns

Boolean (`True` on success, otherwise `False`).

unset_dhcp_option_log_server()

Disables the log server on the DHCP client on the specified interface.

Syntax

`unset_dhcp_option_log_server``(`*if_name*`)`

where:

Parameter	Description
if_name	The interface port name (string).

Returns

Boolean (`True` on success, otherwise `False`).

set_dhcp_class_id()

Sets the specified vendor class ID on the specified interface.

Syntax

`set_dhcp_class_id``(`*if_name, class_id*`)`

where:

Parameter	Description
if_name	The interface port name (string).
class_id	The vendor class ID. A string up to 64 characters long.

Returns

Boolean (`True` on success, otherwise `False`).

unset_dhcp_class_id()

Removes the vendor class ID from the specified interface.

Syntax

`unset_dhcp_class_id``(`*if_name*`)`

where:

Parameter	Description
if_name	The interface port name (string).

Returns

Boolean (`True` on success, otherwise `False`).

class DHCP_Relay()

This class provides functions to get and set DHCP relay configurations.

get_relay_serv()

Determines whether DHCP relay is enabled or disabled for DHCPv4 and DHCPv6.

Syntax

```
get_relay_serv()
```

Returns

A dictionary showing whether DHCP relay is enabled or disabled:

Parameter	**Description**
ena_v4_relay	Whether DHCPv4 relay is enabled (string). Valid values: `“yes”`, `”no”`. Default value: `”no”`.
ena_v6_relay	Whether DHCPv6 relay is enabled (string). Valid values: `“yes”`, `”no”`. Default value: `”no”`.

set_dhcpv4_relay()

Enables DHCPv4 relay service.

Syntax

```
set_dhcpv4_relay()
```

Returns

Boolean (`True` on success, otherwise `False`).

unset_dhcpv4_relay()

Disables DHCPv4 relay service.

Syntax

```
unset_dhcpv4_relay()
```

Returns

Boolean (`True` on success, otherwise `False`).

set_dhcpv6_relay()

Enables DHCPv6 relay service.

Syntax

`set_dhcpv6_relay``()`

Returns

Boolean (`True` on success, otherwise `False`).

unset_dhcpv6_relay()

Disables DHCPv6 relay service.

Syntax

`unset_dhcpv6_relay``()`

Returns

Boolean (`True` on success, otherwise `False`).

get_interface_relay_address()

Gets all DHCPv4 and DHCPv6 relay addresses for all interfaces or for the specified interface.

Syntax

`get_interface_relay_address``(`*if_name*`)`

where:

Parameter	**Description**
if_name	(Optional) The interface port name (string).

Returns

A dictionary containing all DHCPv4 and DHCPv6 relay addresses for all interfaces or for the specified interface:

Parameter	**Description**
if_name	(Optional) The interface port name (string).
dhcpv4_relay	A dictionary containing DHCPv4 relay addresses.
v4_relay_address	DHCPv4 relay server address (string). A valid IPv4 address.
dhcpv6_relay	A dictionary containing DHCPv6 relay addresses.

Parameter	Description
v6_relay_address	DHCPv6 relay server address (string). A valid IPv6 address.
v6_relay_out_if	DHCPv6 outgoing interface (string).

set_relay4_address()

Sets a DHCPv4 relay address for the specified interface.

Syntax

`set_relay4_address``(`*if_name, ip_addr*`)`

where:

Parameter	Description
if_name	The interface port name (string).
ip_addr	The IP address (string). A valid IPv4 address.

Returns

Boolean (`True` on success, otherwise `False`).

unset_relay4_address()

Removes a DHCPv4 relay address from the specified interface.

Syntax

`unset_relay4_address``(`*if_name, ip_addr*`)`

where:

Parameter	Description
if_name	The interface port name (string).
ip_addr	The IP address (string). A valid IPv4 address.

Returns

Boolean (`True` on success, otherwise `False`).

set_relay6_address()

Sets a DHCPv6 relay address and relay outgoing interface for the specified interface.

Syntax

`set_relay6_address(`*if_name, ipv6_addr, out_if_name***`)`**

where:

Parameter	**Description**
if_name	The interface port name (string).
ipv6_addr	The IP address (string). A valid IPv6 address.
out_if_name	(Optional) Relay outgoing interface (string).

Returns

Boolean (`True` on success, otherwise `False`).

unset_relay6_address()

Removes a DHCPv6 relay address and relay outgoing interface from the specified interface.

Syntax

`unset_relay6_address(`*if_name, ipv6_addr***`)`**

where:

Parameter	**Description**
if_name	The interface port name (string).
ipv6_addr	The IP address (String). A valid IPv6 address.
out_if_name	(Optional) Relay outgoing interface (string).

Returns

Boolean (`True` on success, otherwise `False`).

class DHCP_Snooping()

This class provides functions to set and get DHCP Snooping configurations.

python_dhcpsnp_get_status_opt82()

Checks if DHCP Snooping and DHCP Option 82 are enabled on the switch.

Syntax

`python_dhcpsnp_get_status_opt82``()`

Returns

Returns a dictionary showing whether DHCP Snooping and DHCP Option 82 are enabled or disabled:

Parameter	**Description**
dhcpsnp_feature	The status of DHCP Snooping (string). Valid values: `“enable”`, `”disable”`.
option_82	The status of DHCP Option 82 (string). Valid values: `“enable”`, `”disable”`.

python_dhcpsnp_put_status_opt82()

Enables or disables DHCP Snooping and DHCP Option 82 on the switch.

Syntax

`python_dhcpsnp_put_status_opt82``(`*dhcpsnp_feature, option_82*`)`

where:

Parameter	**Description**
dhcpsnp_feature	The status of DHCP Snooping (string). Valid values: `“enable”`, `”disable”`.
option_82	The status of DHCP Option 82 (string). Valid values: `“enable”`, `”disable”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_dhcpsnp_get_binding_entry()

Displays the DHCP Snooping binding table entries.

Syntax

```
python_dhcpsnp_get_binding_entry()
```

Returns

A list of dictionaries with DHCP Snooping binding table entry information:

Parameter	Description
mac	The MAC address of the binding entry. A string in the following format: "`XX:XX:XX:XX:XX:XX`".
ip_addr	The IP address of the binding entry (string).
lease_time	The lease time, in seconds, of the binding entry. An integer from 1-4294967295.
type	The type of the binding entry (string). Valid values: "`static`", "`dynamic`".
vlan	The VLAN ID of the binding entry. An integer from 1-4093.
if_name	The name of the switch interface associated with the binding entry (string). For example: "`Ethernet1/12`".

python_dhcpsnp_post_binding_entry()

Adds entries to the DHCP Snooping binding table.

Syntax

`python_dhcpsnp_post_binding_entry``(`*dict_dhcpsnp_entry*`)`

where *dict_dhcpsnp_entry* is a dictionary containing the following variables:

Parameter	**Description**
mac	The IP address of the binding entry (string).
ip_addr	The lease time, in seconds, of the binding entry. An integer from 1-4294967295.
lease_time	The type of the binding entry (string). Valid values: `“static”`,`”dynamic”`.
vlan	The VLAN ID of the binding entry. An integer from 1-4093.
if_name	The name of the switch interface associated with the binding entry (string). For example: `“Ethernet1/12”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_dhcpsnp_del_binding_entry()

Deletes DHCP Snooping binding table entries.

Syntax

`python_dhcpsnp_del_binding_entry``(`*mac_vlan_if*`)`

where:

Parameter	**Description**
mac_vlan_if	(Optional) The binding entry identified by either its MAC address, VLAN ID, or interface name. Valid values: ● for MAC address - a string in the following format: `“XX.XX.XX.XX.XX.XX”` ● for VLAN ID - an integer from 1-4093 ● for interface name - a string (for example, `“Ethernet1/12”`)

Returns

Boolean (`True` on success, otherwise `False`).

python_dhcpsnp_get_vlan()

Displays the VLANs for which DHCP Snooping is configured.

Syntax

`python_dhcpsnp_get_vlan``()`

Returns

A dictionary showing the VLANs for which DHCP Snooping is configured:

Parameter	**Description**
vlan_enabled	The VLAN IDs for which DHCP Snooping is configured (string).

python_dhcpsnp_put_vlan()

Configures DHCP Snooping on the specified VLAN.

Syntax

`python_dhcpsnp_put_vlan``(`*vlan_enabled*`)`

where:

Parameter	**Description**
vlan_enabled	The VLANs for which DHCP Snooping is configured (string).

Returns

Boolean (`True` on success, otherwise `False`).

python_dhcpsnp_del_vlan()

Disables DHCP Snooping on the specified VLAN.

Syntax

python_dhcpsnp_del_vlan(*vlan_id*)

where:

Parameter	Description
vlan_id	The VLAN ID. An integer from 1-4093.

Returns

Boolean (`True` on success, otherwise `False`).

python_dhcpsnp_get_statistics()

Displays DHCP Snooping statistics.

Syntax

python_dhcpsnp_get_statistics()

Returns

A dictionary showing DHCP Snooping statistics:

Parameter	Description
rcv_req_pkts	The number of received request packets (integer).
rcv_rep_pkts	The number of received reply packets (integer).
drop_pkts	The number of dropped packets (integer).

python_dhcpsnp_clear_statistics()

Deletes all DHCP Snooping statistics.

Syntax

python_dhcpsnp_clear_statistics()

Returns

Boolean (`True` on success, otherwise `False`).

python_dhcpsnp_get_trust_port()

Displays DHCP Snooping trusted interfaces.

Syntax

`python_dhcpsnp_get_trust_port``()`

Returns

A dictionary with DHCP Snooping trusted interface information:

Parameter	**Description**
if_name	The name of the DHCP Snooping trusted interface (string). For example: “`Ethernet1/12`”.
trusted	Whether the interface is trusted (string). Valid values: “`yes`”, “`no`”.

python_dhcpsnp_set_trust_port()

Configures DHCP Snooping trusted interfaces.

Syntax

`python_dhcpsnp_set_trust_port``(`*if_name, trusted*`)`

where:

Parameter	**Description**
if_name	The name of the switch interface (string). For example: “`Ethernet1/12`”.
trusted	Whether the interface is trusted (string). Valid values: “`yes`”, “`no`”.

Returns

Boolean (`True` on success, otherwise `False`).

DNS Module

This module manages the Domain Name Servers (DNS) configuration on the switch.

To use this module, in the Python file or in the Python interpreter, enter:

```
import hostpDnsClntApi
```

class DNS()

This class provides functions to get and set DNS configurations.

enable_dns_domain_lookup()

Enables DNS on the switch.

Syntax

```
enable_dns_domain_lookup()
```

Returns

Boolean (`True` on success, otherwise `False`).

disable_dns_domain_lookup()

Disables DNS on the switch.

Syntax

```
disable_dns_domain_lookup()
```

Returns

Boolean (`True` on success, otherwise `False`).

add_dns_name_server()

Configures a single or multiple DNS servers on the switch.

Syntax

`add_dns_name_server``(`*vrf, nameserver1, nameserver2, nameserver3*`)`

where:

Parameter	Description
vrf	The VRF instance for the DNS server (string).
nameserver1	The name of the first DNS server (string).
nameserver2	(Optional) The name of the second DNS server (string).
nameserver3	(Optional) The name of the third DNS server (string).

Returns

Boolean (`True` on success, otherwise `False`).

del_dns_name_server()

Deletes a single or multiple already configured DNS servers.

Syntax

`del_dns_name_server``(`*vrf, nameserver1, nameserver2, nameserver3*`)`

where:

Parameter	Description
vrf	The VRF instance for the DNS server (string).
nameserver1	The name of the first DNS server (string).
nameserver2	(Optional) The name of the second DNS server (string).
nameserver3	(Optional) The name of the third DNS server (string).

Returns

Boolean (`True` on success, otherwise `False`).

add_default_domain()

Configures a default domain name.

Syntax

`add_default_domain``(`*domain_name, vrf*`)`

where:

Parameter	Description
domain_name	The default domain name (string).
vrf	(Optional) The VRF instance for the domain name (string).

Returns

Boolean (`True` on success, otherwise `False`).

del_default_domain()

Deletes an already configured default domain name.

Syntax

`del_default_domain``(`*domain_name, vrf*`)`

where:

Parameter	Description
domain_name	The default domain name (string).
vrf	(Optional) The VRF instance for the domain name (string).

Returns

Boolean (`True` on success, otherwise `False`).

add_name_to_ip()

Assigns a hostname to an IP address.

Syntax

`add_name_to_ip``(`*vrf, hostname, ip_addr1, ip_addr2*`)`

where:

Parameter	**Description**
vrf	The VRF instance for the hostname (string).
hostname	The hostname to be assigned to the specified IP address (string).
ip_addr1	The IP address to be assigned to the specified hostname (string).
ip_addr2	(Optional) A second IP address to be assigned to the specified hostname (string).

Returns

Boolean (`True` on success, otherwise `False`).

del_name_to_ip()

Deletes a hostname/IP address association.

Syntax

`del_name_to_ip``(`*vrf, hostname, ip_addr1, ip_addr2*`)`

where:

Parameter	**Description**
vrf	The VRF instance for the hostname (string).
hostname	The hostname to delete (string).
ip_addr1	The IP address to delete (string).
ip_addr2	(Optional) A second IP address to delete (string).

Returns

Boolean (`True` on success, otherwise `False`).

add_domain_name()

Configures a domain name.

Syntax

`add_domain_name``(`*domain_name, vrf*`)`

where:

Parameter	Description
domain_name	The domain name (string).
vrf	The VRF instance for the domain name (string).

Returns

Boolean (`True` on success, otherwise `False`).

del_domain_name()

Deletes an already configured domain name.

Syntax

`del_domain_name``(`*domain_name, vrf*`)`

where:

Parameter	Description
domain_name	The domain name (string).
vrf	(Optional) The VRF instance for the domain name (string).

Returns

Boolean (`True` on success, otherwise `False`).

dns_show_domain_list_info()

Displays DNS domain information.

Syntax

`dns_show_domain_list_info``(`*vrf*`)`

where:

Parameter	**Description**
vrf	The VRF instance for the DNS domains (string).

Returns

A dictionary showing DNS domain information.

Parameter	**Description**
domain_lookup	The status of DNS on the switch (string). Valid values: `“enabled”,”disabled”`.
dynamic_domain	Dynamic domain information (string).
dynamic_nameserver	Dynamic DNS information (string).
domain_list	Domain information. A list of dictionaries containing string elements.
nameserver_list	DNS server information. A list of dictionaries containing string elements.
nametoip_list	Hostname/IP address associations. A list of dictionaries containing string elements (hostname to IP address mappings).

Dot1QEncaps Module

This modules manages Dot1Q encapsulation.

To use this module, in the Python file or in the Python interpreter, enter:

`import dot1qEncapsApi`

class Dot1qEncapsulation()

This class provides functions that manage Dot1Q encapsulation.

set_dot1q_tag_interface()

Enables Dot1Q tag on an interface.

Syntax

`set_dot1q_tag_interface``(`*if_name, tag*`)`

where:

Parameter	**Description**
if_name	The interface name (string);
tag	The Dot1Q tag value. An integer from 1-4093.

Returns

Boolean (`True` on success, otherwise `False`).

unset_dot1q_tag_interface()

Disables Dot1Q tag on an interface.

Syntax

`unset_dot1q_tag_interface``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The interface name (string);

Returns

Boolean (`True` on success, otherwise `False`).

get_dot1q_tag_interface()

Gets Dot1Q tag on an interface.

Syntax

`get_dot1q_tag_interface`(*if_name*)

where:

Parameter	**Description**
if_name	The interface name (string).

Returns

A dictionary containing the interface name and the configured tag value.

Dot1X Module

This modules manages 802.1X authentication.

To use this module, in the Python file or in the Python interpreter, enter:

```
import dot1xApi
```

class Dot1x()

This class provides functions that manage 802.1X authentication.

getLastErr()

Gets the last error message corresponding to the last unsuccessful call of the API.

Syntax

```
getLastErr()
```

Returns

A string containing the error or `None` if no errors were registered until that given moment.

enable()

Enables 802.1X globally.

Syntax

```
enable()
```

Returns

Boolean (`True` on success, otherwise `False`).

disable()

Disables 802.1X globally.

Syntax

```
disable()
```

Returns

Boolean (`True` on success, otherwise `False`).

passThroughEnable()

Enables 802.1X pass-through globally.

Syntax

`passThroughEnable``()`

Returns

Boolean (`True` on success, otherwise `False`).

passThroughDisable()

Disables 802.1X pass-through globally.

Syntax

`passThroughDisable``()`

Returns

Boolean (`True` on success, otherwise `False`).

setPortMode()

Sets the 802.1X port mode.

Syntax

`setPortMode``(`*mode, ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	**Description**
mode	The port mode (integer). Valid values: ● `0` - None ● `1` - Auto ● `2` - Force-Authorized ● `3` - Force-Unauthorized
ifType	The interface type (string). Valid values: `“ethernet”`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

Boolean (`True` on success, otherwise `False`).

setErrVlan()

Sets the 802.1X error VLAN value.

Syntax

`setErrVlan``(`*vlanType, vlanID, ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	**Description**
vlanType	The type of error VLAN to be set (integer). Valid values: ● `0` - guest ● `1` - unauthorized ● 2 - fallback
vlanID	The ID of the error VLAN to be set. An integer from 1-4094. **Note:** The *vlanID* must be `0` to disable the error VLAN.
ifType	The interface type (string). Valid values: `“ethernet”`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

Boolean (`True` on success, otherwise `False`).

setDynamicVlan()

Enables or disables the 802.1X dynamic VLAN.

Syntax

`setDynamicVlan``(`*enable, ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	**Description**
enable	Whether the dynamic VLAN option is enabled or disabled (boolean). Valid values: `True`, `False`.
ifType	The interface type (string). Valid values: `“ethernet”`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

Boolean (`True` on success, otherwise `False`).

setDynamicAcl()

Enables or disables the 802.1X dynamic ACL.

Syntax

`setDynamicAcl``(`*enable, ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	**Description**
enable	Whether the dynamic ACL option is enabled or disabled (boolean). Valid values: `True`, `False`.
ifType	The interface type (string). Valid values: `“ethernet”`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

Boolean (`True` on success, otherwise `False`).

setTimeout()

Enables or disables the 802.1X timeout value.

Syntax

`setTimeout(` *timeoutType, timeoutValue, ifType, chassisNumber, ifNumber, ifSubNumber* `)`

where:

Parameter	**Description**
timeoutType	Indicates the timer type (integer). Valid values: ● `0` - Quiet timeout ● `1` - Reauthentication timeout ● `2` - Supplicant timeout ● `3` - TX timeout
timeoutValue	The value set for the timer. An integer from 1-65535.
ifType	The interface type (string). Valid values: `“ethernet”`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

Boolean (`True` on success, otherwise `False`).

setReauth()

Enables or disables the 802.1X reauthentication.

Syntax

`setReauth``(`*enable, ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	Description
enable	Whether the dynamic ACL option is enabled or disabled (boolean). Valid values: `True`, `False`.
ifType	The interface type (string). Valid values: `“ethernet”`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

Boolean (`True` on success, otherwise `False`).

setNoMacTableBinding()

Enables or disables the 802.1X no-mac-table-binding.

Syntax

`setNoMacTableBinding``(`*enable, ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	Description
enable	Whether the no-mac-table-binding option is enabled or disabled (boolean). Valid values: `True`, `False`.
ifType	The interface type (string). Valid values: `“ethernet”`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

Boolean (`True` on success, otherwise `False`).

setMAB()

Enables or disables the 802.1X MAC Authentication Bypass (MAB).

Syntax

`setMAB``(`*enable, ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	**Description**
enable	Whether MAB is enabled or disabled (boolean). Valid values: `True`, `False`.
ifType	The interface type (string). Valid values: `"ethernet"`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

Boolean (`True` on success, otherwise `False`).

reinit()

Reinitializes the 802.1X controlled port.

Syntax

`reinit``(`*ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	**Description**
ifType	The interface type (string). Valid values: `"ethernet"`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

Boolean (`True` on success, otherwise `False`).

setHostMode()

Sets the 802.1X host mode.

Syntax

`setHostMode(`*hostModeType, ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	**Description**
hostModeType	The host mode type (integer). Valid values: ● 0 - Single-host ● 1 - Multi-host ● 2 - Multi-authentication
ifType	The interface type (string). Valid values: `“ethernet”`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

Boolean (`True` on success, otherwise `False`).

setAccounting()

Sets the 802.1X accounting.

Syntax

`setAccounting(`*enable, ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	**Description**
enable	Whether the accounting is enabled or disabled (boolean). Valid values: `True`, `False`.
ifType	The interface type (string). Valid values: `“ethernet”`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

Boolean (`True` on success, otherwise `False`).

setMaxReq()

Sets the 802.1X number of EAP requests sent.

Syntax

`setMaxReq``(`*maxReqValue, ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	**Description**
maxReqValue	The number of requests sent. An integer from 1-10.
ifType	The interface type (string). Valid values: `“ethernet”`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

Boolean (`True` on success, otherwise `False`).

clearStatistics()

Clears the 802.1X statistics on a specified interface.

Syntax

`clearStatistics``(`*ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	**Description**
ifType	The interface type (string). Valid values: `“ethernet”`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

Boolean (`True` on success, otherwise `False`).

clearDiagnostics()

Clears the 802.1X diagnostics on a specified interface.

Syntax

`clearDiagnostics``(`*ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	**Description**
ifType	The interface type (string). Valid values: `“ethernet”`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

Boolean (`True` on success, otherwise `False`).

clearStatisticsAll()

Clears the 802.1X statistics on all interfaces.

Syntax

`clearStatisticsAll``()`

Returns

Boolean (`True` on success, otherwise `False`).

clearDiagnosticsAll()

Clears the 802.1X diagnostics on all interfaces.

Syntax

`clearDiagnosticsAll``()`

Returns

Boolean (`True` on success, otherwise `False`).

setMacMove()

Sets the 802.1X MAC move.

Syntax

`setMacMove(`*enable***`)`**

where:

Parameter	**Description**
enable	Whether the MAC move is enabled or disabled (boolean). Valid values: `True`, `False`. Default value: `False`.

Returns

Boolean (`True` on success, otherwise `False`).

show()

Gets the 802.1X information for a specified interface.

Syntax

`show(`*detail, ifType, chassisNumber, ifNumber, ifSubNumber***`)`**

where:

Parameter	**Description**
detail	Shows detailed 802.1X information for an interface (boolean). Valid values: `True`, `False`.
ifType	The interface type (string). Valid values: `“ethernet”`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

A list of dictionaries containing interface information:

Parameter	**Description**
interface	The interface name (string). Valid values: `“ethernet<x,y,z>”`.
connectedSupplicants	The number of connected supplicants (integer).

Parameter	Description
sessions	A dictionary containing the following information: ● `username`: The user name (string) ● `macAddress`: The MAC address (string) ● `vlan`: The VLAN ID (integer). Default value: `1` ● `paeState`: The PAE FSM state (string)
maxRequests	The number of maximum requests (integer).
guestVlan	Whether the guest VLAN option is enabled or not (boolean).
macAuthenticationBypass	Whether the MAC-authentication-bypass option is enabled or not (boolean). Valid values: `True`, `False`.
noMacTableBinding	Whether the no-mac-table-binding option is enabled or not (boolean). Valid values: `True`, `False`.
unauthorizedVlan	Whether the unauthorized VLAN option is enabled or not (boolean). Valid values: `True`, `False`.
hostMode	The host mode (string).
fallbackVlan	Whether the fallback VLAN option is enabled or not (boolean). Valid values: `True`, `False`.
accounting	Whether the accounting option is enabled or not (boolean). Valid values: `True`, `False`.
dynamicAcl	Indicates whether the dynamic ACL is enabled (string).
quietPeriod	The duration in seconds of the quiet period (integer).
reauthPeriod	The duration in seconds until trying to re-authenticate (integer).
pae	The PAE FSM role (string). It always returns “`authenticator`”.
dynamicVlan	Whether the dynamic VLAN option is enabled or not (boolean). Valid values: `True`, `False`.
reauthentication	Whether the reauthentication option is enabled or not (boolean). Valid values: `True`, `False`.
interface	The interface name (string).
reauthPeriod	The duration in seconds until trying to re-authenticate (integer).
authenticatedSupplicants	The number of authenticated supplicants (integer).
txPeriod	The duration is seconds between EAP Request Identity packets (integer).

Parameter	Description
suppTimeout	The duration is seconds until the communication with the supplicant transitions into a timeout phase. (integer).
portControl	The port control mode on this interface (string).

showAll()

Gets the 802.1X global configuration.

Syntax

`showAll(`*detail***`)`**

where:

Parameter	Description
detail	Shows detailed 802.1X information for an interface (boolean). Valid values: `True`, `False`.

Returns

A list of dictionaries containing interface information:

Parameter	Description
enable	Whether 802.1X is globally enabled or disabled (boolean). Valid values: `True`, `False`.
passThrough	Whether 802.1X pass-through is globally enabled (boolean). Valid values: `True`, `False`.
interface	The interface name (string). Valid values: `“ethernet<x,y,z>”`.
connectedSupplicants	The number of connected supplicants (integer).
sessions	A dictionary containing the following information: ● `username`: The user name (string) ● `macAddress`: The MAC address (string) ● `vlan`: The VLAN ID (integer). Default value: `1` ● `paeState`: The PAE FSM state (string)
maxRequests	The number of maximum requests (integer).
guestVlan	Whether the guest VLAN option is enabled or not (boolean).
macAuthenticationBypass	Whether the MAC-authentication-bypass option is enabled or not (boolean). Valid values: `True`, `False`. Default value: `False`.

Parameter	**Description**
noMacTableBinding	Whether the no-mac-table-binding option is enabled or not (boolean). Valid values: `True`, `False`.
unauthorizedVlan	Whether the unauthorized VLAN option is enabled or not (boolean). Valid values: `True`, `False`.
hostMode	The host mode (string).
fallbackVlan	Whether the fallback VLAN option is enabled or not (boolean). Valid values: `True`, `False`.
accounting	Whether the accounting option is enabled or not (boolean). Valid values: `True`, `False`.
dynamicAcl	Indicates whether the dynamic ACL is enabled (string).
quietPeriod	The duration in seconds of the quiet period (integer).
reauthPeriod	The duration in seconds until trying to re-authenticate (integer).
pae	The PAE FSM role (string). It always returns `“authenticator”`.
dynamicVlan	Whether the dynamic VLAN option is enabled or not (boolean). Valid values: `True`, `False`.
reauthentication	Whether the reauthentication option is enabled or not (boolean). Valid values: `True`, `False`.
interface	The interface name (string).
reauthPeriod	The duration in seconds until trying to re-authenticate (integer).
authenticatedSupplicants	The number of authenticated supplicants (integer).
txPeriod	The duration in seconds between EAP Request Identity packets (integer).
suppTimeout	The duration in seconds until the communication with the supplicant transitions into a timeout phase (integer).
portControl	The port control mode on this interface (string).

showDiagnostics()

Gets the 802.1X diagnostics for a specified interface.

Syntax

`showDiagnostics(`*ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	Description
ifType	The interface type (string). Valid values: `“ethernet”`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

A dictionary containing diagnostics information:

Parameter	Description
authAuthEapLogoffWhileAuthenticating	The number of authentication EAP logoff packets received while authenticating (integer).
authAuthEapStartsWhileAuthenticating	The number of authentication EAP start packets received while authenticating (integer).
authAuthFailWhileAuthenticating	The number of authentication fails while authenticating (integer).
authAuthSuccessesWhileAuthenticating	The number of authentication successes while authenticating (integer).
authEntersAuthenticating:	The number of times the authentication enters in authenticating state (integer).
authAuthTimeoutsWhileAuthenticating	The number of authentication timeouts while authenticating (integer).
interface	The interface name (string).
authAuthEapLogoffWhileAuthenticated	The number of authentication EAP logoff packets received while authenticated (integer).
authAuthReauthsWhileAuthenticated	The number of re-authentications triggered while authenticated (integer).

showDiagnosticsAll()

Gets the 802.1X diagnostics for all interfaces.

Syntax

```
showDiagnosticsAll()
```

Returns

A dictionary containing diagnostics information:

Parameter	Description
authAuthEapLogoffWhileAuthenticating	The number of authentication EAP logoff packets received while authenticating (integer).
authAuthEapStartsWhileAuthenticating	The number of authentication EAP start packets received while authenticating (integer).
authAuthFailWhileAuthenticating	The number of authentication fails while authenticating (integer).
authAuthSuccessesWhileAuthenticating	The number of authentication successes while authenticating (integer).
authEntersAuthenticating:	The number of times the authentication enters in authenticating state (integer).
authAuthTimeoutsWhileAuthenticating	The number of timeouts while authenticating (integer).
interface	The interface name (string).
authAuthEapLogoffWhileAuthenticated	The number of EAP logoff packets received while authenticated (integer).
authAuthReauthsWhileAuthenticated	The number of re-authentications triggered while authenticated (integer).

showPortStatistics()

Gets the 802.1X port statistics for an interface.

Syntax

`showPortStatistics``(`*ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	**Description**
ifType	The interface type (string). Valid values: `“ethernet”`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4.

Returns

A dictionary containing port statistics information:

Parameter	**Description**
eapolPortUnavailable	Holds the number of times a virtual port could not be created (integer).
lastEapolFrameSource	The source MAC address for the last EAPoL frame (string).
eapolLogoffFramesRx	The number of EAPoL frames received (integer).
eapLengthErrorFramesRx	The number of EAPoL frames with length error received (integer).
eapolAuthEapFramesTx	The number of auth EAPoL frames sent (integer).
invalidFramesEapolRx	The number of invalid EAPoL frames received (integer).
lastEapolFrameVersion	The version of the last EAPoL frame received (integer).
interface	The interface name (string).
eapolStartFramesRx	The number of EAPoL Start frames received (integer).
eapolEapFramesRx	The number of EAPoL EAP frames received (integer).

showPortStatisticsAll()

Gets the 802.1X port statistics for all interfaces.

Syntax

```
showPortStatisticsAll()
```

Returns

A dictionary containing port statistics information:

Parameter	**Description**
eapolPortUnavailable	Holds the number of times a virtual port could not be created (integer).
lastEapolFrameSource	The source MAC address for the last EAPoL frame (string).
eapolLogoffFramesRx	The number of EAPoL frames received (integer).
eapLengthErrorFramesRx	The number of EAPoL frames with length error received (integer).
eapolAuthEapFramesTx	The number of auth EAPoL frames sent (integer).
invalidFramesEapolRx	The number of invalid EAPoL frames received (integer).
lastEapolFrameVersion	The version of the last EAPoL frame received (integer).
interface	The interface name (string).
eapolStartFramesRx	The number of EAPoL Start frames received (integer).
eapolEapFramesRx	The number of EAPoL EAP frames received (integer).

showSessionStatistics()

Gets the 802.1X session statistics for a specified interface.

Syntax

`showSessionStatistics``(`*ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	**Description**
ifType	The interface type (string). Valid values: `“ethernet”`.
chassisNumber	The chassis number (integer). Valid values: `1`.
ifNumber	The interface number (integer).
ifSubNumber	(Optional) The interface sub-number. An integer from 1-4. Default value: `None`.

Returns

A dictionary containing the supplicant session information:

Parameter	**Description**
username	The supplicant user name (string).
inPackets	The number of packets received by the supplicant (integer).
outOctets	The number of octets sent by the supplicant (integer).
inOctets	The number of octets received by the supplicant. (integer).
outPackets	The number of packets sent by the supplicant. (integer).
sessionId	The session ID (string).
time	The seconds since the session has been active (integer).
interface	The interface name (string).
terminateCause	The cause for ending the session (string).

showSessionStatisticsAll()

Gets the 802.1X session statistics for all interfaces.

Syntax

```
showSessionStatisticsAll()
```

Returns

A dictionary containing the supplicant session information:

Parameter	**Description**
username	The supplicant user name (string).
inPackets	The number of packets received by the supplicant (integer).
outOctets	The number of octets sent by the supplicant (integer).
inOctets	The number of octets received by the supplicant (integer).
outPackets	The number of packets sent by the supplicant. (integer).
sessionId	The session ID (string).
time	The seconds since the session has been active (integer).
interface	The interface name (string).
terminateCause	The cause for ending the session (string).

ECMP Module

This module manages the Equal-Cost Multi-Path (ECMP) configuration on the switch.

To use this module, in the Python file or in the Python interpreter, enter:

```
import weightedEcmpApi
```

class WeightedEcmp()

This class provides functions that get and set weighted ECMP configurations.

get_weighted_ecmp_state()

Checks if weighted ECMP is enabled on the switch.

Syntax

`get_weighted_ecmp_state``()`

Returns

A dictionary showing the status of weighted ECMP on the switch:

Parameter	**Description**
weighted_ecmp_state	The status of weighted ECMP (string). Valid values: `“enable”`, `”disable”`.

set_weighted_ecmp_state()

Enables or disables weighted ECMP on the switch.

Syntax

`set_weighted_ecmp_state``(`*state*`)`

where:

Parameter	**Description**
state	The status weighted ECMP (string). Valid values: `“enable”`, `”disable”`.

Returns

Boolean (`True` on success, otherwise `False`).

class WeightedEcmp_ipv4()

This class sets ECMP weight for IPv4 nexthops.

set_weighted_ecmp_nexthop_ipv4()

Configures ECMP weight for the specified IPv4 nexthop.

Syntax

`set_weighted_ecmp_nexthop_ipv4``(`*addr, weight*`)`

where:

Parameter	Description
addr	The IPv4 address of the nexthop (string).
weight	The ECMP weight of the specified nexthop. An integer from 1-4.

Returns

Boolean (`True` on success, otherwise `False`).

class WeightedEcmp_ipv4_show()

This class gets the ECMP weight of a specified IPv4 nexthop.

get_weighted_ecmp_ipv4()

Gets ECMP weight for an IPv4 nexthop.

Syntax

`get_weighted_ecmp_ipv4``(`*addr*`)`

where:

Parameter	Description
addr	The IPv4 address of the nexthop (string).

Returns

A dictionary showing ECMP weight information:

Parameter	Description
ipv4_nexthop_address	The IPv4 address of the nexthop (string).
ipv4_nexthop_weight	The IPv4 nexthop weight (string).

class WeightedEcmp_ipv6()

This class sets ECMP weight for IPv6 nexthops.

set_weighted_ecmp_nexthop_ipv6()

Sets ECMP weight for the specified IPv6 nexthop.

Syntax

`set_weighted_ecmp_nexthop_ipv6``(`*addr, weight*`)`

where:

Parameter	Description
addr	The IPv6 address of the nexthop (string).
weight	The ECMP weight of the specified nexthop. An integer from 1-4.

Returns

Boolean (`True` on success, otherwise `False`).

class WeightedEcmp_ipv6_show()

This class gets the ECMP weight of a specified IPv6 nexthop.

get_weighted_ecmp_ipv6()

Gets ECMP weight for a specified IPv6 nexthop.

Syntax

`get_weighted_nexthop_ipv6``(`*addr*`)`

where:

Parameter	Description
addr	The IPv6 address of the nexthop (string).

Returns

A dictionary showing ECMP weight information:

Parameter	Description
ipv6_nexthop_address	The IPv6 address of the nexthop (string).
ipv6_nexthop_weight	The IPv6 nexthop weight (string).

class WeightedEcmp_interface()

This class sets the ECMP weight for a specified switch interface.

set_weighted_ecmp_interface()

Configures the ECMP weight of the specified switch interface.

Syntax

`set_weighted_ecmp_interface``(`*if_name, weight*`)`

where:

Parameter	**Description**
if_name	The name of the switch interface (string). For example: "`Ethernet1/12`".
weight	The ECMP weight of the specified interface. An integer from 1-4.

Returns

Boolean (`True` on success, otherwise `False`).

class WeightedEcmp_interface_show()

This class gets the ECMP weight of a specified switch interface.

get_weighted_ecmp_interface()

Displays ECMP weight information for the specified switch interface.

Syntax

`get_weighted_ecmp_interface``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The name of the switch interface (string). For example: "`Ethernet1/12`".

Returns

A dictionary showing ECMP weight information:

Parameter	**Description**
interface_name	The name of the switch interface (string). For example: "`Ethernet1/12`".
interface_weight	The ECMP weight of the specified interface. An integer from 1-4.

FDB Module

This module manages the Forwarding Database (FDB) configuration on the switch.

To use this module, in the Python file or in the Python interpreter, enter:

`import fdbApi`

class FDB()

This class provides functions that get and set FDB configurations.

python_get_fdb_info()

Displays all MAC addresses that match the search criteria.

Syntax

`python_get_fdb_info``(`*dict_fdb_filter*`)`

where *dict_fdb_filter* contains the following optional variables:

Parameter	**Description**
fdb_type	The type of FDB to filter on (string). Valid values: `"static"`, `"multicast"`, `"dynamic"`.
mac_address	The MAC address to filter on (string).
interfaces	The list of switch interfaces to filter on. A list containing interface names, which are string (for example, `"Ethernet1/12"`).
vlan_id	The VLANs to filter on. An integer from 1-4094.

Returns

Multiple possible return values:

- an error description string
- a boolean with the value `false`
- a dictionary showing MAC address information that matched the search filter:

Parameter	**Description**
address_table	The list of MAC addresses matching the search filter. A list with MAC addresses as string.
is_static	Whether the MAC address is statically configured or dynamically learned (boolean). Values: `True` for static MAC address or `False` for dynamic MAC address.
mac_address	The MAC address (string).

Parameter	Description
if_name	The name of the switch interface (string). For example: `“Ethernet1/12”`).
vlan_id	The VLAN of the MAC address; An integer from 1-4094.

python_get_fdb_count_info()

Displays the number of MAC table entries matching the search criteria.

Syntax

`python_get_fdb_count_info``(`*dict_fdb_filter*`)`

where *dict_fdb_filter* contains the following optional variables:

Parameter	Description
fdb_type	The type of FDB to filter on (string). Valid values: `”static”`, `”multicast”`, `”dynamic”`.
mac_address	The MAC address to filter on (string).
interfaces	The list of switch interfaces to filter on. A list containing interface names, which are string (for example, `“Ethernet1/12”`).
vlan_id	The VLANs to filter on. An integer from 1-4094.

Returns

Multiple possible return values:

- an error description string
- a boolean with the value `False`
- a dictionary showing MAC address information that matched the search filter:

Parameter	Description
dynamic_add_cnt	The number of dynamically learned MAC addresses matching the filter (integer).
static_add_cnt	The number of statically configured MAC addresses matching the filter (integer).
multicast_add_cnt	The number of multicast MAC addresses matching the filter (integer).
total_in_use_cnt	The total numbers of MAC address matching the filter. It’s the sum of the number of dynamically learned, statically configured, and multicast MAC addresses (integer).

python_get_static_fdb_cfg()

Displays all statically configured MAC addresses matching the search criteria.

Syntax

`python_get_static_fdb_cfg``(`*dict_fdb_filter*`)`

where *dict_fdb_filter* containg the following optional variables:

Parameter	**Description**
mac_address	The MAC address to filter on (string).
interfaces	The list of switch interfaces to filter on. A list containing interface names, which are string (for example, `“Ethernet1/12”`).
vlan_id	The VLANs to filter on. An integer from 1-4094.

Returns

Multiple possible return values:

- an error description string
- a boolean with the value `False`
- a dictionary showing MAC address information that matched the search filter:

Parameter	**Description**
address_table	The list of MAC addresses matching the search filter. A list with MAC addresses as string.
is_static	Whether the MAC address is statically configured or dynamically learned (boolean). Values: `True` for static MAC address or `False` for dynamic MAC address.
mac_address	The MAC address (string).
if_name	The name of the switch interface (string). For example: `“Ethernet1/12”`.
vlan_id	The VLAN of the MAC address. An integer from 1-4094.

python_add_static_mac()

Configures a static MAC table entry or adds interfaces to already configured static MAC table entries.

Syntax

`python_add_static_mac``(`*return_dict_added_static_fdb*`)`

where *return_dict_added_static_fdb* contains the following variables:

Parameter	**Description**
mac_address	The MAC address to configure (string).
interfaces	The list of switch interfaces for the MAC address. A list containing interface names, which are string. For example, `“Ethernet1/12”`.
vlan_id	The VLAN for the MAC address. An integer from 1-4094.
vxlan	The VXLAN MAC of Overlay Network (integer). Valid values: ● `0` - Not VXLAN MAC ● `1` - VXLAN MAC Default value: `0`.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

python_remove_bridge_fdb()

Deletes existing MAC table entries or interfaces from already existing multicast MAC table entries.

Syntax

`python_remove_bridge_fdb``(`*dict_removed_fdb_filter*`)`

where *dict_removed_fdb_filter* contains the following variables:

Parameter	Description
fdb_type	The type of MAC address to delete (string). Valid values: `“static”`, `”dynamic”`.
mac_address	The MAC address to delete (string).
interfaces	The list of switch interfaces for the multicast MAC address. A list containing interface names, which are string. For example, `“Ethernet1/12”`.
vlan_id	(Optional) The VLAN for the MAC address. An integer from 1-4094.
vxlan	(Optional) The VXLAN MAC of Overlay Network (integer). Valid values: ● `0` - Not VXLAN MAC ● `1` - VXLAN MAC Default value: `0`.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

python_get_fdb_plearning_state()

Displays the status of MAC learning for a specific switch interface.

Syntax

`python_get_fdb_plearning_state``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The name of the switch interface (string). For example: `“Ethernet1/12”`.

Returns

Multiple possible return values:

- error description string
- a boolean with the value `false`
- a dictionary showing the status of MAC learning on the specified interface:

Parameter	**Description**
learning_state	The status of MAC learning (string). Valid values: `“enabled”,”disabled”`.

python_set_fdb_plearning_state()

Configures MAC learning on a specific interface.

Syntax

`python_set_fdb_plearning_state``(`*if_name, learning_state*`)`

where:

Parameter	**Description**
if_name	The name of the switch interface (string). For example: `“Ethernet1/12”`
learning_state	The status of MAC learning (string). Valid values: `“enabled”`, `”disabled”`.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

python_get_fdb_glearning_cfg()

Displays the status of global MAC learning on the switch.

Syntax

`python_get_fdb_glearning_cfg``()`

Returns

Multiple possible return values:

- error description string
- a boolean with the value `false`
- a dictionary showing the status of global MAC learning on the switch:

Parameter	**Description**
global_learning_state	The status of global MAC learning (string). Valid values: `“enabled”`, `”disabled”`.

python_set_fdb_glearning_cfg()

Configures global MAC learning on the switch.

Syntax

`python_set_fdb_glearning_cfg``(`*glearning_state*`)`

where:

Parameter	**Description**
glearning_state	The status of global MAC learning (string). Valid values: `“enabled”`, `”disabled”`.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

python_get_fdb_aging_time_cfg()

Displays MAC table entry aging time configuration.

Syntax

`python_get_fdb_aging_time_cfg``()`

Returns

Multiple possible return values:

- error description string
- a boolean with the value `false`
- a dictionary showing the MAC aging time configuration:

Parameter	**Description**
aging_time	The MAC aging time, in seconds. An integer from 0-1000000.

python_set_fdb_aging_time_cfg()

Configures MAC table entry aging time on the switch.

Syntax

`python_set_fdb_aging_time_cfg``(`*aging_time*`)`

where:

Parameter	**Description**
aging_time	The MAC aging time, in seconds. An integer from 0-1000000.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

FDMB Module

This module manages MAC Move Loop Detections.

To use this module, in the Python file or in the Python interpreter, enter:

```
import fdbmApi
```

class FDBM

This class provides functions to manage MAC Move Loop Detection.

python_get_fdbm_info

Gets all MAC Move Loop Detection information.

Syntax

```
python_get_fdbm_info()
```

Returns

A dictionary showing MAC Move Loop Detection information:

Parameter	**Description**
enable	Enable or disable mac address-table loop-detect port-down (string). Valid values: `"yes"`, `"no"`.
down_edge_port	Enable or disable mac address-table loop-detect port-down edge-port (string). Valid values: `"yes"`, `"no"`.
error_disable_recovery	Enable or disable error-disable recovery cause mac-loop-detect (string). Valid values: `"yes"`, `"no"`.
errdis_interfaces	A list of interfaces error-disabled by MAC Move Loop Detection.
if_name	The name of the error-disabled interface (string). An interface name. For example: `"Ethernet1/12"`, `"po10"`.
time_left	The left time. After it, interface will be recovered from error-disable status if error-disable recovery is enabled. An integer from 1-65535.

python_get_fdbm_cfg

Gets all MAC Move Loop Detection setting.

Syntax

`python_get_fdbm_cfg``()`

Returns

A dictionary showing MAC Move Loop Detection setting:

Parameter	**Description**
enable	Enable or disable mac address-table loop-detect port-down (string). Valid values: `"yes"`, `"no"`.
down_edge_port	Enable or disable mac address-table loop-detect port-down edge-port (string). Valid values: `"yes"`, `"no"`.
error_disable_recovery	Enable or disable error-disable recovery cause mac-loop-detect (string). Valid values: `"yes"`, `"no"`.

python_set_fdbm_mac_loop_detect

Sets mac address-table loop-detect port-down.

Syntax

`python_set_fdbm_mac_loop_detect``(`*status*`)`

where:

Parameter	**Description**
status	Whether mac address-table loop-detect port-down is set or unset (string). Valid values: `"yes"` for set, `"no"` for unset.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_fdbm_down_edge_port

Sets mac address-table loop-detect port-down edge-port.

Syntax

`python_set_fdbm_down_edge_port``(`*status*`)`

where:

Parameter	**Description**
status	Whether mac address-table loop-detect port-down edge-port is set or unset (string). Valid values: `"yes"` for set, `"no"` for unset.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_fdbm_errdis_recover

Sets error-disable recovery cause mac-loop-detect.

Syntax

`python_set_fdbm_errdis_recover``(`*status*`)`

where:

Parameter	**Description**
status	Whether error-disable recovery cause mac-loop-detect is set or unset (string). Valid values: `"yes"` for set, `"no"` for unset.

Returns

Boolean (`True` on success, otherwise `False`).

FCoE Initialization Protocol Snooping Module

This module manages FCoE Initialization Protocol Snooping (FIPS).

To use this module, in the Python file or in the Python interpreter, enter:

```
import fipsApi
```

class FIPS()

This class provides functions to manage FCoE FIP Snooping configuration, and to retrieve the FCoE login information.

python_fips_is_platform_supported()

Returns whether FCoE FIP Snooping is supported on a particular hardware platform or not.

Syntax

```
python_fips_is_platform_supported()
```

Returns

Boolean (`True` on success, otherwise `False`).

python_fcoe_fips_get_fipsmode()

Gets FCoE FIPS mode and status.

Syntax

```
python_fcoe_fips_get_fipsmode()
```

Returns

A dictionary containing IP address status details:

Parameter	**Description**
fips_state	The FIPS feature state (string). Valid values: `“enable”`, `“disable”`. Default value: `“disable”`.
fips_mode	The FIPS feature mode (string). Valid values: `"global"`, `"per-vlan"`. **Note:** Valid only if FIPS is enabled.

python_fcoe_fips_set_fipsmode()

Sets FCoE FIPS mode and status. The FIPS feature can be enabled in either global or per-VLAN mode. The same method also disables FCoE FIPS.

Syntax

`python_fcoe_fips_set_fipsmode``(`*fips_state*`)`

where:

Parameter	**Description**
fips_state	The FIPS feature state (string). Valid values: `“enable”`, `“disable”`.
fips_mode	The FIPS feature mode (string). Valid values: `"global"`, `"per-vlan"`. **Note:** Valid only if FIPS is enabled.

Returns

Boolean (`True` on success, otherwise `False`).

python_fcoe_fips_get_vlan()

Gets the FIPS enabled VLANs list and the FCMAP.

Note: This API is valid only for per-VLAN FIPS mode. When FIPS mode is Global, the list is empty.

Syntax

`python_fcoe_fips_get_vlan``()`

Returns

A dictionary containing IP address status details:

Parameter	**Description**
vlan	The VLAN number. An integer from 1-4093.
fcmap	The FCMAP configured for a given VLAN. A string in hexadecimal format. Valid values: `“0x000000”` - `“0xffffff”`. Default value: `“0x0efc00”`.

python_fcoe_fips_set_vlan()

Enables per-VLAN FIPS on a specified VLAN.

Note: This API is valid only for per-VLAN FIPS mode. When FIPS mode is Global, the switch displays an error message.

Syntax

`python_fcoe_fips_set_vlan``(`*vlan, fcmap*`)`

where:

Parameter	Description
vlan	The VLAN number. An integer from 1-4093.
fcmap	(Optional) The FCMAP configured for a given VLAN. A string in hexadecimal format. Valid values: “`0x000000`” - “`0xffffff`”. Default value: “`0x0efc00`”.

Returns

Boolean (`True` on success, otherwise `False`).

python_fcoe_fips_set_fcmap()

Sets FCMAP on a specified VLAN.

Note: This API is valid only for per-VLAN FIPS mode. When FIPS mode is Global, the switch displays an error message. `False` is displayed if FIPS is not enabled on the VLAN.

Syntax

`python_fcoe_fips_set_fcmap``(`*vlan, fcmap*`)`

where:

Parameter	Description
vlan	The VLAN number. An integer from 1-4093.
fcmap	The FCMAP configured for a given VLAN. A string in hexadecimal format. Valid values: “`0x000000`” - “`0xffffff`”.

Returns

Boolean (`True` on success, otherwise `False`).

python_fcoe_fips_del_vlan()

Disables per-VLAN FIPS on a specified VLAN.

Note: This API is valid only for per-VLAN FIPS mode. When FIPS mode is Global, the switch displays an error message. `False` is displayed if FIPS is not enabled on the VLAN.

Syntax

`python_fcoe_fips_del_vlan``(`*vlan*`)`

where:

Parameter	Description
vlan	The VLAN number. An integer from 1-4093.

Returns

Boolean (`True` on success, otherwise `False`).

python_fcoe_fips_get_fcf()

Gets the list of FCoE Forwarders (FCF) discovered across all VLANs.

Note: This API is valid only when FIPS is enabled.

Syntax

`python_fcoe_fips_get_fcf``()`

Returns

A dictionary containing FCF information:

Parameter	Description
ifname	The interface name (string).
vlan	The VLAN number. An integer from 1-4093.
fcf_mac	The FCF MAC address (string).

python_fcoe_fips_get_fcf_by_vlan()

Gets the list of FCoE Forwarders (FCF) discovered on a specified VLAN.

Note: This API is valid only when FIPS is enabled.

Syntax

`python_fcoe_fips_get_fcf_by_vlan``(`*vlan*`)`

where:

Parameter	**Description**
vlan	The VLAN number. An integer from 1-4093.

Returns

A dictionary containing FCF information:

Parameter	**Description**
ifname	The interface name (string).
vlan	The VLAN number. An integer from 1-4093.
fcf_mac	The FCF MAC address (string).

python_fcoe_fips_get_database()

Gets the list of logged-in FCoE Enodes across all VLANs on all FCFs.

Note: This API is valid only when FIPS is enabled.

Syntax

`python_fcoe_fips_get_database``()`

Returns

A dictionary containing FCoE information:

Parameter	**Description**
ifname	The interface name (string).
vlan	The VLAN number. An integer from 1-4093.
vn_port mac	The VN_Port MAC address - Fabric assigned MAC (FPMA). A string.
fcf_mac	The FCF MAC address (string).
enode mac	The enode MAC address (string).

python_fcoe_fips_get_database_by_vlan()

Gets the list of logged-in FCoE Enodes on a a specified VLAN.

Note: This API is valid only when FIPS is enabled.

Syntax

`python_fcoe_fips_get_database_by_vlan``(`*vlan*`)`

where:

Parameter	**Description**
vlan	The VLAN number. An integer from 1-4093.

Returns

A dictionary containing FCoE information:

Parameter	**Description**
ifname	The interface name (string).
vlan	The VLAN number. An integer from 1-4093.
vn_port mac	The VN_Port MAC address - Fabric assigned MAC (FPMA). A string.
fcf_mac	The FCF MAC address (string).
enode mac	The enode MAC address (string).

python_fcoe_fips_get_ifmode()

Gets the FIPS interface configuration.It includes the interface state and FCF mode.

- FIPS Interface State: FIPS can be enabled/disabled per interface level. FIPS ACLs are not installed on FIPS disabled interfaces
- FCF Mode of Interface: by default all interfaces learn FCFs by listening to Discovery Advertisements. This mode is called Auto mode. In this mode, the interface can be transitioned to an FCF port or Enode port based on the connected neighbor

Notes:

- This API is valid only when FIPS is enabled.
- To disable FCF learning on any interface when it is connected to Enode, FCF mode must be OFF.
- To avoid unwanted logins on an interface connected to the uplink FCF, FCF mode must be ON.

Syntax

`python_fcoe_fips_get_ifmode``(`*ifname*`)`

where:

Parameter	**Description**
ifname	The interface name (string).

Returns

A dictionary containing FCoE information:

Parameter	**Description**
ifname	The interface name (string).
fips_state	The FIPS feature state (string). Valid values: `“enable”`, `“disable”`.
fcf_mode	The FCF mode (string). Valid values: `"Auto"`, `"On"`, `"Off"`. Default value: `"Auto"`.

python_fcoe_fips_set_ifmode()

Sets the FIPS interface configuration. It includes the interface state and FCF mode.

- FIPS Interface State: FIPS can be enabled/disabled per interface level. FIPS ACLs are not installed on FIPS disabled interfaces
- FCF Mode of Interface: by default all interfaces learn FCFs by listening to Discovery Advertisements. This mode is called Auto mode. In this mode, the interface can be transitioned to an FCF port or Enode port based on the connected neighbor

Notes:

- This API is valid only when FIPS is enabled.
- To disable FCF learning on any interface when it is connected to Enode, FCF mode must be OFF.
- To avoid unwanted logins on an interface connected to the uplink FCF, FCF mode must be ON.

Syntax

`python_fcoe_fips_set_ifmode``(`*ifname, fips_state, fcf_mode*`)`

where:

Parameter	**Description**
ifname	The interface name (string).
fips_state	The FIPS feature state (string). Valid values: `“enable”`, `“disable”`. Default value: `“enable”`.
fcf_mode	(Optional) The FCF mode (string). Valid values: `"Auto"`, `"On"`, `"Off"`. Default value: `“Auto”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_fcoe_fips_get_control_stats()

Gets the FIPS Snooping Control packets counters.

Syntax

```
python_fcoe_fips_get_control_stats()
```

Returns

A dictionary containing FIPS Snooping Control packets counters information:

Parameter	**Description**
FCF Added	The number of FCFs discovered (integer).
FCF Removed	The number of FCFs removed (integer).
FCoE Added	The number of FCOEs added (integer).
FCoE Removed	The number of FCOEs removed (integer).
Total FIP Packets received	The total number of FIP packets received (integer).
Total FIP Packets sent	The total number of FIP packets sent (integer).
Vlan Requests received	The number of VLAN requests received (integer).
Discovery Adv received	The number of discovery advertisements received (integer).
FLOGI Ack received	The number of FLOGI Ack received (integer).
FLOGO Ack received	The number of FLOGO Ack received (integer).
CVL received	The number of CVL received (integer).
Enode FKA received	The number of Enode keep-alive received (integer).
VN-Port FKA received	The number of vn-port keep-alive received (integer).
Forward Unicast FIP packets	The number of unicast FIP packets forwarded (integer).
Forward ALL_FCF_MAC packets	The number of ALL_FCF_MAC packets forwarded (integer).
Forward ALL_ENODE_MAC packets	The number of ALL_ENODE_MAC packets forwarded (integer).
FIP decode errors	The number of FIP decode errors (integer).

python_fcoe_fips_get_acl_stats()

Gets the FIPS Access Control Lists (ACLs).

Syntax

`python_fcoe_fips_get_acl_stats``()`

Returns

A dictionary containing ACL information:

Parameter	Description
dict_acl	A dictionary of installed ACLs.

python_fcoe_fips_get_acl_count()

Gets the FIPS Access Control Lists (ACLs) total count.

Syntax

`python_fcoe_fips_get_acl_count``()`

Returns

A dictionary containing ACL information:

Parameter	Description
Number of FCFs detected	The number of FCF's discovered (integer).
Number of ACLs installed	The number of FIPS ACLs installed (integer).

python_fcoe_fips_set_server_port_trunk()

Sets FIPS server-port-trunk on a given LAG.

Note: This configuration is required only if the LAG is between switch and server, and the server ports are FCoE capable.

Syntax

`python_fcoe_fips_set_server_port_trunk``(`*port_channel, status*`)`

where:

Parameter	**Description**
port_channel	The LAG name (string).
status	The FIPS feature state of the server-port-trunk (string). Valid values: `“enable”`, `“disable”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_fcoe_fips_get_server_port_trunk()

Gets FIPS server-port-trunk on a given LAG.

Note: This configuration is required only if the LAG is between switch and server, and the server ports are FCoE capable.

Syntax

`python_fcoe_fips_get_server_port_trunk``(`*port_channel*`)`

where:

Parameter	**Description**
port_channel	The LAG name (string).

Returns

A dictionary containing FIPS information:

Parameter	**Description**
port_channel	The LAG name (string).
server-port-trunk status	The FIPS feature state of the server-port-trunk (string). Valid values: `“enable”`, `“disable”`. Default value: `“disable”`.

HostpCpy Module

This module updates the image and configuration file via TFTP.

To use this module, in the Python file or in the Python interpreter, enter:

`import hostpCpyApi`

class HostpCpy()

This class provides functions to update the image and configuration files via TFTP.

update_startup_cfg_tftp()

Updates the startup configuration using TFTP.

Syntax

`update_startup_cfg_tftp``(`*serverip, cfgfile, vrf_name*`)`

where:

Parameter	Description
serverip	The server IP address (string). A valid IP address.
cfgfile	The configuration source file. A string up to 256 characters long.
vrf_name	(Optional) The VRF name. A string up to 64 characters long. Default value: `“Management”`.

Returns

A dictionary that indicates the startup configuration update status:

Parameter	Description
status	Configuration update status (string).

update_image_tftp()

Updates the image using TFTP.

Syntax

update_image_tftp(*serverip, imgfile, imgtype, vrf_name*)

where:

Parameter	**Description**
serverip	The server IP address (string). A valid IP address.
imgfile	The image file name. A string up to 256 characters long.
imgtype	System image type (string). Valid values: `“all”`, `”boot”`, `”onie”`, `”os”`.
vrf_name	(Optional) The VRF name. A string up to 64 characters long. Default value: `“Management”`.

Returns

A dictionary that indicates the image update status:

Parameter	**Description**
status	Image update status (string).

get_update_image_status()

Gets the image update status.

Syntax

`get_update_image_status``()`

Returns

A dictionary that indicates the image update status:

Parameter	**Description**
status	Image update status (string).

switch_reboot()

Halts the system and performs a cold restart.

Syntax

```
switch_reboot()
```

Returns

Boolean (`True` on success, otherwise `False`).

HSC Module

This module manages Hardware Switch Controller (HSC).

To use this module, in the Python file or in the Python interpreter, enter:

`import hscApi`

class HscCfg()

The functions in this class get and set HSC configuration.

set_hsc_mode()

Sets the HSC mode.

Syntax

`set_hsc_mode``(`*mode*`)`

where:

Parameter	**Description**
mode	The HSC mode (string). Valid values: `“vtep”`, `“none”`.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

get_hsc_mode()

Gets the HSC mode.

Syntax

`get_hsc_mode``()`

Returns

A dictionary containing HSC mode details:

Parameter	**Description**
mode	The HSC mode (string). Valid values: `“vtep”`, `“none”`.

set_hsc_device_name()

Sets the HSC VTEP device name. A null string “” deletes the device name.

Syntax

`set_hsc_device_name``(`*devicename_in*`)`

where:

Parameter	Description
devicename_in	The device name. A string between 1-254 characters, or null “”.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

get_hsc_device_name()

Gets the HSC VTEP device name.

Syntax

`get_hsc_device_name``()`

Returns

A dictionary containing device name information:

Parameter	Description
devicename	The device name. A string between 2-255 characters.

set_hsc_ha_mode()

Sets the HSC HA mode.

Syntax

`set_hsc_ha_mode``(`*mode*`)`

where:

Parameter	Description
mode	The HSC ha mode (string). Valid values: `“vlag”`, `“none”`.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

get_hsc_ha_mode()

Gets the HSC HA mode.

Syntax

`get_hsc_ha_mode``()`

Returns

A dictionary containing HSC ha mode:

Parameter	Description
ha_mode	The HSC ha mode (string). Valid values: `“vlag”`, `“none”`.

set_hsc_controller()

Sets the HSC controller configuration.

Syntax

`set_hsc_controller``(`*provider_in, ip_in, vrf_in, port_in, backoff, inactivity*`)`

where:

Parameter	**Description**
provider_in	(Optional) A specified value which shows the controller provider (string). Valid values: ● `"nsx"` - NSXGW provider ● `"none"` - remove all the controller configuration Default value: `"nsx"`.
ip_in	The IP address of the controller (string). A IPv4 address in the following format: `"a.b.c.d"`.
port_in	(Optional) The port number. An integer from 1-65535. Default value: `6640`.
vrf_in	(Optional) The VRF context value (string). Valid values: `"management"`, `"default"`. Default value: `"management"`.
backoff	(Optional) The backoff timer for re-connecting. An integer from 1000-60000, or `0` to disable backoff timer. Default value: `8000`.
inactivity	(Optional) The inactivity-probe timer for keeping alive, in milliseconds. An integer from 10000-3600000, or `0` to disable the inactivity-probe timer. Default value: `120000`.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

get_hsc_controller()

Gets the HSC controller configuration.

Syntax

```
get_hsc_controller()
```

Returns

A dictionary containing controller configuration:

Parameter	Description
provider	A specified value which shows the controller provider (string). Valid values: ● `"nsx"` - NSXGW provider ●
ip	The IP address of the controller (string). A IPv4 address in the following format: `"a.b.c.d"`.
port	The port number. An integer from 1-65535.
vrf	The VRF context value (string). Valid values: `"management"`, `"default"`.
backoff	The controller backoff timer for re-connecting. An integer from 1000-60000, or `0` if the backoff timer is disabled. Default value: `8000`.
inactivity-probe	The inactivity-probe timer for keeping alive, in milliseconds. An integer from 10000-3600000, or `0` if the inactivity-probe timer is disabled. Default value: `120000`.

set_hsc_tunnel()

Sets the HSC tunnel IP.

Syntax

`set_hsc_tunnel``(`*tunnel*`)`

where:

Parameter	**Description**
tunnel	The IP address to be used as tunnel IP (string). A IPv4 address in the following format: `"a.b.c.d"`. A null string “” deletes the configuration.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

get_hsc_tunnel()

Gets the HSC tunnel IP.

Syntax

`get_hsc_tunnel``()`

Returns

A dictionaty containing tunnel IP information:

Parameter	**Description**
tunnel_ip	The IP address used as VXLAN tunnel IP (string). A IPv4 address in the following format: `"a.b.c.d"`.

set_hsc_vtep()

Sets the HSC VTEP configuration.

Syntax

`set_hsc_vtep``(`*id, ip, port, vrf, username, passwd*`)`

where:

Parameter	**Description**
id	The VTEP ID (integer). Valid values: `1`,`2`.
ip	The local VTEP IP address used to connect with HSC (string). A IPv4 address in the following format: `"a.b.c.d"`.
port	(Optional) The port number used to connect with HSC. An integer from 1-65535. Default value: `443`.
vrf	(Optional) The VRF context value (string). Valid values: `"management"`, `"default"`.
username	(Optional) The user name used by the VTEP to connect with HSC. An alphanumeric string, starting with an alpha character, between 2-28 characters. Default value: `“admin”`.
passwd	(Optional) The password used by the VTEP to connect with HSC. A string between 8-80 characters, or `“admin”`. Default value: `“admin”`.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

unset_hsc_vtep()

Deletes the HSC VTEP instance configuration.

Syntax

`unset_hsc_vtep``(`*id*`)`

where:

Parameter	**Description**
id	The VTEP ID (integer). Valid values: `1`,`2`.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

set_hsc_vtep_ip()

Sets the HSC VTEP IP.

Syntax

`set_hsc_vtep_ip``(`*id, ip*`)`

where:

Parameter	**Description**
id	The VTEP ID (integer). Valid values: `1`,`2`.
ip	The local VTEP IP address used to connect with HSC (string). A IPv4 address in the following format: `"a.b.c.d"`.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

set_hsc_vtep_port()

Sets the HSC VTEP port list.

Syntax

`set_hsc_vtep_port``(`*id, eaction, elist, paction, plist, vaction, vlist*`)`

where:

Parameter	**Description**
id	The VTEP ID (integer). Valid values: `1`,`2`.
eaction *paction* *vaction*	The configuration type (string). Valid values: `"add"`, `"remove"`, `"set"`. **Note:** “`Set`” overwrites the current port-lists.
elist	The ethernet port list (string). For example: `"1/3, 1/5-10"`.
plist	The aggregation port list (string). For example: `"2,4,5-8"`.
vlist	The VLAG instance list (string). For example: `"1,5,7-9"`.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

class HscInfo()

The functions in this class get HSC global parameters.

get_hsc_info_controller_connect()

Gets the HSC controller connection information.

Syntax

`get_hsc_info_controller_connect``()`

Returns

A list containing HSC controller connection information:

Parameter	**Description**
type	The string value for Connection Type. For example: `"SSL (NSX Controller)"`.
peer	The peer of the connection (string). For example: `"A.B.C.D:port"`.
inact	The inactive probe time in milliseconds (integer). `-1` means the inactive probe time is invalid.
backoff	The maximum backoff time in milliseconds (integer). `-1` means the inactive backoff time is invalid.

get_hsc_info_restc_connection()

Gets the HSC rest client connection information.

Syntax

`get_hsc_info_restc_connection``()`

Returns

A list containing HSC rest client connection information:

Parameter	**Description**
owner	The rest client VTEP name (string). Valid values: `"vtep1"`, `"vtep2"`.
peer	The peer address string, in the following format: `"{https|http}://a.b.c.d:port"`.

Parameter	Description
vrf_name	The VRF context (string). Valid values: `"management"`, `"default"`.
state	The connection state (string). Valid values: ● `"init"` - HSCD is in initiate state ● `"logging"` - connecting to NWVD, HSDC rest client is in logging state ● `"running"` - connection to NWVD has been created, HSDC rest client is running state ● `"unready"` - NWVD is not ready, HSDC is in checking state

get_hsc_info_vtep_basic()

Gets the HSC VTEP basic information.

Syntax

```
get_hsc_info_vtep_basic()
```

Returns

A dictionary containing HSC VTEP basic information:

Parameter	Description
status	The VTEP status (string). Valid values: `"enable"`, `"disable"`.
mode	The HA mode configuration (sting). Valid values: `"vlag"`, `"none"`.
name	The HSC device name (string).
tunnel	The local tunnel IP address for NSXGW. A string in the following format: “`a.b.c.d`”.
bfd	The BFD status value (string). Valid values: `"enabled"`, `"disabled"`.
port_counts	The physical port count (integer).
mapping_counts	The total mappings count (integer).

get_hsc_info_vtep_mac()

Gets the HSC VTEP MAC information.

Syntax

`get_hsc_info_vtep_mac``()`

Returns

A dictionary containing HSC VTEP MAC information:

Parameter	**Description**
count	The number of the listed MAC-table items (integer).
vni	The VXLAN Network Identifier. An integer from 1-16777214..
mac_address	The MAC address value string.
tunnel	The tunnel IPv4 address. A string in the following format: “`a.b.c.d`”.

get_hsc_info_tunnel ()

Gets the HSC tunnel information.

Syntax

`get_hsc_info_tunnel` `()`

Returns

A list containing HSC tunnel information:

Parameter	**Description**
local_ip	The IP address of local switch. This can be the management IP address of the local switch (string). An IPv4 address in the following format: “`a.b.c.d`”.
remote_ip	The remote IP address of the tunnel (string). An IPv4 address in the following format: “`a.b.c.d`”.
bfd_enables	The remote BFD status (string). Valid values: `"true"`, `"false"`.

get_hsc_info_virtual_net()

Gets the HSC virtual network information.

Syntax

```
get_hsc_info_virtual_net()
```

Returns

A list containing HSC virtual network information:

Parameter	**Description**
vni	The VXLAN Network Identifier. An integer from 1-16777214.
name	The unique string value name of the virtual network (string).

get_hsc_info_virtual_port()

Gets the HSC virtual port information.

Syntax

```
get_hsc_info_virtual_port()
```

Returns

A list containing HSC virtual port information:

Parameter	**Description**
port_name	The virtual port name (string).
vlanid	The Vlan ID (integer).
vnid	The VXLAN Network Identifier (integer).

hwProfile Module

This module manages hardware breakout profiles.

To use this module, in the Python file or in the Python interpreter, enter:

```
import hwProfileApi
```

class HwProfileBreakout()

The function in this class get and set the hardware breakout profiles.

get_hw_breakout()

Gets a list of dictionaries containing details about the configured hardware breakout profiles the breakout mode.

Syntax

`get_hw_breakout``(`*default*`)`

where:

Parameter	Description
default	(Optional) The default hardware breakout profile (boolean). Valid values: `True`, `False`.

Returns

`None` in case of error, or a list of dictionaries describing one interface hardware breakout profile:

Parameter	Description
if_name	The interface name (string).
breakout	The hardware breakout mode of the interface (string). Valid values: `"10G-4x"`, `"25G-4x"`, `"40G-1x"`, `"50G-2x"`, `"100G-1x"`.

set_hw_breakout()

Configure the hardware breakout profile of the interface.

Syntax

`set_hw_breakout``(`*if_name, breakout*`)`

where:

Parameter	**Description**
if_name	The interface name (string).
breakout	The hardware breakout mode of the interface (string). Valid values: `"10G-4x"`, `"25G-4x"`, `"40G-1x"`, `"50G-2x"`, `"100G-1x"`.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

IGMPSnooping Module

This module manages Internet Group Management Protocol (IGMP) snooping.

To use this module, in the Python file or in the Python interpreter, enter:

```
import igmpSnoopingApi
```

class IgmpSnooping()

This class provides functions to get and set IGMP snooping status.

lastErrorGet()

Gets the last IGMP Snooping error in a readable format.

Syntax

```
lastErrorGet()
```

Returns

A string showing the last IGMP Snooping error in a readable format.

globalStatusGet()

Gets the IGMP snooping status on the device.

Syntax

```
globalStatusGet()
```

Returns

A dictionary containing IGMP status information where:

Parameter	**Description**
status	Whether IGMP snooping is enabled or disabled (string). Valid values: `"enable"`, `"disable"`.

globalStatusSet()

Enables or disables IGMP snooping on the device.

Syntax

`globalStatusSet(`*enable***`)`**

where:

Parameter	**Description**
enable	IGMP snooping status (boolean). Valid values: `True`, `False`.

Returns

Boolean (`True` on success, otherwise `False`).

vlanStatusGet()

Gets the IGMP snooping status for all VLAN(s) or for a specified VLAN.

Syntax

`vlanStatusGet(`*vlan***`)`**

where:

Parameter	**Description**
vlan	(Optional) The VLAN ID. If no VLAN is specified, the information for all VLANs is displayed. An integer from 1-4093.

Returns

A list of dictionaries containing the IGMP snooping status for all VLANs:

Parameter	**Description**
vlan	The VLAN ID. An integer from 1-4093.
status	Whether IGMP snooping is enabled or disabled (string). Valid values: `”enable”`, `”disable”`, `“n/a”`.
fast_leave	Whether IGMP fast leave is enabled or disabled (string). Valid values: `”enable”`, `”disable”`, `“n/a”`.
query_interval	The IGMP query interval, in seconds. An integer from 1-18000. Default value: `125`.
version	IGMP snooping version (integer). Valid values: 2,3.

vlanStatusSet()

Sets the IGMP snooping information on a specified VLAN.

Syntax

`vlanStatusSet``(`*vlan, status, fastLeave, queryInterval, version*`)`

where:

Parameter	Description
vlan	The VLAN ID. An integer from 1-4093.
status	Whether IGMP snooping is enabled or disabled (string). Valid values: `"enable"`, `"disable"`, `"n/a"`.
fastLeave	(Optional) Whether IGMP fast leave is enabled or disabled (string). Valid values: `"enable"`, `"disable"`, `"n/a"`. Default value: `None`.
queryInterval	(Optional) IGMP query interval, in seconds. An integer from 1-18000. Default value: `None`.
version	(Optional) IGMP snooping version (integer). Valid values: `2`,`3`. Default value: `None`.

Returns

Boolean (`True` on success, otherwise `False`).

groupGet()

Gets a list of IGMP snooping groups for VLANs or interfaces.

Syntax

`groupGet(`*vlan, interface***`)`**

where:

Parameter	Description
vlan	(Optional) The VLAN ID. An integer from 1-4093. Default value: `None`.
interface	(Optional) The Ethernet interface name (string). Default value: `None`.

Returns

The dictionary *igmpSnoopingGroupInfo* contains the following information:

Parameter	Description
vlan	The VLAN ID. An integer from 1-4093.
interface	The Ethernet interface name (string).
group	The IGMP group IPv4 address.
flags	The group flags (string). Valid values: `D` - for Dynamic, `S` - for Static.
expire	The source expiry time interval. A string in the following format: `“HH:MM:SS”` or `“stopped”`.
version	The IGMP version number (string). Valid values: `“V1”`, `“V2”`, `“V3”`.
filter_mode	The signal membership mode for IGMP V3 (string). Valid values: `”include”`, `”exclude”`.

The dictionaries *includeSourceInfo* and *excludeSourceInfo* contain the following information:

Parameter	Description
source	The included source IPv4 address.
uptime	The time since switch is running. A string in the following format: `“HH:MM:SS”`.
expire	The source expiry time interval. A string in the following format: `“HH:MM:SS”` or `“stopped”`.
forward	Whether to forward traffic for this source IP (string). Valid values: `”Yes”`, `”No”`.
flags	The source IP status (string). Valid values: `D` - for Dynamic, `S` - for Static.

mrouterGet()

Gets the IGMP snooping multicast router entries for the specified VLAN and/or interface.

Syntax

`mrouterGet``(`*vlan, interface*`)`

where:

Parameter	Description
vlan	(Optional) The VLAN ID. An integer from 1-4093. Default value: `None`.
interface	(Optional) Ethernet interface name (string). Default value: `None`.

Returns

A dictionary containing multicast router details:

Parameter	Description
vlan	The VLAN ID. An integer from 1-4093.
interface	The Ethernet interface name (string).
mrouter_ip	The IGMP multicast router IPv4 address (string).
mrouter_type	The multicast router entry type (string). Valid values: one of “`dynamic`”, “`static`”, “`PIM hello`”.
expires	The expiry time interval. A string in the following format: “`HH:MM:SS`”.

mrouterSet()

Adds or deletes the IGMP snooping multicast router entries for the specified VLAN and/or interface.

Syntax

`mrouterSet``(`*vlan, interface, add*`)`

where:

Parameter	Description
vlan	The VLAN ID. An integer from 1-4093.
interface	The Ethernet interface name (string).
add	The operation type (boolean). `True` to add an operation, `False` to delete it.

Returns

Boolean (`True` on success, otherwise `False`).

querierGet()

Gets IGMP querier information for the specified VLAN or for all VLANs.

Syntax

`querierGet``(`*vlan*`)`

where:

Parameter	Description
vlan	(Optional) The VLAN ID. If no VLAN is specified, the information for all VLANs is displayed. An integer from 1-4093. Default value: `None`.

Returns

A list of dictionaries containing IGMP snooping querier entries:

Parameter	Description
vlan	The VLAN ID. An integer from 1-4093.
querier_ip	The querier IPv4 address (string).
querier_state	The querier state (string). Valid values: `“Querier”`, `“Non-Querier”`.
version	The IGMP snooping version (integer). Valid values: `2`,`3`.
expires	The expiry time interval. A string in the following format: `“HH:MM:SS”`.

querierSet()

Sets or unsets IGMP snooping querier for the specified VLAN.

Syntax

`querierSet``(`*vlan, enable, address*`)`

where:

Parameter	**Description**
vlan	The VLAN ID. An integer from 1-4093.
enable	Whether IGMP snooping is enabled or disabled (boolean). Valid values: `True`, `False`.
address	(Optional) The IPv4 address used by the IGMP snooping querier (string). Default value: `None`. **Note:** Mandatory when enabling IGMP Snooping Querier.

Returns

Boolean (`True` on success, otherwise `False`).

IP Module

This module manages IP addresses.

To use this module, in the Python file or in the Python interpreter, enter:

```
import ipApi
```

class IP()

This class provides functions that manage IP addresses.

get_ipinfo()

Gets the IP properties of a specified interface.

Syntax

`get_ipinfo``(`*if_name*`)`

where:

Parameter	Description
if_name	(Optional) The interface name (string).

Returns

Multiple possible return values:

- `False` if an error occurred
- a dictionary containing the IP details:

Parameter	Description
ip_addr	IP address for the interface (string).
if_name	IP interface name. **Note:** The interface must exist.
switchport	Whether the port is a switch port (string). Valid values: “`yes`”, “`no`”. Default value: “`yes`”.
mtu	The maximum transmission unit, in bytes. An integer from 64-9216. Default value: `1500`.
vrf_name	The name of the VRF to which the interface belongs (if applicable).
admin_state	The admin status. Valid values: “`up`”, ”`down`”.
ip_prefix_len	IP address mask. A positive integer from 1-32.

set_ip_addr()

Sets the IP address of a specified interface.

Syntax

`set_ip_addr(`*if_name, ip_addr, secondary***`)`**

where:

Parameter	**Description**
if_name	IP interface name. **Note:** The interface must exist.
ip_addr	IP address for the interface.
secondary	Secondary value for an interface (integer).

Returns

Boolean (`True` on success, otherwise `False`).

set_bridge_port()

Makes the specified interface a bridge port.

Syntax

`set_bridge_port(`*if_name***`)`**

where:

Parameter	**Description**
if_name	IP interface name. **Note:** The interface must exist.

Returns

Boolean (`True` on success, otherwise `False`).

unset_bridge_port()

Changes the specified interface from a bridge port to a routed port.

Syntax

`unset_bridge_port``(`*if_name*`)`

where:

Parameter	**Description**
if_name	IP interface name (string). **Note:** The interface must exist.

Returns

Boolean (`True` on success, otherwise `False`).

set_if_flagup()

Sets the interface flag to make it operational.

Syntax

`set_if_flagup``(`*if_name*`)`

where:

Parameter	**Description**
if_name	IP interface name. **Note:** The interface must exist.

Returns

Boolean (`True` on success, otherwise `False`).

unset_if_flagup()

Unsets the interface flag to make it non-operational.

Syntax

`unset_if_flagup``(`*if_name*`)`

where:

Parameter	Description
if_name	IP interface name (string). **Note:** The interface must exist.

Returns

Boolean (`True` on success, otherwise `False`).

set_if_bgp_unnumbered()

Enables BGP unnumbered on a switch interface.

Syntax

`set_if_bgp_unnumbered``(`*if_name*`)`

where:

Parameter	Description
if_name	The name of the switch interface (string). For example: “`Ethernet1/12`”.

Returns

Boolean (`True` on success, otherwise `False`).

unset_if_bgp_unnumbered()

Disables BGP unnumbered on a switch interface.

Syntax

`unset_if_bgp_unnumbered`(*if_name*)

where:

Parameter	**Description**
if_name	The name of the switch interface (string). For example: “`Ethernet1/12`”.

Returns

Boolean (`True` on success, otherwise `False`).

L2F Module

This module manages the Layer 2 Failover (L2F) configuration on the switch.

To use this module, in the Python file or in the Python interpreter, enter:

```
import l2fApi
```

class L2F

This class provides functions to get and set L2F configurations.

python_get_l2f_info

Displays information about all enabled L2F profiles.

Note: The teaming global status must be enabled.

Syntax

```
python_get_l2f_info()
```

Returns

Multiple possible return values:

- a boolean with the value `false`
- a list of dictionary showing specified enabled L2F profile information:

Parameter	Description
profile_id	The L2F profile number. An integer from 1-200.
profile_status	The status of the L2F profile (string). Valid values: “`enabled`”, “`disabled`”.
limit_port_num	The limit of monitor ports for the L2F profile. An integer from 1-1024.
monitor_forward_num	The number of active monitor ports. An integer from 0 to the total configured limit of monitor ports.
monitor_interfaces	The list of monitor ports for the L2F profile. A list of monitor ports for the specified profile entry.
control_state	The status of the control port (string) Valid values: “`triggered`”, “`default`”.
control_interfaces	The list of control ports for the L2F profile. A list of control ports for the specified profile entry.
if_name	The name of the switch ethernet interface (string).
if_status	The status of the switch ethernet interface (string). Valid values: “`err-dis`”, “`admin-down`”, “`forward`”, “`non-forward`”.

Parameter	Description
agg_mem_table	The member ports of the LAG. A list of member interface for specific port-aggregation.
mem_if_name	The name of the switch port-aggregation interface (string).
mem_if_status	The status of the member interface of port-aggregation (string). Valid values: `“err-dis”`, `“admin-down”`, `“forward”`, `“non-forward”`.

python_get_l2f_id_info

Displays information about a specific L2F profile.

Note: The teaming global status must be enabled.

Syntax

`python_get_l2f_id_info``(`*profile_id*`)`

where:

Parameter	Description
profile_id	The L2F profile number. An integer from 1-200.

Returns

Multiple possible return values:

- a boolean with the value `false`
- a dictionary showing information about the specified L2F profile:

Parameter	Description
profile_id	The L2F profile number. An integer from 1-200.
profile_status	The status of the L2F profile (string). Valid values: `“enabled”`, `“disabled”`.
limit_port_num	The limit of monitor ports for the L2F profile. An integer from 1-1024.
monitor_forward_num	The number of active monitor ports. An integer from 0 to the total configured limit of monitor ports.
monitor_interfaces	The list of monitor ports for the L2F profile. A list of monitor ports for the specified profile entry.
control_state	The status of the control port (string) Valid values: `“triggered”`, `“default”`.
control_interfaces	The list of control ports for the L2F profile. A list of control ports for the specified profile entry.

Parameter	**Description**
if_name	The name of the switch ethernet interface (string).
if_status	The status of the switch ethernet interface (string). Valid values: `“err-dis”`, `“admin-down”`, `“forward”`, `“non-forward”`.
agg_mem_table	The member ports of the LAG. A list of member interface for specific port-aggregation.
mem_if_name	The name of the switch port-aggregation interface (string).
mem_if_state	The status of the member interface of port-aggregation (string). Valid values: `“err-dis”`, `“admin-down”`, `“forward”`, `“non-forward”`.

python_get_l2f_cfg

Displays information about all L2F profiles with non-default configurations.

Syntax

`python_get_l2f_cfg``()`

Returns

Multiple possible return values:

- a boolean with the value `false`
- a list of dictionary showing information of specified L2F profile with non-default configuration:

Parameter	**Description**
profile_id	The L2F profile number. An integer from 1-200.
profile_status	The status of the L2F profile (string). Valid values: `“enabled”`, `“disabled”`.
limit_port_num	The limit of monitor ports for the L2F profile. An integer from 1-1024.
monitor_interfaces	The list of monitor ports for the L2F profile. A list of monitor ports for the specified profile entry.
control_interfaces	The list of control ports for the L2F profile. A list of control ports for the specified profile entry.
if_name	The name of the switch ethernet interface (string).

python_get_l2f_id_cfg

Displays the settings of a specific L2F profile.

Syntax

`python_get_l2f_id_cfg``(`*profile_id*`)`

where:

Parameter	**Description**
profile_id	The L2F profile number. An integer from 1-200.

Returns

Multiple possible return values:

- a boolean with the value `false`
- a dictionary showing the settings of the specified L2F profile:

Parameter	**Description**
profile_id	The L2F profile number. An integer from 1-200.
profile_status	The status of the L2F profile (string). Valid values: `“enabled”`, `“disabled”`.
limit_port_num	The limit of monitor ports for the L2F profile. An integer from 1-1024.
monitor_interfaces	The list of monitor ports for the L2F profile. A list of monitor ports for the specified profile entry.
control_interfaces	The list of control ports for the L2F profile. A list of control ports for the specified profile entry.
if_name	The name of the switch ethernet interface (string).

python_set_l2f_id_mport_limit

Configures the limit of monitor ports for a specific L2F profile.

Syntax

`python_set_l2f_id_mport_limit``(`*profile_id*`,`*limit_port_num*`)`

where:

Parameter	**Description**
profile_id	The L2F profile number. An integer from 1-200.
limit_port_num	The limit of monitor ports for the L2F profile. An integer from 1-1024.

Returns

Boolean (`True` on success, otherwise `False`).

python_get_l2f_global_status

Displays the global status of L2F on the switch.

Syntax

`python_get_l2f_global_status``()`

Returns

Multiple possible return values:

- a boolean with the value `false`
- a dictionary showing the global status of L2F on the switch:

Parameter	**Description**
global_status	The global status of L2F (string). Valid values: `“enabled”`, `“disabled”`.

python_set_l2f_global_status

Configures the global status of L2F on the switch.

Syntax

`python_set_l2f_global_status``(`*global_status*`)`

where:

Parameter	Description
global_status	The global status of L2F (string). Valid values: `“enabled”`, `“disabled”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_l2f_id_profile_status

Sets the profile status for a specific L2F profile.

Syntax

`python_set_l2f_id_profile_status``(`*profile_id*`,`*profile_status*`)`

where:

Parameter	Description
profile_id	The L2F profile number. An integer from 1-200.
profile_status	The status of the specified profile entry (string). Valid values: `“enabled”`, `“disabled”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_del_l2f_id

Deletes an L2F profile.

Syntax

`python_del_l2f_id``(`*profile_id*`)`

where:

Parameter	Description
profile_id	The L2F profile number. An integer from 1-200.

Returns

Boolean (`True` on success, otherwise `False`).

python_update_l2f_interface

Adds or removes switch ports to or from the MMON monitor or control ports list for a specific L2F profile.

Syntax

`python_update_l2f_interface``(`*dict_l2f_if_oper_info*`)`

where *dict_l2f_if_oper_info* contains the following mandatory parameters:

Parameter	Description
operate_type	The type of operation (string). Valid values: “`add`”, “`del`”.
profile_id	The L2F profile number. An integer between 1-200.
port_type	The type of operated port (string). Valid values: “`monitor`”, “`control`”.
interfaces	The list of switch interface name to add or remove. A list of strings.

Returns

Boolean (`True` on success, otherwise `False`).

LACP Module

This module configures system and interface Link Aggregation Control Protocol (LACP).

To use this module, in the Python file or in the Python interpreter, enter:

```
import lacpApi
```

class LacpSystem()

This class provides methods to get and set an LACP system configuration.

python_lacp_get_sys_priority()

Gets LACP system priority.

Syntax

```
python_lacp_get_sys_priority()
```

Returns

The LACP system priority (integer).

python_lacp_get_max_bundle()

Gets the LACP max bundle, which is the supported maximum number of links for each LAG.

Syntax

```
python_lacp_get_max_bundle()
```

Returns

The supported maximum number of links per LAG (integer).

python_lacp_get_all_link_details()

Gets all LACP interface details.

Syntax

`python_lacp_get_all_link_details``()`

Returns

A list of dictionaries containing LACP link details, where:

Parameter	**Description**
if_name	Ethernet interface name (string). **Note:** The interface must exist.
lag_mode	LAG mode (string). Valid values: `“lacp_active”`, `”lacp_passive”`, `”no_lacp”`.
lacp_prio	LACP priority for the physical port. A positive integer from 1-65535. Default value: `32768`.
lacp_timeout	LACP timeout for the physical port (string). Valid values: `“short”`, `”long”`. Default value: `”long”`.

python_lacp_set_system()

Sets LACP system priority.

Syntax

`python_lacp_set_system``(`*sys_prio*`)`

where:

Parameter	**Description**
sys_prio	The interface system priority (integer).

Returns

Boolean (`True` on success, otherwise `False`).

LAG Module

This module configures LAGs and finds information about LAGs and associated interfaces.

To use this module, in the Python file or in the Python interpreter, enter:

`import lagApi`

class LAG()

This class provides functions to configure and get information about LAG.

python_get_lag()

Gets a list of all the LAG information for the device.

Syntax

`python_get_lag``()`

Returns

A list of LAG dictionaries containing LAG information for the device:

Parameter	**Description**
lag_name	LAG name (string).
lag_id	LAG identifier. A positive integer from 1-65535.
interfaces	A dictionary containing physical interface members of the LAG: ● `if_name` ● `lag_mode` ● `lacp_prio` ● `lacp_timeout`
suspend_ individual	If the LAG does not get the LACP BPUD from peer ports the port aggregation, the result is one of the following strings: ● `"Suspended"`: LACP on the ports is suspended rather than put into individual state ● `"Individual"`: LAG on the ports is put into individual state Default value: `"Suspended"`.
min_links	LACP minimum links number. An integer from 1-32. Default value: `1`.
if_name	Ethernet interface name (string). **Note:** The interface must exist.
lag_mode	LAG mode (string). Valid values: “`lacp_active`”, “`lacp_passive`”, “`no_lacp`”.

Parameter	Description
lacp_prio	LACP priority for the physical port. A positive integer from 1-65535. Default value: `32768`.
lacp_timeout	LACP timeout for the physical port (string). Valid values: `“short”`, `“long”`. Default value: `“long”`.

python_get_lag_id()

Gets a list of all the LAG information for the specified LAG ID.

Syntax

`python_get_lag``(`*lag_id*`)`

where:

Parameter	Description
lag_id	LAG identifier. A positive integer from 1-65535.

Returns

A dictionary containing LAG information for the device:

Parameter	Description
lag_name	LAG name (string).
lag_id	LAG identifier. A positive integer from 1-65535.
interfaces	A dictionary containing physical interface members of the LAG: ● `if_name` ● `lag_mode` ● `lacp_prio` ● `lacp_timeout` Up to 32 interfaces can be added.
suspend_ individual	If the LAG does not get the LACP BPUD from peer ports the port aggregation, the result is one of the following strings: ● `"Suspended"`: LACP on the ports is suspended rather than put into individual state ● `"Individual"`: LAG on the ports is put into individual state Default value: `"Suspended"`.
min_links	LACP minimum links number. An integer from 1-32. Default value: `1`.
if_name	Ethernet interface name (string). **Note:** The interface must exist.

Parameter	**Description**
lag_mode	LAG mode (string). Valid values: “`lacp_active`”, “`lacp_passive`”, “`no_lacp`”.
lacp_prio	LACP priority for the physical port. A positive integer from 1-65535. Default value: 32768.
lacp_timeout	LACP timeout for the physical port (string). Valid values: ”`short`”, ”`long`”. Default value: ”`long`”.

python_update_lag_id()

Updates LAG information for the specified LAG ID.

Syntax

`python_update_lag_id``(`*lag_id*`)`

where:

Parameter	**Description**
lag_id	LAG identifier. A positive integer from 1-65535.

Returns

A dictionary containing LAG information for the specified LAG ID:

Parameter	**Description**
lag_name	LAG name (string).
lag_id	LAG identifier. A positive integer from 1-65535.
interfaces	A list of interfaces associated with this LAG, containing: ● `if_name` ● `lag_mode` ● `lacp_prio` ● `lacp_timeout` Up to 32 interfaces can be added.
if_name	Ethernet interface name (string). **Note:** The interface must exist.
lag_mode	LAG mode (string); Valid values: “`lacp_active`”, “`lacp_passive`”, “`no_lacp`”.
lacp_prio	LACP priority for the physical port. A positive integer from 1-65535. Default value: 32768.

Parameter	Description
lacp_timeout	LACP timeout for the physical port (string). Valid values: `“short”`, `“long”`. Default value: `“long”`.
suspend_ individual	Whether the LACP state is suspended or individual (string). Valid values: `“Individual”`, `“Suspended”`, `”N/A”`.

python_update_lag_id_details()

Updates LAG information for the specified LAG ID.

Syntax

`python_update_lag_id_details``(`*lag*`)`

where:

Parameter	Description
lag_id	LAG identifier. A positive integer from 1-65535.
interfaces	A list of interfaces associated with this LAG, containing: ● `if_name` ● `lag_mode` ● `lacp_prio` ● `lacp_timeout` Up to 32 interfaces can be added.
if_name	Ethernet interface name (string). **Note:** The interface must exist.
lag_mode	LAG mode (string); Valid values: `”lacp_active”`, `“lacp_passive”`, `“no_lacp”`.
lacp_prio	LACP priority for the physical port. A positive integer from 1-65535. Default value: `32768`.
lacp_timeout	LACP timeout for the physical port (string); Valid values: `“short”`, `”long”`. Default value: `”long”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_create_lag_id()

Creates a new LAG with the information provided.

Syntax

`python_create_lag_id``(`*lag*`)`

where:

Parameter	Description
lag	A dictionary containing LAG information for the specified LAG.

This dictionary contains:

Parameter	Description
lag_id	LAG identifier. A positive integer from 1-65535.
interfaces	A list of interfaces associated with this LAG, each containing: ● `if_name` ● `lag_mode` ● `lacp_prio` ● `lacp_timeout` Up to 32 interfaces can be added.
if_name	Ethernet interface name (string). **Note:** The interface must exist.
lag_mode	LAG mode (string); Valid values: `“lacp_active”`, `“lacp_passive”`, `”no_lacp”`.
lacp_prio	LACP priority for the physical port. A positive integer from 1-65535. Default value: `32768`.
lacp_timeout	LACP timeout for the physical port (string); Valid values: `“short”`, `“long”`. Default value: `“long”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_delete_lag_all()

Deletes all LAGs on the device.

Syntax

`python_delete_lag_all``()`

Returns

Boolean (`True` on success, otherwise `False`).

python_delete_lag_id()

Syntax

`python_delete_lag_id``(`*lag_id*`)`

where:

Parameter	**Description**
lag_id	The LAG identifier. A positive integer from 1-65535.

Returns

Boolean (`True` on success, otherwise `False`).

class LoadBalance()

This class provides functions to get and set load balance methods for port aggregations.

get_load_balance_method()

Gets the load balance method for port aggregations.

Syntax

`get_load_balance_method``()`

Returns

A list of strings containing the load balance method. Valid values:

- `“destination-ip”`
- `“destination-mac”`
- `“destination-port”`
- `“source-dest-ip”`
- `“source-dest-mac”`
- `“source-dest-port”`
- `“source-interface”`
- `“source-ip”`
- `“source-mac”`
- `“source-port”`

Default value: `“source-dest-ip”`

reset_load_balance_method()

Resets the load balance method for port aggregations.

Syntax

`reset_load_balance_method``()`

Returns

Boolean (`True` on success, otherwise `False`).

set_load_balance_method()

Sets the load balance method for port aggregations.

Syntax

`set_load_balance_method``(`*lbm_str*`)`

where:

Parameter	Description
lbm_str	The load balance method (string): ● `“destination-ip”` ● `“destination-mac”` ● `“destination-port”` ● `“source-dest-ip”` ● `“source-dest-mac”` ● `“source-dest-port”` ● `“source-interface”` ● `“source-ip”` ● `“source-mac”` ● `“source-port”` Default value: `“source-dest-ip”`

Returns

Boolean (`True` on success, otherwise `False`).

LDAP Module

This module manages the Lightweight Directory Access Protocol (LDAP) configuration on the switch.

To use this module, in the Python file or in the Python interpreter, enter:

```
import ldapApi
```

class LDAP

The functions in this class get and set LDAP configurations.

get_ldap_status

Displays the status of LDAP on the switch.

Syntax

```
get_ldap_status()
```

Returns

A string showing the status of LDAP on the switch.
Valid values: `“enable”`, `“disable”`.

set_ldap_enabled

Enables LDAP on the switch.

Syntax

```
set_ldap_enabled()
```

Returns

Boolean (`True` on success, otherwise `False`).

set_ldap_disabled

Disables LDAP on the switch.

Syntax

```
set_ldap_disabled()
```

Returns

Boolean (`True` on success, otherwise `False`).

get_global_retransmit

Displays the global LDAP server retransmit configuration.

Syntax

`get_global_retransmit``()`

Returns

An integer showing the retransmit count, between 1 and 5, or `“not configured”`.

set_global_retransmit

Configures the global LDAP server retransmit count.

Syntax

`set_global_retransmit``(`*retransmit*`)`

where:

Parameter	**Description**
retransmit	The LDAP server retransmit count. An integer from 1-5.

Returns

Boolean (`True` on success, otherwise `False`).

get_global_timeout

Displays the global LDAP server connection timeout configuration.

Syntax

`get_global_timeout``()`

Returns

An integer showing the timeout period in seconds, or `“not configured”`.

set_global_timeout

Configures the global LDAP server connection timeout period.

Syntax

`set_global_timeout``(`*timeout*`)`

where:

Parameter	Description
timeout	The LDAP server timeout period, in seconds. An integer from 1-60.

Returns

Boolean (`True` on success, otherwise `False`).

get_global_authorization

Displays the global LDAP server authorization value.

Syntax

`get_global_authorization``()`

Returns

A string showing the LDAP authorization value, or “`not configured`”.

set_global_authorization

Configures the global LDAP server authorization method.

Syntax

`set_global_authorization``(`*authorization*`)`

where:

Parameter	Description
authorization	The LDAP global authorization method (string). Valid values: “`bitmap`”, “`rbac`”.

Returns

Boolean (`True` on success, otherwise `False`).

get_global_pki

Displays the configured global LDAP server Private Key Infrastructure (PKI) name.

Syntax

`get_global_pki``()`

Returns

A string showing the name of the LDAP PKI name, or `“not configured”`.

set_global_pki

Configures the LDAP global PKI name.

Note: The PKI must be already created before executing this function.

Syntax

`set_global_pki``(`*pki*`)`

where:

Parameter	**Description**
pki	The LDAP global PKI name (string).

Returns

Boolean (`True` on success, otherwise `False`).

get_profile

Displays information about an LDAP profile.

Syntax

`get_profile``(`*profile*`)`

where:

Parameter	**Description**
profile	The name of the LDAP profile (string).

Returns

A dictionary showing information about the specified LDAP profile:

Parameter	**Description**
profile_name	The name of the LDAP profile (string).
host	The IP address of the LDAP server (string). Valid values: “`IPv4`”, “`IPv6`”.
retransmit	The LDAP retransmit time. An integer from 1-5.
predefined_credential_dn	The Domain Name (DN) for binding with the LDAP server (string).
base_dn	The LDAP based DN (string).
port	The TCP port for sending messages to the LDAP server. An integer from 1-65535.
credential_key_form	The LDAP encryption/decryption key for the DN (integer). Valid values: `0`-clear text, `7`-encrypted.
group_filter	The filter when performing group searches (string).
attribute_group	The custom LDAP attribute group name (string).
authorization	The LDAP authorization method (string). Valid values: “`bitmap`”, “`rbac`”.
bind_mode	The LDAP binding method (string). Valid values: “`prompted`”, “`predefined`”.
pki_name	The name of the PKI profile used for LDAP (string).
predefined_credential_key	The password for the Domain Name (string).
attribute_username	The custom LDAP attribute user name (string).
attribute_permission_name	The custom LDAP attribute permission name (string).

Parameter	**Description**
attribute_permission_ deny_bitmap	The custom LDAP attribute permission denial bitmap (string).
attribute_permission_ oper_bitmap	The custom LDAP attribute permission operative bitmap (string).
attribute_permission_ admin_bitmap	The custom LDAP attribute permission administrative bitmap (string).
attribute_permission_ deny_role	The custom LDAP attribute permission deny role (string).
attribute_permission_ admin_role	The custom LDAP attribute permission administrator role (string).
attribute_permission_ oper_role	The custom LDAP attribute permission operator role (string).
timeout	The LDAP server connection timeout period, in seconds. An integer from 1-60.
security	The LDAP transmit mode and security option (string). Valid values: ● `“ldaps”` ● `“startTLS”` ● `“ldaps ignore”` ● `“startTLS ignore”` ● `“clear”`

get_all_profiles

Displays a list of all configured LDAP profiles.

Syntax

```
get_all_profiles()
```

Returns

A list of all LDAP profiles:

where:

Parameter	Description
profile_name	The name of the LDAP profile (string).
host	The IP address of the LDAP server (string). Valid values: `“IPv4”`, `“IPv6”`.
retransmit	The LDAP retransmit time. An integer from 1-5.
predefined_credential_ dn	The Domain Name (DN) for binding with the LDAP server (string).
base_dn	The LDAP based DN (string).
port	The TCP port for sending messages to the LDAP server. An integer from 1-65535.
credential_key_form	The LDAP encryption/decryption key for the DN (integer). Valid values: `0`-clear text, `7`-encrypted.
group_filter	The filter when performing group searches (string).
attribute_group	The custom LDAP attribute group name (string).
authorization	The LDAP authorization method (string). Valid values: `“bitmap”`, `“rbac”`.
bind_mode	The LDAP binding method (string). Valid values: `“prompted”`, `“predefined”`.
pki_name	The name of the PKI profile used for LDAP (string).
predefined_credential_ key	The password for the Domain Name (string).
attribute_username	The custom LDAP attribute user name (string).
attribute_permission_ name	The custom LDAP attribute permission name (string).
attribute_permission_ deny_bitmap	The custom LDAP attribute permission denial bitmap (string).
attribute_permission_ oper_bitmap	The custom LDAP attribute permission operative bitmap (string).

Parameter	**Description**
attribute_permission_ admin_bitmap	The custom LDAP attribute permission administrative bitmap (string).
attribute_permission_ deny_role	The custom LDAP attribute permission deny role (string).
attribute_permission_ admin_role	The custom LDAP attribute permission administrator role (string).
attribute_permission_ oper_role	The custom LDAP attribute permission operator role (string).
timeout	The LDAP server connection timeout period, in seconds. An integer from 1-60.
security	The LDAP transmit mode and security option (string). Valid values: ● `“ldaps”` ● `“startTLS”` ● `“ldaps ignore”` ● `“startTLS ignore”` ● `“clear”`

set_profile

Configures an LDAP profile.

Syntax

`set_profile``(`*profile_name*`,`*host*`,`*port*`,`*base_dn*`,`*bind_mode*`,`*security*`,`*retransmit*`,` *timeout*`,`*authorization*`,`*attr_group*`,`*attr_per_name*`,`*attr_per_admin_bitmap*`,` *attr_per_oper_bitmap*`,`*attr_per_deny_bitmap*`,`*attr_per_admin_role*`,`*attr_per_oper_role*`,` *attr_per_deny_role*`,`*attr_username*`,`*pre_cred_dn*`,`*pre_cred_key*`,`*cred_key_form*`,` *group_filter*`,`*pki_name*`)`

where:

Parameter	**Description**
profile_name	The name of the LDAP profile (string).
host	The IP address of the LDAP server (string). Valid values: “`IPv4`”, “`IPv6`”.
retransmit	The LDAP retransmit time. An integer from 1-5.
predefined_credential_dn	The Domain Name (DN) for binding with the LDAP server (string).
base_dn	The LDAP based DN (string).
port	The TCP port for sending messages to the LDAP server. An integer from 1-65535.
credential_key_form	The LDAP encryption/decryption key for the DN (integer). Valid values: `0`-clear text, `7`-encrypted.
group_filter	The filter when performing group searches (string).
attribute_group	The custom LDAP attribute group name (string).
authorization	The LDAP authorization method (string). Valid values: “`bitmap`”, “`rbac`”.
bind_mode	The LDAP binding method (string). Valid values: “`prompted`”, “`predefined`”.
pki_name	The name of the PKI profile used for LDAP (string).
predefined_credential_key	The password for the Domain Name (string).
attribute_username	The custom LDAP attribute user name (string).
attribute_permission_name	The custom LDAP attribute permission name (string).
attribute_permission_deny_bitmap	The custom LDAP attribute permission denial bitmap (string).
attribute_permission_oper_bitmap	The custom LDAP attribute permission operative bitmap (string).

Parameter	**Description**
attribute_permission_ admin_bitmap	The custom LDAP attribute permission administrative bitmap (string).
attribute_permission_ deny_role	The custom LDAP attribute permission deny role (string).
attribute_permission_ admin_role	The custom LDAP attribute permission administrator role (string).
attribute_permission_ oper_role	The custom LDAP attribute permission operator role (string).
timeout	The LDAP server connection timeout period, in seconds. An integer from 1-60.
security	The LDAP transmit mode and security option (string). Valid values: ● `“ldaps”` ● `“startTLS”` ● `“ldaps ignore”` ● `“startTLS ignore”` ● `“clear”`

Returns

Boolean (`True` on success, otherwise `False`).

delete_profile

Deletes an LDAP profile.

Syntax

`delete_profile``(`*profile_name*`)`

where:

Parameter	**Description**
profile_name	The name of the LDAP profile (string).

Returns

Boolean (`True` on success, otherwise `False`).

get_group

Displays information about an LDAP group.

Syntax

`get_group``(`*group_name*`)`

where:

Parameter	**Description**
group_name	The name of the LDAP group (string).

Returns

A dictionary showing LDAP group information:

Parameter	**Description**
group_name	The name of the LDAP group. A string up to 127 characters long.
vrf_name	The VRF instance for the LDAP group. A string up to 63 characters long.
profile_name	The name of the LDAP profile (string).
host	The IP address of the LDAP server (string). Valid values: `“IPv4”`, `“IPv6”`.
retransmit	The LDAP retransmit time. An integer from 1-5.
predefined_credential_dn	The Domain Name (DN) for binding with the LDAP server (string).
base_dn	The LDAP based DN (string).
port	The TCP port for sending messages to the LDAP server. An integer from 1-65535.
credential_key_form	The LDAP encryption/decryption key for the DN (integer). Valid values: `0`-clear text, `7`-encrypted.
group_filter	The filter when performing group searches (string).
attribute_group	The custom LDAP attribute group name (string).
authorization	The LDAP authorization method (string). Valid values: `“bitmap”`, `“rbac”`.
bind_mode	The LDAP binding method (string). Valid values: `“prompted”`, `“predefined”`.
pki_name	The name of the PKI profile used for LDAP (string).
predefined_credential_key	The password for the Domain Name (string).

Parameter	**Description**
attribute_username	The custom LDAP attribute username (string).
attribute_permission_ name	The custom LDAP attribute permission name (string).
attribute_permission_ deny_bitmap	The custom LDAP attribute permission denial bitmap (string).
attribute_permission_ oper_bitmap	The custom LDAP attribute permission operative bitmap (string).
attribute_permission_ admin_bitmap	The custom LDAP attribute permission administrative bitmap (string).
attribute_permission_ deny_role	The custom LDAP attribute permission deny role (string).
attribute_permission_ admin_role	The custom LDAP attribute permission administrator role (string).
attribute_permission_ oper_role	The custom LDAP attribute permission operator role (string).
timeout	The LDAP server connection timeout period, in seconds. An integer from 1-60.
security	The LDAP transmit mode and security option (string). Valid values: ● `“ldaps”` ● `“startTLS”` ● `“ldaps ignore”` ● `“startTLS ignore”` ● `“clear”`

get_all_groups

Displays all configured LDAP groups.

Syntax

```
get_all_groups()
```

Returns

A dictionary showing LDAP group information:

Parameter	**Description**
group_name	The name of the LDAP group. A string up to 127 characters long.
vrf_name	The VRF instance for the LDAP group. A string up to 63 characters long.
profile_name	The name of the LDAP profile (string).
host	The IP address of the LDAP server (string). Valid values: `“IPv4”`, `“IPv6”`.
retransmit	The LDAP retransmit time. An integer from 1-5.
predefined_credential_ dn	The Domain Name (DN) for binding with the LDAP server (string).
base_dn	The LDAP based DN (string).
port	The TCP port for sending messages to the LDAP server. An integer from 1-65535.
credential_key_form	The LDAP encryption/decryption key for the DN (integer). Valid values: `0`-clear text, `7`-encrypted.
group_filter	The filter when performing group searches (string).
attribute_group	The custom LDAP attribute group name (string).
authorization	The LDAP authorization method (string). Valid values: `“bitmap”`, `“rbac”`.
bind_mode	The LDAP binding method (string). Valid values: `“prompted”`, `“predefined”`.
pki_name	The name of the PKI profile used for LDAP (string).
predefined_credential_ key	The password for the Domain Name (string).
attribute_username	The custom LDAP attribute user name (string).
attribute_permission_ name	The custom LDAP attribute permission name (string).

Parameter	**Description**
attribute_permission_ deny_bitmap	The custom LDAP attribute permission denial bitmap (string).
attribute_permission_ oper_bitmap	The custom LDAP attribute permission operative bitmap (string).
attribute_permission_ admin_bitmap	The custom LDAP attribute permission administrative bitmap (string).
attribute_permission_ deny_role	The custom LDAP attribute permission deny role (string).
attribute_permission_ admin_role	The custom LDAP attribute permission administrator role (string).
attribute_permission_ oper_role	The custom LDAP attribute permission operator role (string).
timeout	The LDAP server connection timeout period, in seconds. An integer from 1-60.
security	The LDAP transmit mode and security option (string). Valid values: ● `“ldaps”` ● `“startTLS”` ● `“ldaps ignore”` ● `“startTLS ignore”` ● `“clear”`

add_group

Configures an LDAP group.

Syntax

`add_group``(`*group_name*`)`

where:

Parameter	**Description**
group_name	The name of the LDAP group (string).

Returns

Boolean (`True` on success, otherwise `False`).

add_server_to_group

Adds an LDAP profile to the LDAP group.

Syntax

`add_server_to_group``(`*group_name*`,`*profile_name*`)`

where:

Parameter	**Description**
group_name	The name of the LDAP group (string).
profile_name	The name of the LDAP profile (string).

Returns

Boolean (`True` on success, otherwise `False`).

set_group_vrf

Configures the VRF instance for an LDAP group.

Syntax

`set_group_vrf``(`*group_name*`,`*vrf_name*`)`

where:

Parameter	**Description**
group_name	The name of the LDAP group (string).
vrf_name	The VRF instance for the LDAP group (string).

Returns

Boolean (`True` on success, otherwise `False`).

delete_group

Deletes an LDAP group.

Syntax

`delete_group``(`*group_name*`)`

where:

Parameter	**Description**
group_name	The name of the LDAP group (string).

Returns

Boolean (`True` on success, otherwise `False`).

LFD Module

This module manages Link Flap Dampening (LFD).

To use this module, in the Python file or in the Python interpreter, enter:

`import lfdApi`

class LfdInfo()

This class provides functions to get LFD information.

get_lfd_feature()

Gets the LFD status.

Syntax

`get_lfd_feature``()`

Returns

A dictionary containing LFD details:

Parameter	Description
enable	The LFD status (string). Valid values: `"enabled"`, `"disabled"`.

set_lfd_feature()

Enables or disables LFD.

Syntax

`set_lfd_feature``(`*status, recovery*`)`

where:

Parameter	Description
status	The LFD status (boolean). Valid values: `True`, `False`.
recovery	The timer to recover from link-flap error disable state (boolean). Valid values: `True` - to enable recovery, `False` - to disable recovery.

Returns

Boolean (`True` on success, otherwise `False`).

get_lfd_if_info()

Gets information about LFD parameters for all interfaces or for a specified interface.

Syntax

`get_lfd_if_info``(`*if_name*`)`

where:

Parameter	**Description**
if_name	(Optional) The interface name (string). **Note:** If no interface is specified, the configuration is applied on all interfaces. Default value: `None`.

Returns

A dictionary containing LFD details:

Parameter	**Description**
interface	The interface name (string).
enable	The LFD status (string). Valid values: `"enabled"`, `"disabled"`.
max_flaps	The number of flaps after which interface is shut down. An integer from 0-100. Default value: `0`.
window	The time range for an interface to reach the maximum number of link flaps. After reaching the maximum value, the interface enters error disable state. An integer from 5-500. Default value: `10`.
err_disable	The error disable state of the interface (boolean). Valid values: `True`, `False`.

set_lfd_if_info()

Sets LFD on a specified interface or on all interfaces.

Syntax

`set_lfd_if_info``(`*if_name, status, flaps, window*`)`

where:

Parameter	**Description**
if_name	The interface name (string). **Note:** If no interface is specified, the configuration is applied on all interfaces. Default value: `None`.
status	Enables or disables LFD on an interface (boolean). Valid values: `True`, `False`. Default value: `None`.
flaps	The number of flaps that can occur until the interface enters error disable state. An integer from 0-100. Default value: `None`.
window	The time range for an interface to reach the maximum number of link flaps. After reaching the maximum value, the interface enters error disable state. An integer from 5-500. Default value: `None`.

Returns

Boolean (`True` on success, otherwise `False`).

LLDP Module

This module manages Link Layer Discovery Protocol (LLDP) configurations and LLDP interface configurations, gets LLDP statistics and LLDP neighbor information.

To use this module, in the Python file or in the Python interpreter, enter:

```
import lldpApi
```

class LldpSystem()

This class provides functions to get and set LLDP configurations and LLDP interface configurations.

python_lldp_get_reinit_delay()

Gets the number of seconds until LLDP re-initialization is attempted on an interface.

Syntax

```
python_lldp_get_reinit_delay()
```

Returns

The delay value:

where:

Parameter	Description
reinit_delay	The LLDP re-initialization delay. An integer from 1-10. Default value: 2.

python_lldp_get_msg_tx_interval()

Gets the time interval in seconds between transmissions of LLDP messages.

Syntax

```
python_lldp_get_msg_tx_interval()
```

Returns

The transmit interval value:

where:

Parameter	Description
tx_interval	The LLDP transmit interval. An integer from 5-32768. Default value: 30.

python_lldp_get_tx_delay()

Gets the number of seconds for LLDP transmission delay.

Syntax

`python_lldp_get_tx_delay``()`

Returns

The transmit delay value:

where:

Parameter	Description
tx_delay	The LLDP transmit delay. An integer from 1-8192. Default value: `2`.

python_lldp_set_reinit_delay()

Sets the time to wait, in seconds, before initializing an interface.

Syntax

`python_lldp_set_reinit_delay``(`*reinit_delay*`)`

where:

Parameter	Description
reinit_delay	The LLDP re-initialization delay. An integer from 1-10. Default value: `2`.

Returns

Boolean (`True` on success, otherwise `False`).

python_lldp_set_msg_tx_interval()

Sets the rate, in seconds, for LLDP packets to be sent.

Syntax

`python_lldp_set_msg_tx_interval``(`*tx_interval*`)`

where:

Parameter	Description
tx_interval	The LLDP transmit interval. An integer from 5-32768. Default value: `30`.

Returns

Boolean (`True` on success, otherwise `False`).

python_lldp_set_tx_delay()

Sets the delay time in seconds.

Syntax

`python_lldp_set_tx_delay(``*tx_delay*`**`)`**

where:

Parameter	**Description**
tx_delay	The transmission delay, in seconds. An integer from 1-8192. Default value: `2`. **Note:** This value must not be greater than 25% of the transmit interval value.

Returns

Boolean (`True` on success, otherwise `False`).

class LldpNeighbor()

This class provides functions to get neighbor port information.

python_lldp_get_neighbor()

Gets neighbor information of a port.

Syntax

`python_lldp_get_neighbor``(`*ifname*`)`

where:

Parameter	**Description**
ifname	The interface port name (string).

Returns

A dictionary containing information about the specified port.

Parameter	**Description**
if_name	The ethernet interface name (string).
capability	Contains a bitmap indicating the primary functions implemented for the system. A 16-bit integer. Valid values: ● `1` - Other ● `2` - Repeater ● `3` - MAC Bridge ● `4` - WLAN Access Point ● `5` - Router ● `6` - Telephone ● `7` - DOCSIS cable device ● `8` - Station Only ● `9` - C-VLAN Component of a VLAN Bridge, ● `10` - S-VLAN Component of a VLAN Bridge ● `11` - Two-port MAC Relay (TPMR) ● `12-16` - reserved
rx ttl	The receiving time-to-live (TTL) value (Long). An integer.
system name	The TLV that allows network management to advertise the system's assigned name (string).

Parameter	Description
system description	The TLV that allows network management to advertise the system's description.An alpha-numeric string that includes the full name and version identification of the system's hardware type, software operating system, and networking software.
system mac	The MAC address associated with this instance of the agent (string). Valid values: the switch base MAC address.

python_lldp_get_all_neighbor()

Gets neighbor information of all ports.

Syntax

`python_lldp_get_all_neighbor``()`

Returns

A list of dictionaries containing information about all LLDP neighbor ports:

Parameter	Description
if_name	The ethernet interface name (string).
capability	Contains a bitmap indicating the primary functions implemented for the system. A 16-bit integer. Valid values: ● `1` - Other ● `2` - Repeater ● `3` - MAC Bridge ● `4` - WLAN Access Point ● `5` - Router ● `6` - Telephone ● `7` - DOCSIS cable device ● `8` - Station Only ● `9` - C-VLAN Component of a VLAN Bridge, ● `10` - S-VLAN Component of a VLAN Bridge ● `11` - Two-port MAC Relay (TPMR) ● `12-16` - reserved
rx ttl	The receiving time-to-live (TTL) value (Long). An integer.
system name	The TLV that allows network management to advertise the system's assigned name (string).

Parameter	Description
system description	The TLV that allows network management to advertise the system's description.An alpha-numeric string that includes the full name and version identification of the system's hardware type, software operating system, and networking software.
system mac	The MAC address associated with this instance of the agent (string). Valid values: the switch base MAC address.

class LldpStats

This class gets LLDP statistics.

python_lldp_get_statistics()

Gets LLDP port statistics.

Syntax

`python_lldp_get_statistics``(`*ifname*`)`

where:

Parameter	Description
ifname	The interface port name (string).

Returns

A dictionary of LLDP statistics:

Parameter	Description
total frames	The total number of LLDP transmitted and received frames (integer).
total tlvs discarded	The total number of LLDP TLVs discarded (integer).
total frames transmitted	The total number of LLDP frames transmitted (integer).
total errrored frames	The total number of frames received with errors (integer).
total frames discarded	The total number of discarded frames (integer).
total entries aged	The total number of aged-out entries (integer).
total tlvs unrecognized	The total number of unrecognized LLDP TLVs (integer).

class LldpInterface()

This class provides functions to get and set LLDP interface information.

python_lldp_get_interface()

Gets LLDP interface admin status of a specific interface.

Syntax

`python_lldp_get_interface``(`*ifname*`)`

where:

Parameter	Description
ifname	The interface port name (string).

Returns

A dictionary of LLDP status information for the specified interface:

Parameter	Description
if_name	The ethernet interface name (string).
ena_lldp_rx	Whether the LLDP receive status is enabled on a given interface (string). Valid values: “`yes`”, “`no`”.
ena_lldp_tx	Whether LLDP frame transmission is enabled on a given interface (string). Valid values: “`yes`”, “`no`”.

python_lldp_get_all_interface()

Gets LLDP interface admin status for all interfaces.

Syntax

`python_lldp_get_all_interface``()`

Returns

A list of dictionaries containing LLDP status information for all interfaces:

Parameter	Description
if_name	The ethernet interface name (string).
ena_lldp_rx	Whether the LLDP receive status is enabled on a given interface (string). Valid values: “`yes`”, “`no`”.
ena_lldp_tx	Whether LLDP frame transmission is enabled on a given interface (string). Valid values: “`yes`”, “`no`”.

python_lldp_set_interface()

Sets LLDP interface admin status based on the lldp_interface_admin_status dictionary values.

Syntax

`python_lldp_set_interface``(`*lldp_interface_port_status*`)`

where:

Parameter	Description
lldp_interface_port_status	The LLDP interface status (dictionary) containing: ● `if_name` ● `ena_lldp_rx` ● `ena_lldp_tx`
if_name	The ethernet interface name (string).
ena_lldp_rx	Whether the LLDP receive status is enabled on a given interface (string). Valid values: `“yes”`, `“no”`.
ena_lldp_tx	Whether LLDP frame transmission is enabled on a given interface (string). Valid values: `“yes”`, `“no”`.

Returns

Boolean (`True` on success, otherwise `False`).

class LldpTlv()

This class provides functions to get and set LLDP TLVs.

get_lldp_tlv()

Gets LLDP TLV selections for a specific interface or for all interfaces.

Syntax

`get_lldp_tlv``(`*if_name*`)`

where:

Parameter	**Description**
if_name	(Optional) The ethernet interface name (string).

Returns

A dictionary or list of dictionaries of LLDP options:

Parameter	**Description**
link-aggregation	The Link Aggregation TLV informs the remote port whether or not the sending port believes it is currently in a link aggregation (string). Valid values: `“yes”`, `“no”`.
mac-phy-status	The MAC/PHY Configuration/Status is an optional TLV that identifies the following: • the duplex and bit-rate capability of the sending node that is connected to the physical medium • the current duplex and bit-rate settings of the sending node • whether these settings are the results of auto-negotiation during link initiation or of manual set override action Valid values: `“yes”`, `“no”`.
management-address	The most appropriate address for management use. If no management address is available, the MAC address for the station or port will be used (string). Valid values: `“yes”`, `“no”`.
max-frame-size	Maximum Frame Size TLV (string). Detects mis-configurations or incompatibility between two stations with different maximum supported frame sizes (string). Valid values: `“yes”`, `“no”`.
port-description	Port Description TLV. Allows network management to advertise the station’s port description (string). Valid values: `“yes”`, `“no”`.

Parameter	**Description**
port-protocol-vlan	Port and Protocol VLAN ID TLV. Optional TLV that allows a bridge port to advertise a port and protocol (string). Valid values: `“yes”`, `“no”`.
port-vlan	Port VLAN ID TLV. Optional fixed length TLV that allows a VLAN bridge to advertise the port VLAN identifier (PVID) that is associated with untagged or priority tagged frames. (string). Valid values: `“yes”`, `“no”`.
power-mdi	Power Via MDI TLV. Allows network management to advertise and discover the MDI power support capabilities of the sending station (string). Valid values: `“yes”`, `“no”`.
protocol-identity	Protocol Identity TLV. Optional TLV that allows a station to advertise particular protocols that are accessible through the port (string). Valid values: `“yes”`, `“no”`.
system-capabilities	System Capabilities TLV (string). Optional TLV that identifies the primary function(s) of the system and whether or not these functions are enabled. Valid values: `“yes”`, `“no”`.
system-description	Allows network management to advertise the system’s description (string). Valid values: `“yes”`, `“no”`.
system-name	Allows network management to advertise the system’s assigned name (string). Valid values: `“yes”`, `“no”`.
vid-management	Optional TLV that allows a station to advertise the value of a Management VID associated with the system (string). Valid values: `“yes”`, `“no”`.
vlan-name	VLAN Name TLV. Optional TLV that allows a station to advertise the assigned name of any VLAN with which it is configured (string). Valid values: `“yes”`, `“no”`.

reset_lldp_tlv()

Resets the LLDP TLV selection for the specified interface.

Syntax

`reset_lldp_tlv``(`*ifname, tlv_str*`)`

where:

Parameter	Description
ifname	The ethernet interface name (string).
tlv_str	The name of the TLV to be enabled (string). Valid values: ● `“link-aggregation”` ● `“mac-phy-status”` ● `“management-address”` ● `“max-frame-size”` ● `“port-description”` ● `“port-protocol-vlan”` ● `“port-vlan”` ● `“power-mdi”` ● `“protocol-identity”` ● `“system-capabilities”` ● `“system-description”` ● `“system-name”` ● `“vid-management”` ● `“vlan-name”`

Returns

Boolean (`True` on success, otherwise `False`).

set_lldp_tlv()

Sets the LLDP TLV selection for the specified interface.

Syntax

`set_lldp_tlv``(`*if_name, tlv_str*`)`

where:

Parameter	**Description**
if_name	The ethernet interface name (string).
tlv_str	The name of the TLV to be enabled (string). Valid values: ● `“link-aggregation”` ● `“mac-phy-status”` ● `“management-address”` ● `“max-frame-size”` ● `“port-description”` ● `“port-protocol-vlan”` ● `“port-vlan”` ● `“power-mdi”` ● `“protocol-identity”` ● `“system-capabilities”` ● `“system-description”` ● `“system-name”` ● `“vid-management”` ● `“vlan-name”`

Returns

Boolean (`True` on success, otherwise `False`).

MoveNotify Module

This module manages MAC move notifications.

To use this module, in the Python file or in the Python interpreter, enter:

```
import moveNotifyApi
```

class MOVE_NOTIFY()

This class provides functions to manage MAC move notifications.

python_get_notify_info()

Gets all MAC Move Notification information.

Syntax

```
python_get_notify_info()
```

Returns

A dictionary containing MAC move notifications:

Parameter	Description
mac_move_notification	The MAC address-table notification configuration if MAC move notification is disabled (string). Valid values: “`no`”.

or:

Parameter	Description
mac_move_notify_triggers	All MAC move notify triggers after switch boot if MAC move notification is enabled. An integer from 0-4294967295.

python_get_notify_cfg()

Gets all MAC Move Notification settings.

Syntax

`python_get_notify_cfg``()`

Returns

A dictionary containing MAC move notifications:

Parameter	**Description**
mac_move_notification	Enable or disable MAC address-table notification mac-move (string). Valid values: `“yes”`, `“no”`.

python_set_move_notify()

Sets MAC address-table notification.

Syntax

`python_set_move_notify``(`*status*`)`

where:

Parameter	**Description**
status	Enable or disable MAC address-table notification mac-move (string). Valid values: `“yes”`, `“no”`.

Returns

Boolean (`True` on success, otherwise `False`).

MpLogging Module

This module manages Mp Packet Logging.

To use this module, in the Python file or in the Python interpreter, enter:

`import mpLoggingApi`

class MpLogging()

This class provides functions to get and set Mp Packet Logging.

get_logs()

Gets all Mp Packet Logging information.

Syntax

`get_logs``()`

Returns

A dictionary containing Mp Packet Logging information:

Parameter	**Description**
direction	The packets received or sent by CPU (string). Valid values: `"rx"`, `"tx"`. Default value: `None`.
src_port	The source port (integer). Valid value: `1 - N`. **Note:** The maximum port number depends on the platform. Default value: `None`.
dst_port	The destination port (integer). Valid value: `1 - N`. **Note:** The maximum port number depends on the platform. Default value: `None`.
src_mac_l2	The MAC address of the source (string). Default value: `None`.
dst_mac_l2	The MAC address of the destination (string). Default value: `None`.
sender_ip	The IPv4 or IPv6 address of the sender (string). Default value: `None`.
target_ip	The IPv4 or IPv6 address of the destination (string). Default value: `None`.
length	The packet size (integer). Default value: `None`.
time	The time when packet was received/sent. A string in the following format: `"YYYY/MM/DD-HH:MM:SS"`. Default value: `None`.
interface	The interface name (string). Default value: `None`.

Parameter	**Description**
type	The packet type (string). Valid values: • `"ACL_LOG"` • `"ARP"` • `"CDCP"` • `"CFM"` • `"DHCP"` • `"EAPOL"` • `"ECP"` • `"EFM"` • `"FIPS"` • `"GARP"` • `"IGMP"` • `"IPv4 ICMP"` • `"IPv4 Oth"` • `"IPv4 TCP"` • `"IPv4 UDP"` • `"IPv6 ICMP"` • `"IPv6 Oth"` • `"IPv6 TCP"` • `"IPv6 UDP"` • `"LACP"` • `"LLDP"` • `"MLD"` • `"Unknown"` • `"PTP"` • `"SFLOW"` • `"SLP"` • `"STP"` • `"Total"` • `"UFP"` • `"VRRP"` • `"VXLAN"` Default value: `None`.
vlan	The VLAN ID. An integer from 1-4093.
sender_mac_l3	The MAC address of an L3 interface on the sender unit (string). Default value: `None`.
target_mac_l3	The MAC address of an L3 interface on the target unit (string). Default value: `None`.

get_counters()

Gets the counters registered by Mp Packet Logging.

Syntax

get_counters()

Returns

A dictionary containing Mp Packet Logging information:

Parameter	Description
protocol	The packet type (string). Valid values: • `"ACL_LOG"` • `"ARP"` • `"CDCP"` • `"CFM"` • `"DHCP"` • `"EAPOL"` • `"ECP"` • `"EFM"` • `"FIPS"` • `"GARP"` • `"IGMP"` • `"IPv4 ICMP"` • `"IPv4 Oth"` • `"IPv4 TCP"` • `"IPv4 UDP"` • `"IPv6 ICMP"` • `"IPv6 Oth"` • `"IPv6 TCP"` • `"IPv6 UDP"` • `"LACP"` • `"LLDP"` • `"MLD"` • `"Unknown"` • `"PTP"` • `"SFLOW"` • `"SLP"` • `"STP"` • `"Total"` • `"UFP"` • `"VRRP"` • `"VXLAN"`

Parameter	Description
received	The number of received packets from the specified type (integer). Default value: `0`.
sent	The number of sent packets from the specified type (integer). Default value: `0`.

clear_logs()

Clears logs registered by MP Packet Logging.

Syntax

`clear_logs``()`

Returns

Boolean (`True` on success, otherwise `False`).

clear_counters()

Clears counters registered by MP Packet Logging.

Syntax

`clear_counters``()`

Returns

Boolean (`True` on success, otherwise `False`).

MSTP Module

This module manages Multiple Spanning Tree Protocol (MSTP).

To use this module, in the Python file or in the Python interpreter, enter:

`import mstpApi`

class StpInterface()

This class provides functions to get and set interface STP properties.

python_stp_get_all_interface_info()

Syntax

`python_stp_get_all_interface_info``()`

Returns

A list of dictionaries containing the following STP information:

Parameter	**Description**
if_name	Interface name (string). **Note:** The interface must exist.
edge_port	The edge port. Usually a port that does not connect to a bridge is configured as edge port (string). Valid values: `“yes”`, `“no”`.
bpdu_guard	Enables BPDU guard on a port, which automatically shuts down the interface upon receipt of a BPDU (string). Valid values: `“enable”`, `“disable”`.

python_stp_get_interface_info()

Gets interface STP information.

Syntax

`python_stp_get_interface_info``(`*ifname*`)`

where:

Parameter	Description
ifname	The name of the interface (string).

Returns

A dictionary containing the following values:

Parameter	Description
if_name	Interface name (string). **Note:** The interface must exist.
edge_port	The edge port. Usually a port that does not connect to a bridge is configured as edge port (string). Valid values: `“yes”`, `“no”`.
bpdu_guard	Enables BPDU guard on a port, which automatically shuts down the interface upon receipt of a BPDU (string). Valid values: `“enable”`, `“disable”`.

python_stp_set_bpduguard()

Enables or disables STP BPDU Guard for a specified interface.

Syntax

`python_stp_set_bpduguard``(`*ifname, bpdu_guard_string*`)`

where:

Parameter	Description
ifname	The name of the interface (string).
bpdu_guard_string	The BPDU Guard status. The BPDU Guard automatically shuts down the interface upon receipt of a BPDU (string). Valid values: `“enable”`, `“disable”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_stp_set_edge_type()

Sets or resets the STP interface as an edge port.

Syntax

`python_stp_set_edge_type``(`*ifname, status_string*`)`

where:

Parameter	**Description**
ifname	The name of the interface (string).
status_string	The status of the edge port (string). Valid values: `“yes”`, `“no”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_mstp_set_port_loopguard()

Sets or resets STP loop-guard for a specified interface.

Syntax

`python_mstp_set_port_loopguard``(`*ifname, status_string*`)`

where:

Parameter	**Description**
ifname	The name of the interface (string).
status_string	The status of the loop guard (string). Valid values: `“enable”`, `“disable”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_mstp_set_port_rootguard()

Sets or resets STP root-guard for a specified interface.

Syntax

`python_mstp_set_port_rootguard``(`*ifname, status_string*`)`

where:

Parameter	**Description**
ifname	The name of the interface (string).
status_string	The status of the root guard (string). Valid values: `“enable”`, `“disable”`.

Returns

Boolean (`True` on success, otherwise `False`).

class MSTP()

This class provides functions to get and set the MSTP region name.

python_mstp_set_region_name()

Sets the MSTP region name.

Syntax

`python_mstp_set_region_name``(`*region_name*`)`

where:

Parameter	**Description**
region_name	The region name. A string up to 32 characters long.

Returns

Boolean (`True` on success, otherwise `False`).

python_mstp_get_region_name()

Gets the MSTP region name.

Syntax

`python_mstp_get_region_name``()`

Returns

The region name. A string up to 32 characters long.

python_mstp_get_revision()

Gets the revision number for the MSTP bridge.

Syntax

`python_mstp_get_revision``()`

Returns

A dictionary containing revision information:

Parameter	**Description**
revision	The MSTP revision number. An integer from 0-65535.

python_mstp_set_revision()

Sets the revision number for the MSTP bridge.

Syntax

`python_mstp_set_revision``(`*revision*`)`

where:

Parameter	**Description**
revision	The MSTP revision number. An integer from 0-65535.

Returns

Boolean (`True` on success, otherwise `False`).

class MstpInstance()

This class provides functions that control MSTP instances.

python_mstp_add_instance()

Adds an MSTP instance.

Syntax

`python_mstp_add_instance``(`*instance_id, vlan_list*`)`

where:

Parameter	**Description**
instance_id	The instance ID. An integer from 0-64. Instance 0 refers to the CIST.
vlan_list	A list of dictionaries containing VLAN numbers. An integer from 1-4093.

Returns

Boolean (`True` on success, otherwise `False`).

python_mstp_delete_instance()

Deletes an MSTP instance.

Syntax

`python_mstp_delete_instance``(`*instance_id*`)`

where:

Parameter	**Description**
instance_id	(Optional) The instance ID. An integer from 0-64. Instance 0 refers to the CIST. If no arguments are given, all user-created MSTP instances are deleted.

Returns

Boolean (`True` on success, otherwise `False`).

python_mstp_update_instance()

Updates MSTP instance configurations.

Syntax

`python_mstp_update_instance``(`*instance_id, vlan_list*`)`

where:

Parameter	**Description**
instance_id	The instance ID. An integer from 0-64. Instance 0 refers to the CIST.
vlan_list	A list of dictionaries containing VLAN numbers. Each VLAN number must be an integer from 1-4093.

Returns

Boolean (`True` on success, otherwise `False`).

python_mstp_set_instance_priority()

Sets MSTP instance priority.

Syntax

`python_mstp_set_instance_priority``(`*instance_id, instance_prio*`)`

where:

Parameter	**Description**
instance_id	The instance ID. An integer from 0-64. Instance 0 refers to the CIST.
instance_prio	Sets the instance bridge priority. An integer from 0-61440.

Returns

Boolean (`True` on success, otherwise `False`).

python_mstp_check_instance_exist()

Checks whether the specified MSTP instance exists.

Syntax

`python_mstp_check_instance_exist``(`*instance_id*`)`

where:

Parameter	Description
instance_id	The MSTP instance ID. An integer from 0-64. Instance 0 refers to the CIST.

Returns

If the instance exists, returns `True`. If the instance does not exist, returns `False`. If the instance ID is invalid, returns an error along with `False`.

class MstpInterface()

This class provides functions to get and set MSTP interface properties.

python_mstp_get_port_path_cost()

Gets the MSTP interface path cost.

Syntax

`python_mstp_get_port_path_cost``(`*ifname, instance_id*`)`

where:

Parameter	Description
ifname	The name of the interface (string).
instance_id	The instance ID. An integer from 0-64. Instance 0 refers to the CIST.

Returns

The MSTP path cost. An integer from 1-200000000.

python_mstp_set_port_path_cost()

Sets the MSTP interface path cost.

Syntax

`python_mstp_set_port_path_cost``(`*ifname, instance_id, path_cost*`)`

where:

Parameter	**Description**
ifname	The name of the interface (string). **Note:** The interface must exist.
instance_id	The instance ID. An integer from 0-64. Instance 0 refers to the CIST.
path_cost	The port path-cost value on the specified instance. An integer from 1-200000000.

Returns

Boolean (`True` on success, otherwise `False`).

python_mstp_get_port_priority()

Gets the MSTP interface port priority.

Syntax

`python_mstp_get_port_priority``(`*ifname, instance_id*`)`

where:

Parameter	**Description**
ifname	The name of the interface (string). **Note:** The interface must exist.
instance_id	The instance ID. An integer from 0-64. Instance 0 refers to the CIST.

Returns

The port priority (integer):

Parameter	**Description**
port_prio	The port priority, in increments of 32, on the specified instance. A multiple of 32 from 0-224.

python_mstp_set_port_priority()

Sets the MSTP interface port priority.

Syntax

`python_mstp_set_port_priority``(`*ifname, instance_id, port_prio*`)`

where:

Parameter	**Description**
ifname	The name of the interface (string). **Note:** The interface must exist.
instance_id	The instance ID. An integer from 0-64. Instance 0 refers to the CIST.
port_prio	The port priority, in increments of 32, on the specified instance. A multiple of 32 from 0-224.

Returns

Boolean (`True` on success, otherwise `False`).

NAT Module

This module manages Network Address Translation (NAT).

To use this module, in the Python file or in the Python interpreter, enter:

```
import natApi
```

class NAT()

This class provides functions to manage NAT.

get_nat_rules()

Gets all NAT rules.

Syntax

```
get_nat_rules()
```

Returns

A dictionary containing IP address status details:

Parameter	**Description**
rule_id	The ID of a NAT rule (integer).
loc_addr	The NAT Local IPv4 address (string).
glb_addr	The NAT Global IPv4 address (string).
loc_l4_port	The L4 Local NAT port (integer).
glb_l4_port	The L4 Global NAT port (integer).
proto	The L4 protocol (string). Valid values: `"udp"`, `"tcp"`.
vrf_name	The VRF name (string). Default value: `“default”`.

class NAT_CFG()

This class provides functions to configure NAT.

nat_rule_add()

Gets all NAT rules.

Syntax

`nat_rule_add``(`*loc_addr, glb_addr, loc_l4_port, glb_l4_port, proto, vrf_name*`)`

where:

Parameter	**Description**
loc_addr	The NAT Local IPv4 address (string).
glb_addr	The NAT Global IPv4 address (string).
loc_l4_port	(Optional) The L4 Local NAT port (integer).
glb_l4_port	(Optional) The L4 Global NAT port (integer).
proto	(Optional) The L4 protocol (string). Valid values: `"udp"`, `"tcp"`.
vrf_name	(Optional) The VRF name (string). Default value: `“default”`.

Returns

Boolean (`True` on success, otherwise `False`).

nat_rule_del)

Deletes a NAT rule.

Syntax

`nat_rule_del``(`*rule_id*`)`

where:

Parameter	**Description**
rule_id	The NAT rule ID (integer).

Returns

Boolean (`True` on success, otherwise `False`).

class NAT_IF_CFG()

This class provides functions to configure NAT interface options.

nat_realm_set()

Sets a NAT realm for a specified interface.

Syntax

`nat_realm_set``(`*if_name, realm_name*`)`

where:

Parameter	**Description**
if_name	The interface name (string).
realm_name	The name of NAT realm to be set for the given interface (string). Valid values: `"inside"`, `"outside"`.

Returns

Boolean (`True` on success, otherwise `False`).

Nexthop Health Check

This module manages Nexthop health check.

To use this module, in the Python file or in the Python interpreter, enter:

`import nhopHealthcheckApi`

class NexthopHealthCheck()

This class provides functions that manage nexthop health check.

set_nexthop_healthcheck_interval()

Sets nexthop health check interval.

Syntax

`set_nexthop_healthcheck_interval``(`*interval*`)`

where:

Parameter	Description
interval	The value of nexthop health check interval, in seconds. An integer from 5-60.

Returns

Boolean (`True` on success, otherwise `False`).

unset_nexthop_healthcheck()

Disables nexthop health check interval.

Syntax

`unset_nexthop_healthcheck``()`

Returns

Boolean (`True` on success, otherwise `False`).

NTP Authentication Module

This module manages Network Type Protocol (NTP) authentication.

To use this module, in the Python file or in the Python interpreter, enter:

```
import ntpApi
```

class NtpSystem()

This class provides functions to manage NTP system information.

python_ntp_get_enable()

Gets the NTP status.

Syntax

```
python_ntp_get_enable()
```

Returns

A dictionary containing the NTP status:

Parameter	Description
status	The NTP module status (string). Valid values: `“Enable”,”Disable”`.

python_ntp_set_enable()

Enables the NTP module.

Syntax

```
python_ntp_set_enable()
```

Returns

Boolean (`True` on success, otherwise `False`).

python_ntp_unset_enable()

Disables the NTP module.

Syntax

```
python_ntp_unset_enable()
```

Returns

Boolean (`True` on success, otherwise `False`).

python_ntp_get_auth_status()

Gets the NTP authentication status.

Syntax

`python_ntp_get_auth_status``()`

Returns

A dictionary containing the NTP status:

Parameter	**Description**
status	The NTP status (string). Valid values: `“Enable”`, `”Disable”`.

python_ntp_set_auth_status()

Sets the NTP authentication mode.

Syntax

`python_ntp_set_auth_status``()`

Returns

Boolean (`True` on success, otherwise `False`).

python_ntp_unset_auth_status()

Unsets the NTP authentication mode.

Syntax

`python_ntp_unset_auth_status``()`

Returns

Boolean (`True` on success, otherwise `False`).

python_ntp_show_peer_status()

Shows the NTP peers.

Syntax

```
python_ntp_show_peer_status()
```

Returns

A dictionary containing the NTP peers:

Parameter	**Description**
remote	The IP of the remote machine (string).
reach	An 8-bit rotating register. Any 1 bit means a "time packet" was received (string).
st	The status of the remote machine (string). Valid values: `0` - the best value, `16` - unsynchronized.
t	The available types (string). Valid values: `"l"` - for local, `"u"` - for unicast, `"m"` - for multicast, `"b"` - for broadcast, `"-"` - for network address.
when	How many seconds have passed since the last poll (string).
poll	The polling interval, in seconds (string).
delay	The time interval, in milliseconds between our time and that of the remote (string).
offset	The offset, in milliseconds between our time and that of the remote (string).
jitter	The observed jitter, in milliseconds of time with the remote (string).
refid	The identification of the time source to which the remote machines is synced. Valid values: 32-bit, ASCII code identifying the particular server or reference clock.

python_ntp_show_statistics_io()

Shows the NTP peer statistics IO.

Syntax

`python_ntp_show_statistics_io``()`

Returns

A dictionary containing the NTP peer statistics:

Parameter	**Description**
used_received_buffers	The number of used receive buffers (string).
ignored_packets	The number of ignored packets (string).
free_receive_buffers	The number of free receive buffers (string).
dropped_packets	The number of dropped packets (string).
useful_input_wakeups	The number of useful input wake ups (string).
packets_sent	The number of packets sent (string).
received_packets	The number of received packets (string).
input_wakeups	The number of input wake ups (string).
receive_buffers	The number of receive buffers (string).
packet_send_failures	The number of packet send failures (string).
time_since_reset	The time since reset, in milliseconds (string).
low_water_refills	The number of low water refills (string).

python_ntp_show_statistics_local()

Gets the peer local statistics.

Syntax

```
python_ntp_show_statistics_local()
```

Returns

A dictionary containing the peer local statistics:

Parameter	**Description**
uptime	The system uptime, in milliseconds (string).
sysstats_reset	The time, in milliseconds since the system reset (string).
packets_received	The number of packets received (string).
restricted	The number of restricted packets (string).
processed_for_time	Processed for time (string).
bad_length_or_format	Bad length of format (string).
current_version	The current version (string).
declined	Declined (string).
older_version	The older version (string).
rate_limited	Rate limited (string).
authentication_failed	The number of failed authentications (string).
KoD_responses	The number of kiss of death responses triggered (string).

python_ntp_show_statistics_memory()

Gets the peer memory statistics.

Syntax

`python_ntp_show_statistics_memory``()`

Returns

A dictionary containing the peer memory statistics:

Parameter	Description
reclaim_above_count	Number of reclaims above counter (string).
addresses	The peer addresses (string).
maximum_kilobytes	The maximum number of kilobytes (string).
enabled	Enabled status (string). Valid values: `"0x1"`.
reclaim_older_than	The reclaim above count (string).
maximum_addresses	The maximum number of addresses (string).
kilobytes	The number of number of kilobytes (string).
peak_addresses	The number of peak addresses (string).

python_ntp_show_statistics_peer()

Gets the peer statistics for a specified IP address.

Syntax

```
python_ntp_show_statistics_peer()
```

Returns

A dictionary containing the peer statistics:

Parameter	**Description**
status	List containing general information about the specific peer (string).
bad_dispersion	The number of bad dispersions (string).
time_until_next_send	The time until the send packet (string).
candidate_order	The candidate order number (string).
packets_sent	The number of sent packet (string).
associd	The associated ID (string).
remote_host	The number of remote host ID (string).
time_last_received	The time since the last sent packet (string).
duplicate	Duplicate (string).
bad_reference_time	Bad reference time (string).
bad_authentication	Bad authentication (string).
local_address	The local address (string).
packets_received	The number of received packets (string).
bogus_origin	The bogus origin (string).
reachability_change	The reachability change (string).

python_ntp_show_keys_info()

Gets authentication keys information.

Syntax

`python_ntp_show_keys_info``()`

Returns

A list with the authentication keys information:

Parameter	**Description**
key_type	The type of the authentication key (string). Valid values: `"sha1"`, `"md5"`.
key_value	The authentication key. A md5/sha1 string up to 8 characters long.
key_num	The authentication key number. An integer from 1-65534.

python_ntp_set_auth_keys()

Sets the NTP authentication key.

Syntax

`python_ntp_set_auth_keys``(`*dict_ntp*`)`

where: *dict_ntp* is a dictionary containing the following variables:

Parameter	**Description**
key_num	The authentication key number. An integer from 1-65534.
md5 *sha1*	The authentication key. A md5/sha1 string up to 8 characters long.

Returns

Boolean (`True` on success, otherwise `False`).

python_ntp_unset_auth_keys()

Unsets the NTP authentication key.

Syntax

`python_ntp_unset_auth_keys``(`*key*`)`

where::

Parameter	Description
key	The authentication key number. An integer from 1-65534.

Returns

Boolean (`True` on success, otherwise `False`).

python_ntp_show_trusted_keys_info()

Gets the trusted keys list.

Syntax

`python_ntp_show_trusted_keys_info``()`

Returns

A dictionary containing NTP trusted keys:

Parameter	Description
key_num	The trusted key number. An integer from 1-65534.

python_ntp_set_trusted_keys()

Sets a trusted key.

Syntax

`python_ntp_set_trusted_keys``(key)`

where::

Parameter	Description
key	The authentication key number. An integer from 1-65534.

Returns

Boolean (`True` on success, otherwise `False`).

python_ntp_unset_trusted_keys()

Unsets a trusted key.

Syntax

`python_ntp_unset_trusted_keys``(`*key*`)`

where::

Parameter	Description
key	The authentication key number. An integer from 1-65534.

Returns

Boolean (`True` on success, otherwise `False`).

python_ntp_show_peers_info()

Gets the configured NTP servers and peers.

Syntax

`python_ntp_show_peers_info``()`

Returns

A dictionary containing NTP peers/server information:

Parameter	Description
max_poll	The maximum poll interval for NTP messages. An integer from 3-17.
name	The IP address of peer/server (string).
server_type	The server type (string). Valid values: `“static”`, `“dynamic”`.
minpoll	The minimum poll interval for NTP messages. An integer from 3-17.
prefer	The server/peer prefer option (string). Valid values: `"yes"`, `"no"`.
key	The authentication key number. An integer from 1-65534.
type	The configured peer or server (string). Valid values: `"server"`, `"peer"`.

python_ntp_set_peer()

Sets a NTP server or peer.

Syntax

`python_ntp_set_peer(``*ntp_type, name, key, prefer, minpoll, maxpoll*`**`)`**

where:

Parameter	Description
ntp_type	The configured peer or server (string). Valid values: `"server"`, `"peer"`.
name	The IP address of peer/server (string).
key	(Optional) The authentication key number. An integer from 1-65534.
prefer	(Optional) The server/peer prefer option (string). Valid values: `"yes"`, `"no"`.
minpoll	(Optional) The minimum poll interval for NTP messages. An integer from 4-16. Default value: `-1`.
max_poll	(Optional) The maximum poll interval for NTP messages. An integer from 4-16.Default value: `-1`.

Returns

Boolean (`True` on success, otherwise `False`).

python_ntp_unset_peer()

Unsets a trusted key.

Syntax

`python_ntp_unset_peer(`*name, ntp_type***`)`**

where::

Parameter	Description
name	The IP address of peer/server (string).
ntp_type	The configured peer or server (string). Valid values: `"server"`, `"peer"`.

Returns

Boolean (`True` on success, otherwise `False`).

python_ntp_get_vrf()

Gets the NTP VRF mode.

Syntax

`python_ntp_get_vrf``()`

Returns

A dictionary containing VRF details:

vrf	The NTP VRF mode (string).

python_ntp_set_vrf()

Sets the NTP VRF mode.

Syntax

`python_ntp_set_vrf``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	The NTP VRF mode (string).

Returns

Boolean (`True` on success, otherwise `False`).

NWV Module

This module manages Network Virtualization (NWV) configuration.

To use this module, in the Python file or in the Python interpreter, enter:

`import nwvApi`

class NwvCfg()

This class provides a function to set NWV configurations.

get_nwv()

Gets NWV mode information.

Syntax

`get_nwv``()`

Returns

A dictionary containing NWV information:

Parameter	**Description**
encapsulation	The VXLAN encapsulation (string). Default value: ”`vtep`”.
ha	The High Availability mode status (string); one of ”`disabled`”, ”`vlag`”. Default value: ”`disabled`”.
mode	The NWV mode (string); one of ”`disabled`”, ”`static`”, ”`bgp-evpn`”, ”`Vxlan not ready`”. Default value: ”`disabled`”. **Note:** When the mode is ”`Vxlan not ready`”, the configuration is incorrect and you must change the vLAG settings.

set_nwv_mode()

Sets the NWV mode.

Syntax

`set_nwv_mode``(`*mode*`)`

where:

Parameter	**Description**
mode	The new NWM mode. An integer with the following values: ● `0` (Disabled) ● `1` (Static) ● `3` (Bgp-epvn) ● `4` (Static-ha) ● `5` (Bgp-evpn-ha)

Returns

Boolean (`True` on success, otherwise `False`). Otherwise, you must change the NWV mode. For example, to change the `Bgp-epvn` mode, you must first set it to `Disabled` and then `Static`. It is not valid to change the `Bgp-epvn` mode directly to `Static`.

OSPF Module

This module manages Open Shortest Path First (OSPF) information.

To use this module, in the Python file or in the Python interpreter, enter:

`import ospfApi`

class OSPF()

This class provides functions to get and set OSPF configurations.

python_ospf_get_stats_info()

Gets OSPF global statistics.

Syntax

`python_ospf_get_stats_info``(`*vrf_name*`)`

where:

Parameter	Description
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name or "`default`". Default value: "`default`".

Returns

A dictionary containing OSPF global statistics:

Parameter	Description
vrf_name	Default VRF name. Default value: "`default`".
ospf_id	OSPF identifier. Default value: `0`.
clr_timer_str	Time since last OSPF process clear. A string in the following format: "`HH:MM:SS`".
router_id_changes	Router ID changes counter. A positive integer.
dr_election_counter	DR elections counter. A positive integer.
older_lsas_counter	Older received LSAs counter. A positive integer.
nbr_state_change_counter	Neighbor state changes counter. A positive integer.
nbr_bad_lsreqs_counter	Neighbor bad LS received requests counter. A positive integer.
nbr_interval_expired_counter	Neighbor dead-interval expirations counter. A positive integer.

Parameter	Description
nbr_seq_number_mismatch	Neighbor sequence number mismatches counter. A positive integer.
spf_full	Full SPF Computations counter. A positive integer.
spf_summary	Summary SPF Computations counter. A positive integer.
spf_external	External SPF Computations counter. A positive integer.
recv_buf	Received packet buffer. A positive integer.
send_buf	Sent packet buffer. A positive integer.
lsa_buf	LSA buffer. A positive integer.
packet_unuse	Unused packets number. A positive integer.
packet_max	Maximum packets number. A positive integer.
lsa_unuse	Unused LSAs number. A positive integer.
lsa_max	Maximum LSAs number. A positive integer.
router_lsa_type	Router LSA type name. A positive integer.
routerLsa_generated	Number of generated router LSAs. A positive integer.
routerLsa_refreshed	Number of refreshed router LSA. A positive integer.
routerLsa_flushed	Number of flushed router LSAs. A positive integer.
routerLsa_agedOut	Number of aged out router LSAs. A positive integer.
networkLsa_generated	Number of generated network LSAs. A positive integer.
networkLsa_refreshed	Number of refreshed network LSAs. A positive integer.
networkLsa_flushed	Number of flushed network LSAs. A positive integer.
networkLsa_agedOut	Number of aged out network LSAs. A positive integer.
summaryLsa_generated	Number of generated summary LSAs. A positive integer.
summaryLsa_refreshed	Number of refreshed summary LSAs. A positive integer.

Parameter	**Description**
summaryLsa_flushed	Number of flushed summary LSAs. A positive integer.
summaryLsa_agedOut	Number of aged out summary LSAs. A positive integer.
asbrSummaryLsa_generated	Number of generated ASBR summary LSAs. A positive integer.
asbrSummaryLsa_refreshed	Number of refreshed ASBR summary LSAs. A positive integer.
asbrSummaryLsa_flushed	Number of flushed ASBR summary LSAs. A positive integer.
asbrSummaryLsa_agedOut	Number of aged out ASBR summary LSAs. A positive integer.
asExternalLsa_generated	Number of generated AS-External LSAs. A positive integer.
asExternalLsa_refreshed	Number of refreshed AS-External LSAs. A positive integer.
asExternalLsa_flushed	Number of flushed AS-External LSAs. A positive integer.
asExternalLsa_agedOut	Number of aged out AS-External LSAs. A positive integer.
asNssaLsa_generated	Number of generated AS-NSSA LSAs. A positive integer.
asNssaLsa_refreshed	Number of refreshed AS-NSSA LSAs. A positive integer.
asNssaLsa_flushed	Number of flushed AS-NSSA LSAs. A positive integer.
asNssaLsa_agedOut	Number of aged out AS-NSSA LSAs. A positive integer.
type8Lsa_generated	Number of generated type-8 LSAs. A positive integer.
type8Lsa_refreshed	Number of refreshed type-8 LSAs. A positive integer.
type8Lsa_flushed	Number of flushed type-8 LSAs. A positive integer.
type8Lsa_agedOut	Number of aged out type-8 LSAs. A positive integer.
linkOpaqueLsa_generated	Number of generated Link Opaque LSAs. A positive integer.

Parameter	**Description**
linkOpaqueLsa_refreshed	Number of refreshed Link Opaque LSAs. A positive integer.
linkOpaqueLsa_flushed	Number of flushed Link Opaque LSAs. A positive integer.
linkOpaqueLsa_agedOut	Number of aged out Link Opaque LSAs. A positive integer.
areaOpaque_lsa_type	Area Opaque LSA type name. A positive integer.
areaOpaqueLsa_generated	Number of generated Area Opaque LSAs. A positive integer.
areaOpaqueLsa_refreshed	Number of refreshed Area Opaque LSAs. A positive integer.
areaOpaqueLsa_flushed	Number of flushed Area Opaque LSAs. A positive integer.
areaOpaqueLsa_agedOut	Number of aged out Area Opaque LSAs. A positive integer.
asOpaque_lsa_type	AS Opaque LSA type name. A positive integer.
asOpaqueLsa_generated	Number of generated AS External Opaque LSAs. A positive integer.
asOpaqueLsa_refreshed	Number of refreshed AS External Opaque LSAs. A positive integer.
asOpaqueLsa_flushed	Number of flushed AS External Opaque LSAs. A positive integer.
asOpaqueLsa_agedOut	Number of aged out AS External Opaque LSAs. A positive integer.

python_ospf_get_traffic_info()

Gets OSPF traffic statistics.

Syntax

`python_ospf_get_traffic_info`(*vrf_name*)

where:

Parameter	Description
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or "`default`". Default value: "`default`".

Returns

A dictionary containing OSPF traffic statistics:

Parameter	Description
vrf_name	Default VRF name (string). Default value: "`default`".
ospf_id	OSPF identifier. Default value: `0`.
timer_str	Time since last OSPF process clear. A string in the following format: "`HH:MM:SS`".
total_pkt_in	Number of total packets in. A positive integer.
total_pkt_out	Number of total packets out. A positive integer.
hello_in	Number of hello packets in. A positive integer.
hello_out	Number of hello packets out. A positive integer.
db_desc_in	Number of DB descriptor packets in. A positive integer.
db_desc_out	Number of DB descriptor packets out. A positive integer.
ls_req_in	Number of LS Request packets in. A positive integer.
ls_req_out	Number of LS Request packets out. A positive integer.
ls_upd_in	Number of LS Update packets in. A positive integer.
ls_upd_out	Number of LS Update packets out. A positive integer.
ls_ack_in	Number of LS ACK packets in. A positive integer.
ls_ack_out	Number of LS ACK packets out. A positive integer.
error_drops_in	Number of errors related to drops in. A positive integer.
error_drops_out	Number of errors related to drops out. A positive integer.
error_hellosin	Number of errors related to hellos in. A positive integer.

Parameter	**Description**
error_dbsin	Number of errors related to DB Descriptors. A positive integer.
error_lsreqin	Number of errors related to LS Requests. A positive integer.
error_lsuin	Number of errors related to LS Updates. A positive integer.
error_lsackin	Number of errors related to LS ACKs. A positive integer.
error_unknown_in	Number of errors related to unknown in. A positive integer.
error_unknown_out	Number of errors related to unknown out. A positive integer.
error_badcrc	Number of errors related to Bad CRC. A positive integer.
error_wrong_area	Number of errors related to Wrong Area. A positive integer.
error_bad_version	Number of errors related to Bad Version. A positive integer.
error_bad_auth	Number of errors related to Bad Authentication. A positive integer.
error_passive	Number of errors related to Passive. A positive integer.
error_nonbr	Number of errors related to No Neighbor. A positive integer.
error_invalid_src	Number of errors related to Invalid Source. A positive integer.
error_invalid_dst	Number of errors related to Invalid Destination. A positive integer.
error_pktlength	Number of errors related to Packet Length. A positive integer.

python_ospf_get_neighbor_info()

Gets OSPF neighbors statistics.

Syntax

`python_ospf_get_neighbor_info``(`*vrf_name*`)`

where:

Parameter	Description
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values" the VRF name or "`default`". Default value: "`default`".

Returns

A dictionary containing OSPF neighbor statistics:

Parameter	Description
vrf_name	Default VRF name (string). Default value: "`default`".
nbr_router_id	Neighbor router ID identifier. A valid IPv4 or IPv6 address.
priority	The neighbor priority. An integer from 0-255.
dead_timer	The time left for dead interval expiry. A string in the following format: "`HH:MM:SS`".
nbr_addr	Neighbor IP address. A valid IPv4 or IPv6 address.
ifp_name	Ethernet interface name.

python_ospf_get_routes_info()

Gets OSPF routes statistics.

Syntax

`python_ospf_get_routes_info``(`*vrf_name*`)`

where:

Parameter	Description
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values" the VRF name or `"default"`. Default value: `"default"`.

Returns

A dictionary containing OSPF routes statistics:

Parameter	Description
network	Network name. A string in the following format: `"AA:BB:CC:DD/MM"`.
pathcode	Path type (string). Valid values: ● `"connected"` ● `"Discard"` ● `"OSPF"` ● `"OSPF inter area"` ● `"OSPF NSSA external type 1"` ● `"OSPF NSSA external type 2"` ● `"OSPF external type 1"` ● `"OSPF external type 2"`
pathCount	Number of ecmp paths. A positive integer.
route_path_cost	Route-path cost. A positive integer.
route_type2path_cost	Route-type 2 path cost. A positive integer.
next_hop_info	Next-hop information. A list of dictionaries. Depending on the configuration, each dictionary may contain the following values: ● `interface`*:* Neighbor IP address. A valid IPv4 or IPv6 address. ● `area_id`*:* Neighbor area ID. A valid IPv4 or IPv6 address. ● `neighbor_addr`*:* Neighbor IP address. A valid IPv4 or IPv6 address.

python_ospf_get_database_info()

Gets OSPF database statistics.

Syntax

`python_ospf_get_database_info``(`*vrf_name*`)`

where:

Parameter	Description
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values" the VRF name or `"default"`. Default value: `"default"`.

Returns

A dictionary containing OSPF traffic statistics:

Parameter	Description
link_state_id	VRF name. A valid IPv4 or IPv6 address.
adv_router	Advertising router ID. A valid IPv4 or IPv6 address.
lsa_type	LSA type. Valid values: ● `Router-LSA` ● `Network-LSA` ● `Summary-LSA` ● `ASBR-summary-LSA` ● `AS-external-LSA` ● `AS-NSSA-LSA`
lsa_age	LSA age. A positive integer.
ls_seqnum_str	LS sequence number in hexadecimal format.
checksum	LSA checksum in hexadecimal format.
link count	Links number. A positive integer.
area_id	The area-ID of the LSDB. A valid IPv4 or IPv6 address.
route	Network route (string).
tag	External/NSSA LSAs tag, A positive integer.
metric_type	Name of the type of metric (string). Valid values: `""E1", "E2", "N1", "N2"`.

python_ospf_get_border_routers_info()

Gets OSPF border routers.

Syntax

`python_ospf_get_border_routers_info``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values" the VRF name or "`default`". Default value: "`default`".

Returns

A dictionary containing OSPF Area Border Router (ABR) and Autonomous System Boundary Router (ASBR) statistics.:

Parameter	**Description**
abr_id	The ABR ID. A string showing the type, router ID or cost.
abr_route_type	Type of router related to ABR (string).
abr_route_metric	Metric of router related to ABR (string).
asbr_id	The ASBR ID. A string showing the type, router ID or cost.
asbr_route_type	Type of router related to ASBR (string).
asbr_route_metric	Metric of router related to ASBR (string).
type_border_router	The border router type (string). Valid values: "`ABR`" or "`ASBR`".
abr_via	The next-hop IP for ABR (string). A valid IP address.
asbr_via	The next-hop IP for ABSBR (string). A valid IP address.
abr_transit_area	The transit area ID for ABR (string). A valid IP address.
asbr_transit_area	The transit area ID for ASBR (string). A valid IP address.
abr_area_ifname	The OSPF interface for ABR (string). For example: "`Ethernet1/X`" or "`VLAN interface`".
asbr_area_ifname	The OSPF interface for ABSR (string). For example: "`Ethernet1/X`" or "`VLAN interface`".

python_ospf_get_summary_address_info()

Gets OSPF summary address.

Syntax

`python_ospf_get_summary_address_info``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values" the VRF name or "`default`". Default value: "`default`".

Returns

A dictionary showing OSPF summary address information:

Parameter	**Description**
router_id	Router ID in IP address format (string). A valid IP address.
ospf_id	OSPF identifier (integer). Default value: `0`.
vrf_name	Default VRF name (string). Default value: "`default`".
prefix	The IP prefix. A string in the following format: "`XX.XX.XX.XX/XX`".
metric	The metric value. An integer from 0-16777214.
tag	External/NSSA LSAs tag; A positive integer from 0-4294967295.
summary_address_state	The summary address status (string). Valid values: "`Active`", "`Pending`".

python_ospf_get_interface_info()

Gets OSPF interface information.

Syntax

`python_ospf_get_interface_info``(`*if_name, vrf_name*`)`

where:

Parameter	Description
if_name	(Optional) Ethernet interface name (string). **Note:** The interface must exist.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or “`default`”. Default value: “`default`”.

Returns

A dictionary showing OSPF interface information:

Parameter	Description
if_name	The interface name (string). For example: “`Ethernet1/X`” or “`VLAN interface`”.
vrf_name	Default VRF name (string). Default value: ”`default`”.
ospf_id	OSPF process identifier (integer). Default value: `0`.
ospf_status	The status of the OSPF protocol (string). Valid values: “`Up`”, ”`Down`”.
if_addr	The IP address or mask (string). A valid IP address or mask.
if_area_id	The area ID (string). A valid IP address.
if_mtu	The maximum transmission unit. A positive integer from 576-65535.
router_id	The router ID in IP address format (string). A valid IP address.
if_network_type	The network type (string). Valid values: ”`Broadcast`”, ”`Point-to-Point`”.
if_output_cost	Interface output cost. A positive integer from 1-65535.
if_transmit_delay	The interface transmit delay, in seconds. An integer from 1-3600.
priority	The router priority. An integer from 0-255.

Parameter	Description
if_state	The operation state of the interface (string). Valid values: `"DR"`, `"Backup"`, `"DRother"`.
designated_router	Designated router ID (string). A valid IP address.
designated_router_addr	The IP address for the designated router (string).
backup_designated_router	The backup router ID for the designated router (string). A valid IP address.
backup_designated_router_ addr	The backup router ID for the designated router. (string).
hello_interval	The hello interval, in seconds. An integer from 1-65535.
dead_interval	The dead interval, in seconds. An integer from 1-65535.
retransmit_interval	The retransmit interval, in seconds. An integer from 1-65535.
if_hello_timer	The hello interval timer expiration time (string).
neighbor_count	The neighbor count (integer). A positive integer.
adj_neighbor_count	The adjacent neighbor count (integer). A positive integer.
hello_in	Number of total hello packets in. A positive integer.
hello_out	Number of total hello packets out. A positive integer.
ls_req_in	Number of total LS Request packets in. A positive integer.
ls_req_out	Number of total LS Request packets out. A positive integer.
ls_upd_in	Number of total LS Update packets in. A positive integer.
ls_upd_out	Number of total LS Update packets out. A positive integer.
ls_ack_in	Number of total LS ACK packets in. A positive integer.
ls_ack_out	Number of total LS ACK packets out. A positive integer.
db_desc_in	Number of total DB Descriptors packets in. A positive integer.
db_desc_out	Number of total DB Descriptors packets out. A positive integer.

Parameter	**Description**
discarded	Number of total discarded packets. A positive integer.
auth_type	The type of authentication. Valid values: `“message-digest”`, `”simple”`, `”null”`.
key_id	The Key-ID, if the authentication type is MD5/SHA256. An integer from 1-255.
if_mtu_ignore	The maximum transmission unit status (string). Valid values: `“Enable”`, `”Disable”`.
passive_interface	The passive interface status (string). Valid values: `“Enable”`, `”Disable”`.
if_bfd	The BDF status (string). Valid values: `“Enable”`, `”Disable”`.
db_filter_all_out	Database filter all out (string). Valid values: `“Enable”`, `”Disable”`.

python_ospf_get_vlinks_info()

Gets OSPF virtual-links.

Syntax

`python_ospf_get_vlinks_info``(`*area_id, nbr_router_id, vrf_name*`)`

where:

Parameter	**Description**
area_id	(Optional) The transit area ID (string). A valid IP address.
nbr_router_id	(Optional) The neighbor router ID (string). A valid IP address.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

Returns

A dictionary containing OSPF virtual-links statistics:

Parameter	**Description**
vrf_name	Default VRF name (string). Default value: `”default”`.
vlink_name	The virtual-link name (string).

Parameter	Description
nbr_router_id	The neighbor router ID (string). A valid IP address.
ifp_name	The interface name (string). For example: `“Ethernet1/X”` or `“VLAN interface”`.
local_address	The local interface IP address (string). A valid IP address.
remote_address	The remote interface IP address (string). A valid IP address.
area_id	The transit area ID (string). A valid IP address.
transmit_delay	The transmission delay interval, in seconds. An integer from 1-3600.
vlink_state	The Virtual Link status (string). Valid values: `“Up”`, `”Down”`.
hello_interval	The hello interval, in seconds. An integer from 1-65535.
dead_interval	The dead interval, in seconds. An integer from 1-65535.
wait_interval	The wait interval, in seconds; an integer from 1-65535.
retransmit_interval	The retransmit interval, in seconds. An integer from 1-65535.
hello_due	The due time to send the next hello (string).
adjacency_state	The adjacency state across the virtual-link (string).
auth_type	The type of authentication (string). Valid values: `“message-digest`, `”simple”`, `”null”`.
key_id	The Key-ID, if the authentication type is MD5/SHA256. An integer from 1-255.
bfd	The status of BFD (string). Valid values: `“Enable”`, `“Disable”`.

python_set_ospf_if_ospf_status()

Sets OSPF on an interface.

Syntax

`python_set_ospf_if_ospf_status`(*interface, ospf_status, vrf_name*)

where:

Parameter	Description
interface	The interface name (string). For example: “`Ethernet1/X`” or “`VLAN interface`”.
ospf_status	The status of the OSPF protocol (string). Valid values: “`Enable`”, “`Disable`”.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or “`default`”. Default value: “`default`”.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_hello_interval()

Sets the hello interval for the OSPF interface.

Syntax

`python_set_ospf_if_hello_interval`(*interface, hello_interval, vrf_name*)

where:

Parameter	Description
interface	The interface name (string). For example: “`Ethernet1/X`” or “`VLAN interface`”.
hello_interval	The hello interval, in seconds. An integer from 1-65535.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or “`default`”. Default value: “`default`”.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_dead_interval()

Sets the dead interval for the OSPF interface.

Syntax

`python_set_ospf_if_dead_interval`(*interface, dead_interval, vrf_name*)

where:

Parameter	Description
interface	The interface name (string). For example: "`Ethernet1/X`" or "`VLAN interface`".
dead_interval	The dead interval, in seconds. An integer from 1-65535.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or "`default`". Default value: "`default`".

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_retransmit_interval()

Sets the retransmit interval for the OSPF interface.

Syntax

`python_set_ospf_if_retransmit_interval`(*interface, retransmit_interval, vrf_name*)

where:

Parameter	Description
interface	The interface name (string). For example: "`Ethernet1/X`" or "`VLAN interface`".
retransmit_interval	The retransmit interval, in seconds. An integer from 1-65535.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or "`default`". Default value: "`default`".

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_transmit_delay()

Sets the transmit delay for the OSPF interface.

Syntax

`python_set_ospf_if_transmit_delay`(*interface, if_transmit_delay, vrf_name*)

where:

Parameter	**Description**
interface	The interface name (string). For example: "`Ethernet1/X`" or "`VLAN interface`".
if_transmit_delay	The interface transmit delay. An integer from 1-3600.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or "`default`". Default value: "`default`".

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_priority()

Sets the OSPF interface priority.

Syntax

`python_set_ospf_if_priority`(*interface, priority, vrf_name*)

where:

Parameter	**Description**
interface	The interface name (string). For example: "`Ethernet1/X`" or "`VLAN interface`".
priority	The router priority. An integer from 0-255.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or "`default`". Default value: "`default`".

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_output_cost()

Sets the OSPF interface output cost.

Syntax

`python_set_ospf_if_output_cost`(*interface, cost, vrf_name*)

where:

Parameter	**Description**
interface	The interface name (string). For example: “`Ethernet1/X`” or “`VLAN interface`”.
cost	The OSPF cost. An integer from 0-65535.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or “`default`”. Default value: “`default`”.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_network_type()

Sets the interface network type.

Syntax

`python_set_ospf_if_network_type`(*interface, network_type, vrf_name*)

where:

Parameter	**Description**
interface	The interface name (string). For example: “`Ethernet1/X`” or “`VLAN interface`”.
network_type	The network type (string). Valid values: “`Broadcast`”, ”`Point-to-Point`”.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or “`default`”. Default value: “`default`”.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_mtu()

Sets the Maximum Transmission Unit (MTU) for the OSPF interface.

Syntax

`python_set_ospf_if_mtu`(*interface, if_mtu, vrf_name*)

where:

Parameter	**Description**
interface	The interface name (string). For example: "`Ethernet1/X`" or "`VLAN interface`".
if_mtu	The maximum transmission unit. A positive integer from 576-65535.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or "`default`". Default value: "`default`".

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_mtu_ignore()

Sets interface MTU to ignore.

Syntax

`python_set_ospf_if_mtu_ignore`(*interface, if_mtu_ignore, vrf_name*)

where:

Parameter	**Description**
interface	The interface name (string). For example: "`Ethernet1/X`" or "`VLAN interface`".
if_mtu_ignore	Interface MTU ignore (string). Valid values: "`Enable`", "`Disable`".
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or "`default`". Default value: "`default`".

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_bfd()

Sets interface BFD.

Syntax

`python_set_ospf_if_bfd`(*interface, if_bfd, vrf_name*)

where:

Parameter	Description
interface	The interface name (string). For example: “`Ethernet1/X`” or “`VLAN interface`”.
if_bfd	Interface BFD (string). Valid values: ”`Enable`”, “`Disable`”.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or “`default`”. Default value: “`default`”.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_authentication_type()

Sets the interface authentication type.

Syntax

`python_set_ospf_if_authentication_type`(*interface, auth_type, vrf_name*)

where:

Parameter	Description
interface	The interface name (string). For example: “`Ethernet1/X`” or “`VLAN interface`”.
auth_type	The type of authentication (string). Valid values: “`message-digest`”, ”`simple`”, ”`null`”.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or “`default`”. Default value: “`default`”.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_authentication_key()

Sets the interface authentication key.

Syntax

`python_set_ospf_if_authentication_key`(*interface, auth_key, vrf_name*)

where:

Parameter	Description
interface	The interface name (string). For example: `“Ethernet1/X”` or `“VLAN interface”`.
auth_key	The authentication key (string).
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_message_digest_key()

Configures the MD5/SHA-256 key.

Syntax

`python_set_ospf_if_message_digest_key`(*interface, key_id, key_type, password, vrf_name*)

where:

Parameter	Description
interface	The interface name (string). For example: `“Ethernet1/X”` or `“VLAN interface”`.
key_id	The key identifier, An integer from 1-255.
key_type	The key type, MD5:1 or SHA-256:2 (integer). Valid values:`1`, `2`.
password	The encrypted MD5 or SHA-256 password (string).
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_passive()

Sets the interface as passive.

Syntax

`python_set_ospf_if_passive`(*interface, if_passive_cfg, vrf_name*)

where:

Parameter	Description
interface	The interface name (string). For example: “`Ethernet1/X`” or “`VLAN interface`”.
if_passive_cfg	The interface passive configuration (string). Valid valueS: “`Enable`”, ”`Disable`”.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or “`default`”. Default value: “`default`”.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_area_id()

Sets the area ID for the OSPF interface.

Syntax

`python_set_ospf_if_area_id`(*interface, area_id, vrf_name*)

where:

Parameter	Description
interface	The interface name (string). For example: “`Ethernet1/X`” or “`VLAN interface`”.
area_id	The area ID (string). A valid IP address.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or “`default`”. Default value: “`default`”.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_db_filter_out()

Configures the database filter.

Syntax

`python_set_ospf_if_db_filter_out`(*interface, db_filter_out, vrf_name*)

where:

Parameter	**Description**
interface	The interface name (string). For example: "`Ethernet1/X`" or "`VLAN interface`".
db_filter_out	The database filter out configuration (string). Valid values: "`Enable`", "`Disable`".
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or "`default`". Default value: "`default`".

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_unset_config()

Unsets the list of interface configurations.

Syntax

`python_set_ospf_if_unset_config`(*interface, list_unset, vrf_name*)

where:

Parameter	**Description**
interface	The interface name (string). For example: “`Ethernet1/X`” or “`VLAN interface`”.
list_unset	Displays a list of strings: ● `auth_key` ● `auth_type` ● `hello_interval` ● `dead_interval` ● `if_transmit_delay` ● `retransmit_interval` ● `if_output_cost` ● `priority` ● `if_mtu`
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or “`default`”. Default value: “`default`”.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_if_unset_message_digest_key_config()

Unsets the OSPF interface MD5 key.

Syntax

`python_set_ospf_if_unset_message_digest_key_config` (*interface, key_id, vrf_name*)

where:

Parameter	Description
interface	The interface name (string). For example: `“Ethernet1/X”` or `“VLAN interface”`.
key_id	The MD5 key. An integer from 1-255.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_virtual_link()

Configures the OSPF virtual-link.

Syntax

`python_set_ospf_virtual_link`(*area_id, nbr_router_id, vrf_name*)

where:

Parameter	Description
area_id	The area ID (string). A valid IP address.
nbr_router_id	The neighbor router ID (string). A valid IP address.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_virtual_link_disable()

Disables the virtual-link.

Syntax

`python_set_ospf_virtual_link_disable`(*area_id, nbr_router_id, vrf_name*)

where:

Parameter	Description
area_id	The area ID (string). A valid IP address.
nbr_router_id	The neighbor router ID (string). A valid IP address.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_virtual_hello_interval()

Configures the hello interval for the virtual-link.

Syntax

`python_set_ospf_virtual_hello_interval`(*area_id, nbr_router_id, hello_interval, vrf_name*)

where:

Parameter	Description
area_id	The area ID (string). A valid IP address.
nbr_router_id	The neighbor router ID (string). A valid IP address.
hello_interval	The hello interval, in seconds. An integer from 1-65535.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_virtual_dead_interval()

Configures the dead interval for the virtual-link.

Syntax

`python_set_ospf_virtual_dead_interval`(*area_id, nbr_router_id, dead_interval, vrf_name*)

where:

Parameter	**Description**
area_id	The area ID (string). A valid IP address.
nbr_router_id	The neighbor router ID (string). A valid IP address.
dead_interval	The dead interval, in seconds. An integer from 1-65535.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or “`default`”. Default value: “`default`”.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_virtual_retransmit_interval()

Configures the retransmit interval for the virtual-link.

Syntax

`python_set_ospf_virtual_retransmit_interval`(*area_id, nbr_router_id, retransmit_interval, vrf_name*)

where:

Parameter	**Description**
area_id	The area ID (string). A valid IP address.
nbr_router_id	The neighbor router ID (string). A valid IP address.
retransmit_interval	The retransmit interval, in seconds. An integer from 1-65535.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or “`default`”. Default value: “`default`”.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_virtual_transmit_delay()

Configures the transmit delay for the virtual-link.

Syntax

`python_set_ospf_virtual_transmit_delay`(*area_id, nbr_router_id, transmit_delay, vrf_name*)

where:

Parameter	**Description**
area_id	The area ID (string). A valid IP address.
nbr_router_id	The neighbor router ID (string). A valid IP address.
transmit_delay	The interface transmit delay, in seconds. An integer from 1-3600.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or "`default`". Default value: "`default`".

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_virtual_bfd()

Configures the BFD for virtual-link.

Syntax

`python_set_ospf_virtual_bfd`(*area_id, nbr_router_id, bfd_cfg, vrf_name*)

where:

Parameter	**Description**
area_id	The area ID (string). A valid IP address.
nbr_router_id	The neighbor router ID (string). A valid IP address.
bfd_cfg	The interface BDF configuration (string). Valid values: "`Enable`", "`Disable`".
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or "`default`". Default value: "`default`".

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_virtual_authentication_type()

Configures the authentication type for the virtual-link.

Syntax

`python_set_ospf_virtual_authentication_type`(*area_id, nbr_router_id, auth_type, vrf_name*)

where:

Parameter	**Description**
area_id	The area ID (string). A valid IP address.
nbr_router_id	The neighbor router ID (string). A valid IP address.
auth_type	The type of authentication (string). Valid values: `“message-digest”`, `”simple”`, `“null”`.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_virtual_authentication_key()

Configures the authentication key for the virtual link.

Syntax

`python_set_ospf_virtual_authentication_key`(*area_id, nbr_router_id, auth_key, vrf_name*)

where:

Parameter	**Description**
area_id	The area ID (string). A valid IP address.
nbr_router_id	The neighbor router ID (string). A valid IP address.
auth_key	The authentication key (string).
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_virtual_message_digest_key()

Configures the MD5 key for the virtual-link.

Syntax

`python_set_ospf_virtual_message_digest_key`(*area_id, nbr_router_id, key_id, key_type, password, vrf_name*)

where:

Parameter	**Description**
area_id	The area ID (string). A valid IP address.
nbr_router_id	The neighbor router ID (string). A valid IP address.
key_id	The key identifier. An integer from 1-255.
key_type	The key type, MD5:1 or SHA-256:2 (integer). Valid values: `1`, `2`.
password	The encrypted MD5 or SHA-256 password (string).
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_virtual_unset_config()

Unsets the list of configurations for the virtual-link.

Syntax

`python_set_ospf_virtual_unset_config`(*area_id, nbr_router_id, list_unset, vrf_name*)

where:

Parameter	Description
area_id	The area ID (string). A valid IP address.
nbr_router_id	The neighbor router ID (string). A valid IP address.
list_unset	Displays a list of strings: ● `auth_key` ● `auth_type` ● `hello_interval` ● `dead_interval` ● `transmit_delay` ● `retransmit_interval`
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or “`default`”. Default value: “`default`”.

Returns

Boolean (`True` on success, otherwise `False`).

python_set_ospf_virtual_unset_message_digest_key_config()

Unsets the MD5/SHA-256 key configuration for virtual-link.

Syntax

`python_set_ospf_virtual_unset_message_digest_key_config` (*area_id, nbr_router_id, key_id, vrf_name*)

where:

Parameter	**Description**
area_id	The area ID (string). A valid IP address.
nbr_router_id	The neighbor router ID (string). A valid IP address.
key_id	The MD5 or SHA-256 key to unset (string). An integer from 1-255.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_ospf_get_proc_info()

Gets the OSPF process information.

Syntax

`python_ospf_get_proc_info`(*vrf_name*)

where:

Parameter	**Description**
vrf_name	(Optional) Virtual Routing and Forwarding name. Valid values: the VRF name, `“default”,”all”`. Default value: `“default”`.

Returns

dict_ospfProcInfo returns a dictionary containing the following OSPF processes:

Parameter	**Description**
ospfId	The OSPF process identifier (integer). Default value: `0`.
routerId	Router ID in IP address format (string). A valid IP address.

Parameter	**Description**
uptime	The OSPF process uptime. A string in the following format: `“HH:MM:SS”`.
vfrName	Default VRF name (string). Default value: `“default”.`
abrType	The ABR type (integer). Displays the valid ABR types.
spfStartDelaySec	The SPF schedule start delay, in seconds. An integer from 0-600.
spfStartDelayUsec	The SPF schedule start delay, in microseconds. An integer from 0-1000.
spfMinDelaySec	The minimum SPF schedule delay time. An integer from 1-600.
spfMinDelayUsec	The minimum SPF schedule delay time, in microseconds. An integer from 1-1000.
lsdbCount	The number of external LSAs. Zero or a positive integer.
lsdbChecksum	LAS checksum value (integer).
lsdbOverflow	The number of LSAs exceeding the limit. Zero or a positive integer.
originateNewLsas	The number of new originated LSAs. Zero or a positive integer.
rxNewLsas	The number of new LSAs received. Zero or a positive integer.
distance_all	The distance to all destinations. Zero or a positive integer.
distance_intra	The distance to intra-area destinations. Zero or a positive integer.
distance_inter	The distance to inter-area destinations. Zero or a positive integer.
distance_external	The distance to external destinations. Zero or a positive integer.
auth_type	The type of authentication (string). Valid values: `“message-digest”`, `“simple”`, `“null”`.
mode	The IS area shortcut (integer).
area_id	The area ID. An integer from 0-4294967295
area_type	The area type (string). Valid values: `”Default”`, `”Stub”`, `“NSSA”`.

Parameter	Description
active_if_count	The number of active interfaces in an area. Zero or a positive integer.
area_if_count	The number of interfaces in an area. Zero or a positive integer.
full_virt_nbr_count	Virtual neighbors count. Zero or a positive integer.
full_nbr_count	Total number of neighbors. Zero or a positive integer.
spf_calc_count	The number of SPF calculations. Zero or a positive integer.
area_lsdb_count	The number of LSAs in the area. Zero or a positive integer.
area_lsbd_checksum	The valid checksum of the link state database. A positive integer.

python_ospf_get_multiarea_info()

Gets OSPF multi-area neighbor information.

Syntax

`python_ospf_get_multiarea_info`(*vrf_name*)

where:

Parameter	Description
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or "`default`". Default value: "`default`".

Returns

dict_ospf_multiarea returns a dictionary containing the following OSPF processes:

Parameter	Description
ifName	The interface name (string). For example: "`Ethernet1/X`" or "`VLAN interface`".
nbr_addr	Neighbor IP address (string). A valid IP address.
type	The area type (string).
ifIpAddress	The interface IP address (string). A valid IP address.
ifAreaId	The interface area ID. An integer from 0-4294967295.
ifMTU	The maximum transmission unit (integer).

Parameter	**Description**
proc_id	The OSPF process identifier (integer). Default value: `0`.
ifRouterId	The router ID in IP address format (string). A valid IP address.
ifNetworkType	The interface network type (string). Valid values: `”Point-to-Point”`, `“Broadcast”`.
if_output_cost	Interface output cost. Zero or a positive integer.
if_transmit_delay	The interface transmit delay, in seconds (integer).
Transmit_if_state	The interface state type (integer).
d_router	Designated router ID (string). A valid IP address.
d_router_address	The IP address for the designated router (string).
bd_router	The backup router ID for the designated router (string). A valid IP address.
bd_router_address	The backup address for the designated router. (string).
hello_interval	The hello interval, in seconds (integer).
dead_interval	The dead interval, in seconds (integer).
retransmit_interval	The retransmit interval, in seconds (integer).
neighbor_count	The number of multi-area adjacent neighbors (integer).

python_ospf_get_ribcounters_info()

Gets OSPF rib counter information.

Syntax

`python_ospf_get_ribcounters_info`(*vrf_name*)

where:

Parameter	**Description**
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

Returns

dict_ospf_ribcounter returns a dictionary containing the following processes:

Parameter	**Description**
ospf2rib_route_add	The OSPF route addition calls made to RIB (integer).
ospf2rib_route_add_error	The OSPF to RIB route addition call errors (integer).
ospf2rib_route_delete	The OSPF to RIB route deletion calls (integer).
ospf2rib_route_delete_error	The OSPF to RIB route deletion call errors (integer).
ospf2rib_route_add	The OSPF route addition calls (integer).
ospf2rib_route_dels	The OSPF route deleted calls (integer).
ospf_route_adds_ignored	The OSPF ignored route addition calls (integer).
ospf_route_dels_ignored	The OSPF ignored route deleted calls (integer).
rib2ospf_route_add	The number of route additional calls from RIB to OSPF (integer).
rib2ospf_route_add_error	The number of route additional call errors from RIB to OSPF (integer).
rib2ospf_route_del	The number of route deleted calls from RIB to OSPF (integer).
rib2ospf_route_del_error	The number of route deleted call errors from RIB to OSPF (integer).

python_ospf_get_redist_config()

Gets OSPF redistribute configuration.

Syntax

`python_ospf_get_redist_config`(*vrf_name*)

where:

Parameter	Description
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

Returns

dict_ospf_redist returns a dictionary containing the following processes:

Parameter	Description
redist_direct	Redistribute the direct configuration (string). Valid values: `“Enable”`, `”Disable”`.
direct_metric	Redistribute the direct cost. An integer from 0-16777214.
direct_metric_type	The external metric type (integer). Valid values: `1`,`2`.
direct_tag	The tag value. An integer from 0-4294967295.
direct_rmap_name	The route-map name (string).
redist_bgp	Whether redistribute BGP is enabled (string). Valid values: `“Enable”`, `”Disable”`.
bgp_metric	Redistribute BGP cost. An integer from 0-16777214.
bgp_metric_type	The external metric type (integer). Valid values: `1`,`2.`
bgp_tag	The BGP tag value. An integer from 0-4294967295.
bgp_rmap_name	The BGP route map name (string).
redist_static	Whether redistribute static is enabled (string). Valid values: `“Enable”`, `”Disable”`.
static_metric	Redistribute static cost. An integer from 0-16777214.
static_metric_type	The external metric type (integer). Valid values: `1`,`2.`

Parameter	**Description**
static_tag	The tag value. An integer from 0-4294967295.
static_rmap_name	The static route map name (string).

python_ospf_put_redist_config()

Sets OSPF redistribute configuration.

Syntax

`python_ospf_put_redist_config`(*dict_redist, vrf_name*)

where:

Parameter	**Description**
dict_redist	The dictionary containing the OSPF redistribute configuration details.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

The *dict_redist* dictionary contains the following variables:

Parameter	**Description**
redist_direct	Redistribute the direct configuration (string). Valid values: `“Enable”`, `”Disable”`.
redist_static	Whether redistribute static is enabled (string). Valid values: `“Enable”`, `”Disable”`.
redist_bgp	Whether redistribute BGP is enabled. Valid values: `“Enable”`, `”Disable”`.
direct_metric	Redistribute the direct cost. An integer from 0-16777214.
direct_metric_type	The external metric type (integer). Valid vaues: `1`,`2`.
direct_tag	The tag value. An integer from 0-4294967295.
direct_rmap_name	The route-map name (string).
bgp_metric	Redistribute BGP cost. An integer from 0-16777214.
bgp_metric_type	The external metric type (integer). Valid values: `1`,`2.`
bgp_tag	The BGP tag value. An integer from 0-4294967295.
bgp_rmap_name	The BGP route map name (string).

Parameter	**Description**
static_metric	Redistribute static cost. An integer from 0-16777214.
static_metric_type	The external metric type (integer). Valid values: `1,2.`
static_tag	The tag value. An integer from 0-4294967295.
static_rmap_name	The static route map name (string).

Returns

`Success` or `Error`.

python_ospf_get_nssa_config()

Shows the OSPF NSSA area configuration.

Syntax

`python_ospf_get_nssa_config`(*vrf_name*)

where:

Parameter	**Description**
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `"default"`. Default value: `"default"`.

Returns

A dictionary containing the following elements:

Parameter	**Description**
nssa_area	The NSSA area configuration (string). Valid values: `"Enable"`, `"Disable"`.
nssa_area_id	The NSSA area ID IP address (string). A valid IP address.
nssa_def_info	The NSSA default information originate configuration (string) Valid values: `"Enable"`, `"Disable"`.
nssa_def_metric	The NSSA default metric. An integer from 0-16777214.
nssa_def_metric_type	The NSSA external metric type (integer). Valid values: `1`,`2`.
nssa_no_redist	Whether to stop redistribution in the NSSA area (string). Valid values: `"Enable"`, `"Disable"`.
nssa_no_summary	Whether to stop summary LSAs into the NSSA area (string). Valid values: `"Enable"`, `"Disable"`.
nssa_translate_always	`Always translate type7 LSA` (string). Valid values: `"Enable"`, `"Disable"`.
nssa_stability_interval	The NSSA stability interval. An integer from 0-2147483647.

python_ospf_put_nssa_config()

Sets the OSPF NSSA area configuration.

Syntax

`python_ospf_put_nssa_config`(*dict_nssa, vrf_name*)

where:

Parameter	**Description**
dict_nssa	The dictionary containing the OSPF NSSA configuration details.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

The *dict_nssa* dictionary contains the following variables:

Parameter	**Description**
nssa_area	The NSSA area configuration (string). Valid values: `“Enable”, ”Disable”`.
nssa_area_id	The NSSA area ID IP address (string). A valid IP address.
nssa_def_info	The NSSA default information originate configuration (string). Valid values: `“Enable”, ”Disable”`.
nssa_def_metric	The NSSA default metric. An integer from 0-16777214.
nssa_def_metric_type	The NSSA external metric type (integer). Valid values: `1,2`.
nssa_no_redist	Whether to stop redistribution in the NSSA area (string). Valid values: `“Enable”, ”Disable”`.
nssa_translate_always	`Always translate type7 LSA` (string). Valid values: `“Enable”, ”Disable”`.
nssa_stability_interval	The NSSA stability interval. An integer from 0-2147483647.

Returns

`Success` or `Error`.

python_ospf_put_ area_def_cost_config()

Sets the OSPF area default cost configuration.

Syntax

`python_ospf_put_area_def_cost_config`(*dict_def_cost, vrf_name*)

where:

Parameter	Description
dict_def_cost	The dictionary containing the default cost configuration details.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or "`default`". Default value: "`default`".

The *dict_def_cost* dictionary contains the following variables:

Parameter	Description
area_id	The area ID IP address (string). A valid IP address.
state	Whether the default cost is enabled (string). Valid values: "`Enable`", "`Disable`".
default_cost	The default summary cost value. An integer from 0-16777214.

Returns

`Success` or `Error`.

python_ospf_put_area_auth_config()

Sets area authentication configuration.

Syntax:

`python_ospf_put_area_auth_config`(*dict_auth, vrf_name*)

where:

Parameter	**Description**
dict_auth	The dictionary containing the authentication configuration details.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

The *dict_auth* dictionary contains the following variables:

Parameter	**Description**
area_id	The area ID IP address (string). A valid IP address.
auth	The authentication configuration (string);. Valid values: `“Enable”,”Disable”`.
auth-type	The type of authentication (integer). Valid values: one of `0`-null, `1`-simple or 2- cryptographic.

Returns

`Success` or `Error`.

python_ospf_put_summary_addr_config()

Sets summary address configuration.

Syntax

`python_ospf_put_summary_addr_config`(*dict_summary, vrf_name*)

where:

Parameter	**Description**
dict_summary	The dictionary containing the authentication address configuration details.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

The *dict_summary* dictionary contains the following variables:

Parameter	**Description**
summary_addr	Whether to enable summary address configuration (string). Valid values: `“Enable”,”Disable”`.
prefix	The IP address (string). A valid IP address.
masklen	The mask length. An integer from 0-32.
not-advertise	Whether to suppress routes that match the prefix (string). Valid values: `“Enable”,”Disable”`.
tag	The tag value. An integer from 0-4294967295.

Returns

`Success` or `Error.`

python_ospf_put_area_range_config()

Sets OSPF area range configuration.

Syntax

`python_ospf_put_area_range_config`(*dict_range, vrf_name*)

where:

Parameter	**Description**
dict_range	The dictionary containing the area range configuration details.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

The *dict_range* dictionary contains the following variables:

Parameter	**Description**
area_id	The area ID IP address (string). A valid IP address.
range	Whether to enable the range of IP addresses (string). Valid values: `“Enable”`, `”Disable”`.
prefix	The IP address (string). A valid IP address.
mask	The mask length. An integer from 0-32.
not-advertise	Whether to suppress routes that match the prefix (string). Valid values: `“Enable”`, `”Disable”`.

Returns

`Success` or `Error.`

python_ospf_put_overflow_db_config()

Sets overflow database configuration.

Syntax

`python_ospf_put_overflow_db_config`(*dict_overflow, vrf_name*)

where:

Parameter	Description
dict_overflow	The dictionary containing the overflow database configuration details.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

The *dict_overflow* dictionary contains the following variables:

Parameter	Description
db_overflow	Whether to enable the database overflow configuration (string). Valid values: `“Enable”`, `”Disable”`.
max_lsas	The maximum LSA limit. An integer from 0-4294967294.
limit	The database limit type (string). Valid values: `“hard”`, `“soft”`.
external	Whether to enable the external LSA limit (string). Valid values: `“Enable”`, `”Disable”`.
ext_max_lsas	The maximum external LSA limit. An integer from 0-2147483647.
recovery_time	Tme time to recover from the external LSA limit, in seconds. An integer from 0-65535.

Returns

`Success` or `Error.`

python_ospf_put_refbw_config()

Sets the autocost reference bandwith configuration.

Syntax

`python_ospf_put_refbw_config`(*dict_refbw, vrf_name*)

where:

Parameter	**Description**
dict_refbw	The dictionary containing the autocost reference bandwidth configuration details.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

The *dict_refbw* dictionary contains the following variables:

Parameter	**Description**
autocost_refbw	Whether to enable the autocost reference bandwidth configuration (string). Valid values: `“Enable”`, `”Disable”`.
bw_value	The bandwidth value. An integer from: ● 1-4294 for Gbps ● 1-4294967 for Mbps
bw_unit	The bandwidth unit (string). Valid values: `“gbps”`, `“mbps”`.

Returns

`Success` or `Error.`

python_ospf_put_stub_config()

Sets the stub area configuration.

Syntax

`python_ospf_put_stub_config`(*dict_stub, vrf_name*)

where:

Parameter	**Description**
dict_stub	The dictionary containing the stub area configuration details.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

The *dict_stub* dictionary contains the following variables:

Parameter	**Description**
area-id	The area ID IP address (string). A valid IP address.
stub	Whether to enable the stub area configuration (string). Valid values: `“Enable”`, `”Disable”`.
no_summary	Whether to not inject summary routes into the stub configuration (string). Valid values: `“Enable”`, `”Disable”`.

Returns

`Success` or `Error.`

python_ospf_put_clear_config()

Sets the remove commands for process, statistics, traffic statistics and neighbors.

Syntax

`python_ospf_put_clear_config`(*dict_clear_cfg, vrf_name*)

where:

Parameter	**Description**
dict_clear_cfg	The dictionary containing the remove commands details.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

The *dict_clear-cfg* dictionary contains the following variables:

Parameter	**Description**
process	Remove the OSPF process configurations (string). Valid values: `“Enable”, ”Disable”`
statistics	Remove the OSPF statistic configurations (string). Valid values: `“Enable”, ”Disable”`
traffic	Remove the OSPF process traffic statistic configurations (string). Valid values: `“Enable”, ”Disable”`
neighbors	Remove the OSPF neighbor configurations (string). Valid values: `“Enable”, ”Disable”`

Returns

`Success` or `Error.`

python_ospf_put_procinfo_config()

Sets the OSPF process information.

Syntax

`python_ospf_put_procinfo_config`(*dict_procinfo, vrf_name*)

where:

Parameter	**Description**
dict_procinfo	The dictionary OSPF process information details.
vrf_name	(Optional) Virtual Routing and Forwarding name (string). Valid values: the VRF name or `“default”`. Default value: `“default”`.

The *dict_procinfo* dictionary contains the following variables:

Parameter	**Description**
routerId	The OPSF router ID in IP address format (string). A valid IP address.
defaultMetric	The default metric cost. An integer from 1-16777214.
distance_all	The administrative distance. An integer from 1-255.
bfd	Whether to enable BFD configuration (string). Valid values: `“Enable”, ”Disable”`
shutdown	Whether to enable the shutdown OSPF process (string). Valid values: `“Enable”, ”Disable”`

Returns

`Success` or `Error.`

PBR Module

This module manages Policy Based Routing (PBR).

To use this module, in the Python file or in the Python interpreter, enter:

`import pbrApi`

class PBR()

This class provides functions to set and get PBR information.

manageGlobalPbr()

Enables or disables PBR.

Syntax

`manageGlobalPbr``(`*enable*`)`

where:

Parameter	Description
enable	Whether PBR is enabled or disabled (boolean). Valid values: `True`, `False`.

Returns

Boolean (`True` on success, otherwise `False`).

managePbrPolicy()

Sets or unsets a policy on an interface.

Syntax

`managePbrPolicy``(`*enable, routemap, ifType, chassisNumber, ifNumber, ifSubNumber*`)`

where:

Parameter	**Description**
enable	Whether PBR is enabled or disabled (boolean). Valid values: `True`, `False`.
routemap	The route map name (string).
ifType	The interface type (string). Valid values: `“ethernet”`, `“vlan”`.
chassisNumber	The chassis number for the ethernet interfaces (integer).
ifNumber	The interface number for the ethernet interfaces (integer).
ifSubNumber	The interface sub-number for the ethernet interfaces (integer).

Returns

Boolean (`True` on success, otherwise `False`).

showPbrStatus()

Returns the global PBR status information.

Syntax

`showPbrStatus``()`

Returns

A dictionary containing PBR status details:

Parameter	**Description**
enable	Whether PBR is enabled or disabled (boolean). Valid values: `True`, `False`.

showPbrPolicyInfo()

Returns all the policies from all the interfaces.

Syntax

```
showPbrPolicyInfo()
```

Returns

A dictionary containing PBR policy details:

Parameter	**Description**
ifname	The interface name (string). Valid values: `“ethernet”, “vlan”`.
routemap	The route map name (string).
vrf_name	The name of the VRF associated to the interface. An integer from 1-64.
status	The policy status (string). Valid values: `“active”,”inactive”`.

Platform Module

This module manages port information.

To use this module, in the Python file or in the Python interpreter, enter:

`import platformApi`

class PortInfo()

The functions in this class get and set properties for all interfaces (Ethernet, LAG, VLAN, loopback, management).

get_interface()

Gets the properties of all interfaces.

Syntax

`get_interface``(`*if_name_or_range*`)`

where:

Parameter	**Description**
if_name_or_range	(Optional) The interface range (string). Default value: `None`.

Returns

A dictionary containing lists of interface properties:

Parameter	**Description**
duplex	The communication method of the interface (string). Valid values: `"auto"`, `"full"`, `"half"`.
if_name	The interface name (string).
mtu	The maximum transmission unit, in bytes. A positive integer from 64-9216. Default value: `1500`.
admin_state	The admin status (string). Valid values: `"up"`, `"down"`.

Parameter	**Description**
mac_addr	The MAC address. A string in the following format: "`xxxx.xxxx,xxxx`".
speed	The communication speed of the interface (string). Valid values: ● `auto` (auto negotiate) ● `10` (10Mb/s) ● `100` (100Mb/s) ● `1000` (1Gb/s) ● `10000` (10Gb/s) ● `40000` (40Gb/s). **Note:** Values can vary based on system configuration.

get_mac()

Gets the MAC address of an interface.

Syntax

`get_mac(`*if_name***`)`**

where:

Parameter	**Description**
if_name	The Ethernet interface name (string). **Note:** The interface must exist.

Returns

A dictionary containing the MAC address:

Parameter	**Description**
mac_addr	The MAC address (string).

set_mac()

Sets the MAC address of the specified interface.

Syntax

`set_mac``(`*if_name, mac_address*`)`

where:

Parameter	**Description**
if_name	The Ethernet interface name (string). **Note:** The interface must exist.
mac_addr	The MAC address (string).

Returns

Boolean (`True` on success, otherwise `False`).

is_enabled()

Returns the operational status of the interface.

Syntax

`is_enabled``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The Ethernet interface name (string). **Note:** The interface must exist.

Returns

A dictionary containing the operational status of the interface:

Parameter	**Description**
is_enabled	The operational status of an interface (string). Valid values: `"up"`, `"down"`.

set_enabled()

Enables or disables the interface flag.

Syntax

`set_enabled``(`*if_name_or_range, flag*`)`

where:

Parameter	Description
if_name_or_range	The Ethernet interface name (string). **Note:** The interface must exist.
flag	The new operational status for the interface (string). Valid values: `"up"`, `"down"`.

Returns

Boolean (`True` on success, otherwise `False`).

is_link_up()

Gets the link status of a specified interface.

Syntax

`is_link_up``(`*if_name*`)`

where:

Parameter	Description
if_name	The interface name (string). **Note:** The interface must exist.

Returns

A dictionary containing the linkage status:

Parameter	Description
is_link_up	The linkage status (string). Valid values: `"up"`, `"down"`.

get_mtu()

Gets the Maximum Transmission Unit (MTU) of a specified interface.

Syntax

`get_mtu``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The interface name (string). **Note:** The interface must exist.

Returns

A dictionary containing the MTU description:

Parameter	**Description**
mtu	The MTU threshold. An integer from 64-9216.

set_mtu()

Sets the Maximum Transmission Unit (MTU) of an interface.

Syntax

`set_mtu``(`*if_name_or_range, mtu*`)`

where:

Parameter	**Description**
if_name_or_range	The interface name (string). **Note:** The interface must exist.
mtu	The maximum transmission unit value. An integer from 64-9216.

Returns

Boolean (`True` on success, otherwise `False`).

get_port_speed()

Gets the speed of the specified interface.

Syntax

`get_port_speed``(`*if_name*`)`

where:

Parameter	Description
if_name	The interface name (string). **Note:** The interface must exist.

Returns

The port speed:

Parameter	Description
speed	The communication speed of the interface (string). Valid values: ● `auto` (auto negotiate) ● `10` (10Mb/s) ● `100` (100Mb/s) ● `1000` (1Gb/s) ● `10000` (10Gb/s) ● `40000` (40Gb/s). **Note:** Values can vary based on system configuration.

set_port_speed()

Sets the speed of a specified interface.

Syntax

`set_port_speed``(`*if_name_or_range, port_speed*`)`

where:

Parameter	**Description**
if_name_or_range	The interface range (string). **Note:** The interface must exist.
port_speed	The communication speed of the interface (string). Valid values: ● `auto` (auto negotiate) ● `10` (10Mb/s) ● `100` (100Mb/s) ● `1000` (1Gb/s) ● `10000` (10Gb/s) ● `25000` (10Gb/s) ● `40000` (40Gb/s) ● `100000` (40Gb/s) ● `auto` **Note:** Values can vary based on system configuration.

Returns

Boolean (`True` on success, otherwise `False`).

get_duplex()

Gets the duplex for the port.

Syntax

`get_duplex``(`*if_name*`)`

where:

Parameter	Description
if_name	The interface name (string). **Note:** The interface must exist.

Returns

The duplex of the interface:

Parameter	Description
duplex	The duplex mode of the interface (string). Valid values: ”`auto`”, ”`full`”, ”`half`”.

set_duplex()

Sets the duplex mode on a specified interface.

Syntax

`get_duplex``(`*if_name, duplex*`)`

where:

Parameter	Description
if_name	The interface name (string). **Note:** The interface must exist.
duplex	The duplex mode to be applied (string). Valid values: ”`auto`”, ”`full`”, ”`half`”.

Returns

Boolean (`True` on success, otherwise `False`).

class PortStatistics()

This class provides a function that gets physical port statistics.

get_stats()

Gets the port statistics for the specified interface.

Syntax

`get_stats(`*if_name***`)`**

where:

Parameter	Description
if_name	The interface name (string).

Returns

The following packet statistics:

Parameter	Description
good_pkts_sent	Number of sent packets (integer).
in_un_pkts	Number of received unicast packets (integer).
bad_crc	Number of bad CRCs (integer).
brdc_pkts_rcv	Number of received broadcast packets (integer).
mc_pkts_sent	Number of sent multicast packets (integer).
undersize_pkts	Number of undersize packets (integer).
mc_pkts_rcv	Number of received multicast packets (integer).
in_discards	Number of discarded in packets (integer).
good_octets_rcv	Number of received octets (integer).
oversize_pkts	Number of oversize packets (integer).
brdc_pkts_sent	Number of sent broadcast packets (integer).
good_octets_sent	Number of sent octets (integer).
out_uc_pkts	Number of sent unicast packets (integer).
good_pkts_rcv	Number of received packets (integer).

clear_stats()

Resets statistics for the specified switch interface.

Note: This command is available only for non-aggregated switch interfaces.

Syntax

`clear_stats``(`*if_name*`)`

where:

Parameter	Description
if_name	The interface name (string).

Returns

Boolean (`True` on success, otherwise `False`).

class TransInfo()

This class provides information about transceivers.

get_transceiver()

Gets a list of transceiver properties.

Syntax

`get_transceiver``(`*if_name*`)`

where:

Parameter	Description
if_name	(Optional) The interface name (string). **Note:** The interface must exist.

Returns

A dictionary containing lists of transceiver properties:

Parameter	Description
if_name	The interface name (string).
installed	Whether the transceiver is present (string). Valid values: `"Present"`, `"Not Present"`. Default value: `"Not Present"`.
TX Enable	The data transfer status (string.) Valid values: `"Enabled"`.
type	Type of transceiver (string).
state	Transceiver status (string). Valid values: `"Enabled"`, `"Disabled"`.
link	Interface link status (string). Valid values: `"Up"`, `"Down"`.
vendor	Vendor of transceiver (string).
part_number	Part number of transceiver (string).
serial number	Serial number of transceiver (string).
rev	Revision of transceiver (string).
volts	Volts of transceiver (string).
temperature	Current transceiver temperature, in Celsius (string).
approval	Approval status. The transceiver must support the configuration of the interface in order to be approved. Valid values: `"Approved"`, `"Unapproved"`.

Private VLAN Module

This module manages Private VLAN properties.

To use this module, in the Python file or in the Python interpreter, enter:

`import pvlanApi`

class Pvlan()

This class provides functions to get and set Private VLAN configurations.

manageGlobalPvlan()

Globally enables or disables the private VLAN.

Syntax

`manageGlobalPvlan``(`*enable*`)`

where:

Parameter	**Description**
enable	The global status of the private VLAN (boolean). Valid values: `True` to enable or `False` to disable.

Returns

Boolean (`True` on success, otherwise `False`).

managePvlan()

Creates or deletes a private VLAN.

Syntax

`managePvlan``(`*pvlanType, vlanID, createPvlan*`)`

where:

Parameter	**Description**
pvlanType	The private VLAN type (string). Valid values: `”primary”`, `”community”`, `”isolated”`.
vlanID	The VLAN value. An integer from 2-4093.
createPvlan	Whether the private VLAN is created or deleted. Valid values: `True`, to create the private VLAN or `False`, to delete the private VLAN.

Returns

Boolean (`True` on success, otherwise `False`).

managePvlanAssoc()

Creates or deletes a private VLAN association.

Syntax

`managePvlanAssoc``(`*primaryVlanID, secondaryVlanID, createAssoc>*`)`

where:

Parameter	**Description**
primaryVlanID	The primary VLAN value. An integer from 2-4093.
secondaryVlanID	The secondary VLAN value. An integer from 2-4093.
createAssoc	Whether the private VLAN association is created or deleted (boolean). Valid values: `True`, to create the private VLAN association or `False`, to delete the private VLAN association.

Returns

Boolean (`True` on success, otherwise `False`).

setPortMode()

Enables or disables the private VLAN mode on an interface.

Syntax

`setPortMode``(`*ifType, chassisNumber, ifNumber, ifSubNumber, enable*`)`

where:

Parameter	**Description**
ifType	The interface type (string). Valid values: `“ethernet”` for Ethernet interfaces, `“po”` for LAG interfaces.
chassisNumber	The chassis number for the Ethernet interfaces (integer). For example: `1` for Ethernet1/2, or `None` for Link Aggregation Group (LAG) interfaces.
ifNumber	The interface number for LAG interfaces or Ethernet interfaces (integer). For example: `1` for LAG 1 interface, `2` for Ethernet 1/2.
IfSubNumber	The sub-interface number for the Ethernet interfaces (integer). For example: `3` for Ethernet1/50/3, or `None` for LAG interfaces.
enable	The status of the private VLAN mode (boolean). Valid values: `True`, to enable the private VLAN mode or `False` to disable the private VLAN mode.

Returns

Boolean (`True` on success, otherwise `False`).

managePortMapping()

Creates or removes a Private VLAN port-mapping.

Syntax

`managePortMapping``(`*ifType, chassisNumber, ifNumber, ifSubNumber, primaryVlanID, enable*`)`

where:

Parameter	**Description**
ifType	The interface type (string). Valid values: `“ethernet”` for Ethernet interfaces, `“po”` for LAG interfaces.
chassisNumber	The chassis number for the Ethernet interfaces (integer). For example: `1` for Ethernet1/2, or `None` for Link Aggregation Group (LAG) interfaces.
ifNumber	The interface number for LAG interfaces or Ethernet interfaces (integer). For example: `1` for LAG 1 interface, `2` for Ethernet 1/2.
IfSubNumber	The sub-interface number for the Ethernet interfaces (integer). For example: `3` for Ethernet1/50/3, or `None` for LAG interfaces.
enable	The status of the private VLAN mode (boolean). Valid values: `True`, to enable the private VLAN mode or `False` to disable the private VLAN mode.
ifType	The interface type (string). Valid values: `“ethernet”` for Ethernet interfaces, `“po”` for LAG interfaces.

Returns

Boolean (`True` on success, otherwise `False`).

managePortAssociation()

Creates or removes a private VLAN association on an interface.

Syntax

`managePortAssociation``(`*ifType, chassisNumber, ifNumber, ifSubNumber, primaryVlanID, secondaryVlanID, enable*`)`

where:

Parameter	Description
ifType	The interface type (string). Valid values: `“ethernet”` for Ethernet interfaces, `“po”` for LAG interfaces.
chassisNumber	The chassis number for the Ethernet interfaces (integer). For example: `1` for Ethernet1/2, or `None` for Link Aggregation Group (LAG) interfaces.
ifNumber	The interface number for LAG interfaces or Ethernet interfaces (integer). For example: `1` for LAG 1 interface, `2` for Ethernet 1/2.
IfSubNumber	The sub-interface number for the Ethernet interfaces (integer). For example: `3` for Ethernet1/50/3, or `None` for LAG interfaces.
enable	The status of the private VLAN mode (boolean). Valid values: `True`, to enable the private VLAN mode or `False` to disable the private VLAN mode.
ifType	The interface type (string). Valid values: `“ethernet”` for Ethernet interfaces, `“po”` for LAG interfaces.
chassisNumber	The chassis number for the Ethernet interfaces (integer). For example: `1` for Ethernet1/2, or `None` for Link Aggregation Group (LAG) interfaces.

Returns

Boolean (`True` on success, otherwise `False`).

showPvlanInfo()

Displays the private VLAN information.

Syntax

`showPvlanInfo``()`

Returns

A dictionary containing private VLAN information:

Parameter	**Description**
vlanID	The VLAN ID. An integer from 2-4093.
type	The private VLAN type (string). Valid values: `“primary”`, `”community”`, `”isolated”`.
primaryVlanID	The primary VLAN ID. An integer from 2-4093. Available only for community and isolated private VLANs.

showPvlanInterfaceInfo()

Displays information related to the Private VLAN interface.

Syntax

`showPvlanInterfaceInfo``()`

Returns

A dictionary containing private VLAN interface information:

Parameter	**Description**
ifName	The interface name (string).
portMode	The interface port mode (string). Valid values: `“trunk”`, `”access”`.
pvlanPortMode	The private VLAN port mode (string). Valid values: `”host”`, `”promiscuous”`, `”configured”`.
vlan	A list of private VLAN IDs configured on the port. An integer from 2-4093.

RADIUS Module

This module manages the Remote Authentication Dial-In User Service (RADIUS) configuration on the switch.

To use this module, in the Python file or in the Python interpreter, enter:

`import hostpRadiusApi`

class Radius()

This class provides functions to get and set RADIUS configurations.

get_global_key()

Checks if a RADIUS global authentication key is configured.

Syntax

`get_global_key``()`

Returns

The possible values (string). Valid values: `“configured”`, `“not configured”`.

set_global_key()

Configures the RADIUS global authentication key.

Syntax

`set_global_key``(`*key, key_from*`)`

where:

Parameter	**Description**
key	The RADIUS authentication key (string).
key_form	(Optional) The encryption method of the authentication key (integer). Valid values: `0` (clear text) or `7` (encrypted).

Returns

Boolean (`True` on success, otherwise `False`).

get_global_retransmit()

Displays the number of retries the switch will make to establish a connection with a RADIUS server after the initial attempt failed.

Syntax

`get_global_retransmit``()`

Returns

An integer from 0-5.

set_global_retransmit()

Configures the number of retries the switch will make to establish a connection with a RADIUS server after the initial attempt failed.

Syntax

`set_global_restransmit``(`*retransmit*`)`

where:

Parameter	**Description**
retransmit	The number of retries. An integer from 0-5.

Returns

Boolean (`True` on success, otherwise `False`).

get_global_timeout()

Displays the amount of time, in seconds, before a RADIUS server connection attempt is considered to have failed.

Syntax

`get_global_timeout``()`

Returns

An integer from 1 - 60.

set_global_timeout()

Configures the amount of time, in seconds, before a RADIUS server connection attempt is considered to have failed.

Syntax

`set_global_timeout``(`*timeout*`)`

where:

Parameter	Description
timeout	The time interval, in seconds, after which the RADIUS server will timeout. An integer from 1-60.

Returns

Boolean (`True` on success, otherwise `False`).

get_host()

Displays information about an already configured RADIUS server.

Syntax

`get_host``(`*ip_addr*`)`

where:

Parameter	Description
ip_addr	The hostname or IP address of the RADIUS server (string).

Returns

A dictionary showing RADIUS server information:

Parameter	Description
ip_addr	The hostname or IP address of the RADIUS server (string).
auth-port	The port used for authentication. An integer from 1-65535.
acct-port	The port used for accounting. An integer from 1-65535.
retransmit	The number of retries the switch will make to establish a connection with a RADIUS server after the initial attempt failed. An integer from 0-5.

Parameter	Description
timeout	The amount of time, in seconds, before a RADIUS server connection attempt is considered to have failed. An integer from 1-60.
key	The status of the RADIUS server authentication key (string). Valid values: `“configured”`, `”not configured”`.

get_all_hosts()

Displays information about all configured RADIUS servers.

Syntax

```
get_all_hosts()
```

Returns

A list of dictionaries, each showing RADIUS server information:

Parameter	Description
ip_addr	The hostname or IP address of the RADIUS server (string).
auth-port	The port used for authentication. An integer from 1-65535.
acct-port	The port used for accounting. An integer from 1-65535.
retransmit	The number of retries the switch will make to establish a connection with a RADIUS server after the initial attempt failed. An integer from 0-5.
timeout	The amount of time, in seconds, before a RADIUS server connection attempt is considered to have failed. An integer from 1-65535.
key	The status of the RADIUS server authentication key (integer). Valid values: `“configured”`, `”not configured”`.

set_host()

Configures a RADIUS server.

Syntax

`set_host(`*ip_addr, auth_port, acct_port, retransmit, timeout, key, key_form***`)`**

where:

Parameter	**Description**
ip_addr	The hostname or IP address of the RADIUS server (string).
auth_port	(Optional) The port used for authentication. An integer from 1-65535.
acct_port	(Optional) The port used for accounting. An integer from 1-65535.
retransmit	(Optional) The number of retries the switch will make to establish a connection with a RADIUS server after the initial attempt failed. An integer from 0-5.
timeout	(Optional) The amount of time, in seconds, before a RADIUS server connection attempt is considered to have failed. An integer from 1-60.
key	(Optional) The status of the RADIUS server authentication key (string). Valid values: `“configured”`, `”not configured”`.
key_form	(Optional) The method of encryption for the authentication key (integer). Valid values: `0` (clear text), `7` (encrypted).

Returns

Boolean (`True` on success, otherwise `False`).

delete_host()

Deletes an already configured RADIUS server.

Syntax

`delete_host``(`*ip_addr*`)`

where:

Parameter	**Description**
ip_addr	The hostname or IP address of the RADIUS server (string).

Returns

Boolean (`True` on success, otherwise `False`).

get_group()

Displays information about an already configured RADIUS server group.

Syntax

`get_group``(`*group_name*`)`

where:

Parameter	**Description**
group_name	The name of the RADIUS server group (string).

Returns

A dictionary showing information about the specified RADIUS server group:

Parameter	**Description**
group_name	The name of the RADIUS server group (string).
vrf_name	The VRF instance for the RADIUS server group (string).
hosts	Information about the RADIUS servers members of the specified group. Return values are the same as for get_all_hosts().
source_interface	The switch interface to connect to the RADIUS server (string). Valid values: the interface name (for example, `“Ethernet1/12”`) or `”not configured”`.

get_all_groups()

Displays information about all configured RADIUS server groups.

Syntax

`get_all_groups``()`

Returns

A list of dictionaries, each showing information about a specific RADIUS server group:

Parameter	**Description**
group_name	The name of the RADIUS server group (string).
vrf_name	The VRF instance for the RADIUS server group (string).
hosts	Information about the RADIUS servers members of the specified group. Return values are the same as for get_all_hosts().
source_interface	The switch interface to connect to the RADIUS server (string). Valid values: the interface name (for example, `“Ethernet1/12”`) or `”not configured”`.

add_group()

Configures a RADIUS server group.

Syntax

`add_group``(`*group_name*`)`

where:

Parameter	**Description**
group_name	The name of the RADIUS server group (string).

Returns

Boolean (`True` on success, otherwise `False`).

add_server_to_group()

Adds a RADIUS server to a RADIUS server group.

Syntax

`add_server_to_group``(`*group_name, server_ip*`)`

where:

Parameter	**Description**
group_name	The name of the RADIUS server group (string).
server_ip	The IP address of the RADIUS server (string).

Returns

Boolean (`True` on success, otherwise `False`).

set_group_vrf()

Configures the VRF instance for the RADIUS server group.

Syntax

`set_group_vrf``(`*group_name, vrf_name*`)`

where:

Parameter	**Description**
group_name	The name of the RADIUS server group (string).
vrf_name	The name of the VRF instance (string).

Returns

Boolean (`True` on success, otherwise `False`).

set_group_source_interface()

Configures the source switch interface to connect to the RADIUS server group.

Syntax

`set_group_source_interface``(`*group_name, source_interface*`)`

where:

Parameter	**Description**
group_name	The name of the RADIUS server group (string).
source_interface	The name of the switch interface (string). For example: `“Ethernet1/12”.`

Returns

Boolean (`True` on success, otherwise `False`).

delete_group()

Deletes the specified RADIUS server group.

Syntax

`delete_group``(`*group_name*`)`

where:

Parameter	**Description**
group_name	The name of the RADIUS server group (string).

Returns

Boolean (`True` on success, otherwise `False`).

Route Module

This module manages routes.

To use this module, in the Python file or in the Python interpreter, enter:

```
import routeApi
```

class Route()

This class provides functions that manage static routes.

set_route()

Adds a static IPv4 route for a subnet mask.

Syntax

`set_route(`*ip_addr, ip_prefix_len, ip_gw, dist, tag, desc, vrf_name, if_name*`)`

where:

Parameter	**Description**
ip_addr	The IPv4 address of the route.
ip_prefix_len	The IP prefix length. An integer from 0-32, with `0` being the default gateway.
ip_gw	The gateway IP address.
dist	The distance value for the route. An integer from 1-255. Default value: `1`.
tag	The tag value. An integer from 0-4294967295.
desc	Description of the static route (string).
vrf_name	The VRF name (string). **Note:** The named VRF must exist.
if_name	Interface name used for communication (string). **Note:** The interface must exist.

Returns

Boolean (`True` on success, otherwise `False`).

delete_route()

Deletes a static IPv4 route.

Syntax

`delete_route(`*ip_addr, ip_prefix_len, ip_gw, dist, tag, desc, vrf_name, if_name*`)`

where:

Parameter	**Description**
ip_addr	The IPv4 address of the route.
ip_prefix_len	The IP prefix length. An integer from 0-32, with `0` being the default gateway.
ip_gw	The gateway IP address.
dist	The distance value for the route. An integer from 1-255. Default value: `1`.
tag	The tag value. An integer from 0-4294967295.
desc	Description of the static route (string)
vrf_name	The VRF name (string). **Note:** The named VRF must exist.
if_name	Interface name used for communication (string). **Note:** The interface must exist.

Returns

Boolean (`True` on success, otherwise `False`).

get_route()

Gets all IPv4 routes from the routing table.

Syntax

`get_route(`*vrf_name, ip_dest, ip_prefix_len***`)`**

where:

Parameter	**Description**
vrf_name	The VRF name (string). **Note:** The named VRF must exist.
ip_dest	(Optional) The destination IP address (string). Default value: `None`.
ip_prefix_len	(Optional) The IP prefix length. An integer from 0-32, with `0` being the default gateway.

Returns

A list of routes with the following details:

Parameter	**Description**
tag	The tag value. An integer from 0-4294967295.
dist	The distance value for the route; an integer from 1-255. Default value: `1`.
ip_gw	The gateway IP address.
ip_addr	The IPv4 address of the route.
ip_prefix_len	The IP prefix length. An integer from 0-32, with `0` being the default gateway.
desc	Description of the static route (string).
vrf_name	The VRF name (string).
if_name	Interface name used for communication (string).

Routemap Module

This module manages Routemaps.

To use this module, in the Python file or in the Python interpreter, enter:

```
import routemapApi
```

class Routemap()

This class manages Routemaps.

get_all_routemap_entry()

Gets configured routemaps on switch.

Syntax

```
get_all_routemap_entry()
```

Returns

The list of available routemaps.

Security Mode Module

This module manages the Security Mode configuration on the switch.

To use this module, in the Python file or in the Python interpreter, enter:

```
import secModeApi
```

class SecModeApi()

This class provides functions to get and set Security Mode configurations.

get_current_security_mode()

Displays the current Security Mode.

Syntax

```
get_current_security_mode()
```

Returns

A string showing the current security mode. Valid values: `“secure_mode”`, `“legacy_mode”`.

get_new_security_setting()

Displays what security mode is configured to run on the switch after reloading.

Syntax

```
get_new_security_setting()
```

Returns

A string showing the current security mode. Valid values: `“secure_mode”`, `“legacy_mode”`.

set_security_mode()

Configures what security mode to run on the switch after reloading.

Syntax

`set_security_mode``(`*mode*`)`

where:

Parameter	Description
mode	The security mode (string). Valid values: `“secure_mode”, “legacy_mode”`.

Returns

Boolean (`True` on success, otherwise `False`).

System Module

This module manages and monitors the system and the client-server connection.

To use this module, in the Python file or in the Python interpreter, enter:

```
import systemApi
```

Top-Level System Functions()

The following functions are special and are not part of a class.

client_connect()

Establishes the SMI client-server connection.

Note: This must be the first function you call in any CNOS Python script.

Syntax

```
client_connect()
```

Returns

Boolean (`True` on success, otherwise `False`).

client_disconnect()

Ends the SMI client-server connection and frees up related data structures and global memory.

Note: This must be the last function you call in any CNOS Python script.

Syntax

```
client_disconnect()
```

Returns

Boolean (`True` on success, otherwise `False`).

class SystemInfo()

This class provides functions to retrieve basic system properties.

get_systemInfo()

Returns switch type and version number.

Syntax

```
get_systemInfo()
```

Returns

A dictionary containing the switch type and version number:

Parameter	**Description**
switch_type	The switch platform type (string).
fw_version	The version number of the firmware running on the switch (string).

get_env_fan()

Returns environment fan information for the switch.

Syntax

```
get_env_fan()
```

Returns

One or more dictionaries with fan properties:

Parameter	**Description**
fan_id	A dictionary containing the following: ● `“name”` ● `“module”` ● `“air-flow”` ● `“speed-percentage”` ● `“speed-rpm”`
name	The name of the fan (string).
module	The module type (integer).
air-flow	The air flow direction (string). Valid values: `“Front-to-Back”`, `“Back-to-Front”`, `“Not Installed”`.
speed-percent	The percent of Rotations Per Minute. An integer from 0-100.
speed-rpm	The speed in RPMs (integer).

get_env_power()

Returns power supply information for the switch.

Syntax

```
get_env_power()
```

Returns

One or more dictionaries with power supply properties:

Parameter	Description
power_id	A dictionary containing the following: ● `“Name”` ● `“Manufacturer”` ● `“Model”` ● `“State”`
name	The power supply name (string).
manufacturer	The power supply manufacturer (string).
model	The power supply model (string).
state	The power supply state (string).

get_env_temperature()

Returns power supply information for the switch.

Syntax

```
get_env_temperature()
```

Returns

A dictionary with switch temperature properties:

Parameter	Description
cpu local	A dictionary containing the following:
ambient	A dictionary containing the following: ● `temp`: the temperature, in Celsius (integer) ● `state`: the power supply state (string). Valid values: `”OK”`, `”FAULT”`

Parameter	Description
hotspot	A dictionary containing the following: ● `temp`: the temperature, in Celsius (integer) ● `state`: the power supply state (string). Valid values: `”OK”,”FAULT”`
temperature threshold	A dictionary containing the following: ● `system warning`: the temperature at which a system warning is issued ● `system shutdown:` the temperature at which the system will automatically shut down ● `system set point:` the system set point temperature

get_system_serial_num()

Returns the serial number of the switch.

Syntax

`get_system_serial_num``()`

Returns

The system serial number:

Parameter	Description
serial_num	The system serial number (string).

get_system_inventory()

Returns system inventory details.

Syntax

```
get_system_inventory()
```

Returns

A dictionary containing system inventory information:

Parameter	Description
uptime	The system uptime, in seconds.
name	The system name.
service LED	Whether or not the Service LED is enabled (string). Valid values: `“Enabled”`, `“Disabled”`.
sysObjID	Lenovo product MIB (string).
description	The system description.
model	The system model.
manufacture date	The system Manufacture Date.
serial number	The system Serial Number.
pcb assembly	The system PCB Assembly.
electronic serial number	The system Electronic Serial Number.
firmware revision	The system Firmware Revision.
software revision	The system Software Revision.
uuid	The system UUID.
last reset reason	The system last reset reason.

get_hostname()

Returns the host name of the system.

Syntax

```
get_hostname()
```

Returns

The system host name:

Parameter	Description
hostname	The system host name. A string up to 64 characters long.

set_hostname()

Sets the host name of the system.

Syntax

`set_hostname``(`*hostname_in*`)`

where:

Parameter	**Description**
hostname_in	The system host name. A string up to 64 characters long.

Returns

Boolean (`True` on success, otherwise `False`).

get_system_core()

Gets system core details.

Syntax

`get_system_core``()`

Returns

A dictionary containing system core details:

Parameter	**Description**
file_id	A dictionary containing elements `file_name` and `date`.
file_name	The file name (string).
date	The date of the last file change. A string in the following format: `"YYYY-MM-DD HH:MM:SS"`.

class ClockInfo()

This class provides functions to manage system clock information.

get_clock_date()

Gets the system date.

Syntax

`get_clock_date``()`

Returns

A dictionary containing system date details:

Parameter	Description
date	The current date of the system. A string in the following format: `"HH:MM:SS Timezone Week Month Day Year"`.

set_clock_date()

Sets the system date.

Syntax

`set_clock_date`(*time, day, mon, year*)

where:

Parameter	Description
time	The system time. A string in the following format: `"HH:MM:SS"`.
day	The day of the current date. An integer from 1-31.
month	The month of the current date (string). Valid values: one of the 12 months. **Note:** Insert at least three characters. For example: “`Apr`” or “`April`”.
year	The year of the current date. An integer from 2000-2030.

Returns

Boolean (`True` on success, otherwise `False`).

set_clock_format()

Sets the clock format of the system.

Syntax

`set_clock_format`(*clock_format*)

where:

Parameter	**Description**
clock_format	The clock format: 12-hour or 24-hour (integer). Valid values: `12` or `24`.

Returns

Boolean (`True` on success, otherwise `False`).

set_clock_timezone()

Sets the clock time zone of system.

Syntax

`set_clock_timezone`(*timezone, offsethour, offsetmin*)

where:

Parameter	**Description**
timezone	The time zone of current date. A string of 3 to 8 alphanumeric characters. For example: `"PST"`, `"MST"`, `"CST"`, `"EST"`.
offsethour	The hours offset from UTC. An integer from -23 to 23.
offsetmin	The minutes offset from UTC. An integer from 0-59.

Returns

Boolean (`True` on success, otherwise `False`).

set_clock_summertime()

Sets the summertime of the system.

Syntax

`set_clock_summertime`(*clock*)

where:

Parameter	Description
timezone	The time zone of current date. A string of 3 to 8 alphanumeric characters. For example: `"PST"`, `"MST"`, `"CST"`, `"EST"`.
startweek	The week number to start. An integer from 1-5. **Note:** `1` is the first week and `5` the last week.
startweekday	The week day to start (string). Valid values: one of the seven weekdays. **Note:** Insert at least three characters. For example: `“Mon”` or `“Monday”`.
startmonth	The month day to start (string). Valid values: one of the 12 months. **Note:** Insert at least three characters. For example: `“Apr”` or `“April”`.
starttime	The time to start. A string in the following format: `"HH:MM"`.
endweek	The week number to end. An integer from 1-5. **Note:** `1` is the first week and `5` the last week.
endweekday	The week day to end (string). Valid values: one of the seven weekdays. **Note:** Insert at least three characters. For example: `“Mon”` or `“Monday”`.
endmonth	The month day to end (string). Valid values: one of the 12 months. **Note:** Insert at least three characters. For example: `“Apr”` or `“April”`.
endtime	The time to end. A string in the following format: `"HH:MM"`.
offsetmin	The offset to add in minutes. An integer from 1-1440.

Returns

Boolean (`True` on success, otherwise `False`).

TACACS+ Module

This module manages the Terminal Access Controller Access-Control System Plus (TACACS+) configuration on the switch.

To use this module, in the Python file or in the Python interpreter, enter:

`import hostpTacacsApi`

class Tacacs()

This class provides functions to get and set TACACS+ configurations.

get_feature_status()

Checks if TACACS+ is enabled on the switch.

Syntax

`get_feature_status``()`

Returns

The status of the TACACS+ feature (string). Valid values: `“enable”`, `“disable”`.

set_feature_enabled()

Enables TACACS+ on the switch.

Syntax

`set_feature_enabled``()`

Returns

Boolean (`True` on success, otherwise `False`).

set_feature_disabled()

Disables TACACS+ on the switch.

Syntax

`set_feature_disabled``()`

Returns

Boolean (`True` on success, otherwise `False`).

get_global_key()

Checks if a TACACS+ global encryption key is enabled.

Syntax

`get_global_key``()`

Returns

The status of the TACACS+ global encryption key (string).
Valid values: `”configured”`, `“not configured”`.

set_global_key()

Configures a TACACS+ global encryption key.

Syntax

`set_global_key``(`*key, key_from*`)`

where:

Parameter	**Description**
key	The TACACS+ encryption key (string).
key_from	(Optional) The type of TACACS+ encryption key (integer). Valid values: `0` (clear text password) or `7` (encrypted password). **Note:** The default encryption type is clear text password.

Returns

Boolean (`True` on success, otherwise `False`).

get_host()

Displays information about a configured TACACS+ server.

Syntax

`get_host``(`*ip_addr*`)`

where:

Parameter	**Description**
ip_addr	The address of the TACACS+ server (string).

Returns

A dictionary showing information about the specified TACACS+ server.

Parameter	**Description**
ip_addr	The address of the TACACS+ server (string). Valid values: any IPv4 address, IPv6 address or hostname.
port	The port used to connect to TACACS+ server. An integer from 1-65535.
key	The status of the server encryption key configuration (string). Valid values: `“configured”,”not configured”`.

get_all_hosts()

Displays information about all configured TACACS+ servers.

Syntax

`get_all_hosts``()`

Returns

A list of dictionaries, each showing information about a specific TACACS+ server:

Parameter	**Description**
ip_addr	The address of the TACACS+ server (string). Valid values: any IPv4 address, IPv6 address or hostname.
port	The port used to connect to TACACS+ server. An integer from 1-65535.
key	The status of the server encryption key configuration (string). Valid values: `“configured”,”not configured”`.

set_host()

Configures a TACACS+ server.

Syntax

`set_host``(`*ip_addr, port, key, key_form*`)`

where:

Parameter	**Description**
ip_addr	The address of the TACACS+ server (string). Valid values: any IPv4 address, IPv6 address or hostname.
port	(Optional) The port used to connect to TACACS+ server. An integer from 1-65535.
key	(Optional) The server encryption key used to communicate with the TACACS+ server (string).
key_form	(Optional) The type of the server encryption key (string). Valid values: `0` (clear text password) or `7` (encrypted password). **Note:** The default encryption type is clear text password.

Returns

Boolean (`True` on success, otherwise `False`).

delete_host()

Deletes a configured TACACS+ server.

Syntax

`delete_host``(`*ip_addr*`)`

where:

Parameter	**Description**
ip_addr	The address of the TACACS+ server (string). Valid values: any IPv4 address, IPv6 address or hostname.

Returns

Boolean (`True` on success, otherwise `False`).

get_group()

Displays information about a configured TACACS+ server group.

Syntax

`get_group``(`*group_name*`)`

where:

Parameter	Description
group_name	The name of the TACACS+ server group (string).

Returns

A dictionary showing information about the specified TACACS+ server group:

Parameter	Description
group_name	The name of the TACACS+ server group (string).
vrf_name	The name of the Virtual Routing and Forwarding (VRF) instance (string).
hosts	Displays information about the configured TACACS+ servers that are part of the specified group. The details are the same as the Return values for "get_all_hosts()" on page 465.

get_all_groups()

Displays information about all configured TACACS+ server groups.

Syntax

`get_all_groups``()`

Returns

A list of dictionaries, each showing information about a specific TACACS+ server group:

Parameter	Description
group_name	The name of the TACACS+ server group (string).
vrf_name	The name of the Virtual Routing and Forwarding (VRF) instance (string).
hosts	Information about the configured TACACS+ servers that are members of the specified group. The details are the same as the Return values for get_all_hosts().

add_group()

Configures a TACACS+ server group.

Syntax

`add_group``(`*group_name*`)`

where:

Parameter	Description
group_name	The TACACS+ server group name (string).

Returns

Boolean (`True` on success, otherwise `False`).

add_server_to_group()

Adds a configured TACACS+ server to a TACACS+ server group.

Syntax

`add_server_to_group``(`*group_name, server_ip*`)`

where:

Parameter	Description
group_name	The name of the TACACS+ server group (string).
server_ip	The address of the TACACS+ server (string).

Returns

Boolean (`True` on success, otherwise `False`).

set_group_vrf()

Configures the VRF instance used by a TACACS+ group.

Syntax

`set_group_vrf``(`*group_name, vrf_name*`)`

where:

Parameter	Description
group_name	The name of the TACACS+ server group (string).
vrf_name	The name of the VRF instance to be used by the specified TACACS+ server group (string).

Returns

Boolean (`True` on success, otherwise `False`).

delete_group()

Deletes a TACACS+ server group.

Syntax

`delete_group``(`*group_name*`)`

where:

Parameter	Description
group_name	The name of the TACACS+ server group (string).

Returns

Boolean (`True` on success, otherwise `False`).

Telemetry Module

This module manages telemetry information.

To use this module, in the Python file or in the Python interpreter, enter:

`import telemetryApi`

class TelemetryBST_CancelRequest()

This class provides a function to clear a prior Buffer Statistics Tracking (BST) request.

set_bst_cancel_request ()

Cancels the periodic report timer and stops ongoing periodic requests.

Syntax

`set_bst_cancel_request``(`*dict_cancel_request*`)`

where *dict_cancel_request* is a dictionary containing the following elements:

Parameter	**Description**
req-id	The unique request identifier (integer).
cancel-req-id	The request identifier for the cancellation request (integer).

Returns

Boolean (`True` on success, otherwise `False`).

class TelemetryBST_Tracking()

This class provides functions to get and set buffer statistics tracking parameters.

set_bst_tracking()

Sets buffer statistics tracking parameters on the switch.

Syntax

`set_bst_tracking``(`*dict_bst_tracking*`)`

where *dict_bst_tracking* is a dictionary containing the following elements:

Parameter	**Description**
track-peak-stats	Set to `1` to enable peak statistics tracking, `0` to disable this feature. Default value: `0`.
track-ingress-port-priority-group	Set to `1` to enable ingress port priority group tracking, `0` to disable this feature. Default value: `1`.
track-ingress-port-service-pool	Set to `1` to enable ingress port service pool tracking, `0` to disable this feature. Default value: `1`.
track-ingress-service-pool	Set to `1` to enable ingress service pool tracking, `0` to disable this feature. Default value: `1`.
track-egress-port-service-pool	Set to `1` to enable egress port service pool tracking, `0` to disable this feature. Default value: `1`.
track-egress-service-pool	Set to `1` to enable egress service pool tracking, `0` to disable this feature. Default value: `1`.
track-egress-rqe-queue	Set to `1` to enable egress RQE queue tracking, `0` to disable this feature. Default value: `1`.
track-egress-cpu-queue	Set to `1` to enable egress CPU queue tracking, `0` to disable this feature. Default value: `1`.
track-egress-uc-queue	Set to `1` to enable egress unicast queue tracking, `0` to disable this feature. Default value: `1`.
track-egress-mc-queue	Set to `1` to enable egress multicast queue tracking, `0` to disable this feature. Default value: `1`.
track-device	Set to `1` to enable tracking of this device, `0` to disable this feature. Default value: `1`.

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully set the Buffer statistics. ● `FALSE`: Setting Buffer statistics failed
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`

get_bst_tracking()

Gets buffer statistics tracking parameters.

Syntax

`get_bst_tracking``()`

Returns

A dictionary containing the following elements:

Parameter	**Description**
track-peak-stats	Set to `1` to peak statistics tracking, `0` to disable this feature. Default value: `0`.
track-ingress-port-priority-group	Set to `1` to enable ingress port priority group tracking, `0` to disable this feature. Default value: `1`.
track-ingress-port-service-pool	Set to `1` to enable ingress port service pool tracking, `0` to disable this feature. Default value: `1`.
track-ingress-service-pool	Set to `1` to enable ingress service pool tracking, `0` to disable this feature. Default value: `1`.
track-egress-port-service-pool	Set to `1` to enable egress port service pool tracking, `0` to disable this feature. Default value: `1`.
track-egress-service-pool	Set to `1` to enable egress service pool tracking, `0` to disable this feature. Default value: `1`.
track-egress-rqe-queue	Set to `1` to enable egress RQE queue tracking, `0` to disable this feature. Default value: `1`.
track-egress-cpu-queue	Set to `1` to enable egress CPU queue tracking, `0` to disable this feature. Default value: `1`.

Parameter	Description
track-egress-uc-queue	Set to `1` to enable egress unicast queue tracking, `0` to disable this feature. Default value: `1`.
track-egress-mc-queue	Set to `1` to enable egress multicast queue tracking, `0` to disable this feature. Default value: `1`.
track-device	Set to `1` to enable tracking of this device, `0` to disable this feature. Default value: `1`.

class TelemetryBST_Feature()

This class provides functions to get and set buffer statistics feature parameters.

set_bst_feature()

Sets buffer statistics and tracking feature parameters on the switch.

Syntax

`set_bst_feature(`_dict_bst_feature_**`)`**

where *dict_bst_feature* is a dictionary containing the following elements:

Parameter	Description
bst-enable	Set to `1` to enable BST, `0` to disable it. Enabling BST allows the switch to track buffer utilization statistics. Default value: `0`.
send-async-reports	Set to `1` to enable the transmission of periodic asynchronous reports, `0` to disable this feature. Default value: `0`.
collection-interval	The collection interval, in seconds. This defines how frequently periodic reports will be sent to the configured controller. An integer from 10-3600, or `0` for no periodic collection. Default value: `60`.
trigger-rate-limit	The trigger rate limit, which defines the maximum number of threshold-driven triggered reports that the agent is allowed to send to the controller per `trigger-rate-limit-interval`; an integer from 1-5. Default value: `1`.
trigger-rate-limit-interval	The trigger rate limit interval, in seconds; an integer from 10-60. Default value: `1`.

Parameter	**Description**
send-snapshot-on-trigger	Set to 1 to enable sending a complete snapshot of all buffer statistics counters when a trigger happens, 0 to disable this feature. Default value: `1`.
async-full-report	Set to 1 to enable the async full report feature, 0 to disable it. Default value: `0.` When this feature is enabled, the agent sends full reports containing data related to all counters. When the feature is disabled, the agent sends incremental reports containing only the counters that have changed since the last report.

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully set the Buffer statistics feature. ● `FALSE`: Setting Buffer statistics feature failed
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`

get_bst_feature()

Gets buffer statistics and tracking feature parameters.

Syntax

```
get_bst_feature()
```

Returns

A dictionary containing the following elements:

Parameter	**Description**
bst-enable	Set to 1 to enable BST, 0 to disable it. Enabling BST allows the switch to track buffer utilization statistics. Default value: `0`
send-async-reports	Set to 1 to enable the transmission of periodic asynchronous reports, 0 to disable this feature. Default value: `0`

Parameter	Description
collection-interval	The collection interval, in seconds. This defines how frequently periodic reports will be sent to the configured controller. An integer from 10-3600, or `0` for no periodic collection. Default value: `60`.
trigger-rate-limit	The trigger rate limit, which defines the maximum number of threshold-driven triggered reports that the agent is allowed to send to the controller per `trigger-rate-limit-interval`. An integer from 1-5. Default value: `1`
trigger-rate-limit-interval	The trigger rate limit interval, in seconds. An integer from 10-60. Default value: `1`.
send-snapshot-on-trigger	Set to `1` to enable sending a complete snapshot of all buffer statistics counters when a trigger happens, `0` to disable this feature. Default value: `1`.
async-full-report	Set to `1` to enable the async full report feature, `0` to disable it. Default value: `0`. When this feature is enabled, the agent sends full reports containing data related to all counters. When the feature is disabled, the agent sends incremental reports containing only the counters that have changed since the last report.

class TelemetryBST_ClearCgsnDrop()

This class provides a function that manages telemetry buffer statistics tracking congestion drop counters.

get_bst_clear_cgsn_drops()

Resets the congestion drop counters to the default values.

Syntax

```
get_bst_clear_cgsn_drops()
```

Returns

Boolean (`True` on success, otherwise `False`).

class TelemetryBST_Cgsn_Drop_Ctr()

This class provides a function to manage buffer statistics tracking congestion drop counters.

get_bst_cgsn_drop_ctr()

Gets buffer statistics congestion drop counters on the switch based on the request parameters.

Syntax

`get_bst_cgsn_drop_ctr``(`*dict_bst_cdropctr*`)`

where *dict_bst_cdropctr* is a dictionary containing the following elements:

Parameter	**Description**
req-id	The unique request identifier (integer).
request-type	One of the following: ● `"top-drops"`: Show ports with maximum congestion on the switch and their drop-counters ● `"top-port-queue-drops"`: Show top port-queue level drop-counters on the switch ● `"port-queue-drops"`: Show per port-queue level drop-counters on the switch ● `"port-drops"`: Show per-port total drop counters on the switch
request-params	Request parameters; one of the following: ● `count`: Number of ports required in the report. The ports are sorted with the port suffering maximum congestion at the top (integer) ● `queue-type`: Filters the report on the queue type; one of the following strings: – `"ucast"`: Unicast queues – `"mcast"`: Multicast queues – `"all"`: All supported queues ● `interface-list`: Comma-separated list of ports for the congestion drop counter report; an array. A value of all requests all the ports. ● `queue-list`: An array of queue numbers to be considered for the drop report. ● `collection-interval`: (Optional) The period in which the counters are collected from ASIC. An integer from 10-3600. Default value: `0`.

Returns

A dictionary containing the following elements:

Parameter	**Description**
time-stamp	Time of the report generation.
report-type	One of the following: • `“top-drops”`: Show ports with maximum congestion on the switch and their drop-counters • `“top-port-queue-drops”`: Show top port-queue level drop-counters on the switch • `“port-queue-drops”`: Show per port-queue level drop-counters on the switch • `“port-drops”`: Show per-port total drop counters on the switch
congestion-ctr	Congestion counters contents; a list of dictionaries. Depending on the configuration, each dictionary may contain the following values: • `interface`: Interface name • `ctr`: Counter value (string) • `queue-type` (string). Valid values: `“ucast”`, `“mcast”` • `queue-drop-ctr` (integer). Valid values: – `queue number`: an integer from 1-8. – `counter value`: the 64-bit counter value (string)

class TelemetryBST_ClearStats()

This class provides a function to clear buffer statistics and tracking counters.

get_bst_clear_stats()

Clears buffer statistics and tracking counters.

Syntax

`get_bst_clear_stats``()`

Returns

Boolean (`True` on success, otherwise `False`).

class TelemetryBST_ClearThresholds()

This class provides a function to reset the buffer statistics and tracking thresholds to the default values.

get_bst_clear_thresholds()

Resets the buffer statistics and tracking thresholds to the default values.

Syntax

```
get_bst_clear_thresholds()
```

Returns

Boolean (`True` on success, otherwise `False`).

class TelemetryBST_Limits()

This class provides access to the valid range of parameters used to configure the telemetry Buffer Statistics Tracking (BST) features.

get_bst_limits()

Gets the limits associated to the BST realm indexes.

Syntax

```
get_bst_limits()
```

Returns

A dictionary containing the following elements:

Parameter	**Description**
service-pool	The service pool range. An integer from 0-1.
priority-group	The priority group range. An integer from 0-7.
user-queue	The user queue range. An integer from 0-7.
unicast-queue	The absolute unicast queue range. An integer from 0-16383.
multicast-queue	The absolute multicast queue range. An integer from 0-2600.
cpu-queue	The CPU queue range. An integer from 0-47.
rqe-queue	The RQE queue range. An integer from 0-10.
queue-group	The queue group range. An integer from 0-127.

class TelemetryBST_Report()

This class provides a function to retrieve the buffer statistics and tracking report.

get_bst_report()

Gets buffer statistics congestion drop counters on the switch based on the request parameters.

Syntax

`get_bst_report``(`*dict_get_bst_report*`)`

Returns

dict_get_bst_report is a dictionary containing the following elements:

Realm	**Index # 1**	**Index # 2**	**Statistics**
`ingress-port-priority-group`	*interface* (such as `Ethernet1/7`)	`priority-group`	`um-share-buffer-count` `um-head room-buffer-count`
`ingress-port-service-pool`	*interface* (such as `Ethernet1/7`)	`service-pool`	`um-share-buffer-count`
`ingress-service-pool`	`service-pool`		`um-share-buffer-count`
`egress-port-service-pool`	*interface* (such as `Ethernet1/7`)	`service-pool`	`uc-share-buffer-count`, `um-share-buffer-count`, `mc-share-buffer-count`,
`egress-service-pool`	`service-pool`		`um-share-buffercount`, `mc-share-buffer-count`
`egress-rqe-queue`	`rqe-queue`		`rqe-buffer-count`
`include-device`			`data`
`egress-uc-queue`	`unicast-queue`		`uc-threshold`
`egress-mc-queue`	`multi-cast-queue`		`mc-threshold`
`egress-cpu-queue`	`cpu-queue`		`cpu-threshold`

dict_get_bst_report is a dictionary containing the following elements

Parameter	**Description**
realm	Identifies a group of buffer utilization statistic counters (string). One of: ● `ingress-port-priority-group` ● `ingress-port-service-pool` ● `ingress-service-pool` ● `egress-port-service-pool` ● `egress-service-pool` ● `egress-rqe-queue` ● `egress-cpu-queue` ● `egress-mc-queue` ● `egress-uc-queue` ● `device`
interface	The interface name (string). For example: “`Ethernet1/X`”.
service-pool	The service pool range, from 0-1.
priority-group	The priority group range, from 0-7.
user-queue	The user queue range, from 0-7.
unicast-queue	The absolute unicast queue range, from 0-16383.
multicast-queue	The absolute multicast queue range, from 0-2600.
cpu-queue	The CPU queue range, from 0-47.
rqe-queue	The RQE queue range, from 0-10.
queue-group	The queue group range, from 0-127.
threshold	The threshold level, in percentage, from 1-100.
um-share-threshold	The threshold level for unicast and multicast shared buffer utilization, in percentage, from 1-100.
uc-share-threshold	The threshold level for unicast shared buffer utilization, in percentage, from 1-100.
mc-share-threshold	The threshold level for multicast shared buffer utilization expressed in percentage, from 1-100.

class TelemetryBST_Threshold()

This class provides functions to manage and configure buffer statistics and tracking thresholds.

set_bst_threshold()

Configures the buffer statistics and tracking thresholds.

Syntax

`set_bst_threshold``(`*bst_threshold_cfg*`)`

where *bst_threshold_cfg* is a dictionary containing the following elements:

Realm	Index # 1	Index # 2	Statistics

`ingress-port-priority-group`	*interface* (such as `Ethernet1/7`)	`priority-group`	`um-share-buffer-count`
			`um-head room-buffer-count`

`ingress-port-service-pool`	*interface* (such as `Ethernet1/7`)	`service-pool`	`um-share-buffer-count`
`ingress-service-pool`	`service-pool`		`um-share-buffer-count`
`egress-port-service-pool`	*interface* (such as `Ethernet1/7`)	`service-pool`	`uc-share-buffer-count`, `um-share-buffer-count`, `mc-share-buffer-count`,
`egress-service-pool`	`service-pool`		`um-share-buffercount`, `mc-share-buffer-count`
`egress-rqe-queue`	`rqe-queue`		`rqe-buffer-count`
`include-device`			`data`
`egress-mc-queue`	`multicast-queue`		`mc-threshold`
	`interface`	`user-queue`	`mc-threshold`
`egress-uc-queue`	`unicast-queue`		`uc-threshold`
	`interface`	`user-queue`	`uc-threshold`
`egress-cpu-queue`	`cpu-queue`		`cpu-threshold`

Note: The `egress-uc-queue` and `egress-mc-queue` thresholds can be configured using the absolute queue value. You can also specify the pair (interface, user-queue), where the user-queue is a number from 0-7. The following table contains the parameters that can be used as indexes for each realm:

Parameter	**Description**
realm	Identifies a group of buffer utilization statistic counters (string). One of: ● `ingress-port-priority-group` ● `ingress-port-service-pool` ● `ingress-service-pool` ● `egress-port-service-pool` ● `egress-service-pool` ● `egress-rqe-queue` ● `egress-cpu-queue` ● `egress-mc-queue` ● `egress-uc-queue` ● `device`
interface	The interface name (string). For example: `“Ethernet1/X”`.
service-pool	The service pool range, from 0-1.
priority-group	The priority group range, from 0-7.
user-queue	The user queue range, from 0-7.
unicast-queue	The absolute unicast queue range, from 0-16383.
multicast-queue	The absolute multicast queue range, from 0-2600.
cpu-queue	The CPU queue range, from 0-47.
rqe-queue	The RQE queue range, from 0-10.
queue-group	The queue group range, from 0-127.
threshold	The threshold level, in percentage, from 1-100.
um-share-threshold	The threshold level for unicast and multicast shared buffer utilization, in percentage, from 1-100.
uc-share-threshold	The threshold level for unicast shared buffer utilization, in percentage, from 1-100.
mc-share-threshold	The threshold level for multicast shared buffer utilization expressed in percentage, from 1-100.

Returns

A dictionary containing the following elements:

Parameter	Description
return status	Return status of the API call: ● TRUE: Successfully set the Buffer statistics. ● FALSE: Setting Buffer statistics failed
error message	String value containing the reason for the API failure: ● none: When Return Status is TRUE. ● error string: When Return Status is FALSE.

get_bst_threshold()

Gets buffer statistics and tracking threshold values.

Syntax

get_bst_threshold(*dict_get_bst_threshold*)

Returns

dict_get_bst_threshold is a dictionary containing the following elements:

Realm	Index # 1	Index # 2	Statistics
ingress-port-priority-group	*interface* (such as Ethernet1/7)	priority-group	um-share-threshold um-head room-threshold
ingress-port-service-pool	*interface* (such as Ethernet1/7)	service-pool	um-share-threshold
ingress-service-pool	service-pool		um-share-threshold
egress-port-service-pool	*interface* (such as Ethernet1/7)	service-pool	uc-share-threshold, um-share-threshold, mc-share-threshold,
egress-service-pool	service-pool		um-share-threshold, mc-share-threshold
egress-rqe-queue	queue		rqe-threshold
include-device			threshold
egress-uc-queue	unicast-queue		uc-threshold
	interface	user-queue	uc-threshold
egress-mc-queue	multi-cast-queue		mc-threshold
	interface	user-queue	mc-threshold
egress-cpu-queue	queue		cpu-threshold

The following table contains the parameters that can be used as indexes for each realm:

Parameter	**Description**
realm	Identifies a group of buffer utilization statistic counters (string). Valid values: ● `ingress-port-priority-group` ● `ingress-port-service-pool` ● `ingress-service-pool` ● `egress-port-service-pool` ● `egress-service-pool` ● `egress-rqe-queue` ● `egress-cpu-queue` ● `egress-mc-queue` ● `egress-uc-queue` ● `device`
interface	The interface name (string). For example: "`Ethernet1/X`".
service-pool	The service pool range, from 0-1.
priority-group	The priority group range, from 0-7.
user-queue	The user queue range, from 0-7.
unicast-queue	The absolute unicast queue range, from 0-16383.
multicast-queue	The absolute multicast queue range, from 0-2600.
cpu-queue	The CPU queue range, from 0-47.
rqe-queue	The RQE queue range, from 0-10.
queue-group	The queue group range, from 0-127.
threshold	The threshold level, in percentage, from 1-100.
um-share-threshold	The threshold level for unicast and multicast shared buffer utilization, in percentage, from 1-100.
uc-share-threshold	The threshold level for unicast shared buffer utilization, in percentage, from 1-100.
mc-share-threshold	The threshold level for multicast shared buffer utilization expressed in percentage, from 1-100.

class TelemetryDevice_Feature()

This class provides functions to set and retrieve the device system parameters by enabling heartbeat and heartbeat interval.

set_sys_feature()

Configures the system features.

Syntax

`set_sys_feature``(`*dict_sys_feature*`)`

where *dict_sys_feature* is a dictionary containing the following elements:

Parameter	**Description**
heartbeat-enable	When enabled, the Agent asynchronously sends the registration and heartbeat message to the collector; `1` to enable heartbeat, `0` to disable. Default value: `1`.
msg-interval	Determines the interval with which the registration and heartbeat messages are sent to the collector; units of seconds, range 1-600. Default value: `5`.

Returns

A dictionary containing the following elements::

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully set the system features. ● `FALSE`: Setting the system features failed
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

get_sys_feature()

Gets the system features parameters set on the switch.

Syntax

```
get_sys_feature()
```

Returns

A dictionary containing the following elements:

Parameter	**Description**
heartbeat-enable	When enabled, the Agent asynchronously sends the registration and heartbeat message to the collector; `1` to enable heartbeat, `0` to disable. Default value: `1`.
msg-interval	Determines the interval with which the registration and heartbeat messages are sent to the collector; units of seconds, range 1-600. Default value: `5`.

class Telemetry_DeviceProp()

The functions in this class contain methods to retrieve the device switch properties.

get_swprop()

Retrieves the device properties.

Syntax

```
get_swprop()
```

Returns

A dictionary containing the following elements:

Parameter	**Description**
time-stamp	Time of the report generation.
number-of asics	Number of ASICS in the switch (integer).
asic-info	A dictionary consisting of the following values: ● `asic-id`: ASIC identifier on the switch (string) ● `chip-id`: part number of the silicon (string) ● `num-ports`: number of ports available on the switch, managed by this ASIC (integer)
supported features	A list of the features supported by the Agent (string).
network-os	The network operating system currently used on the switch.

Parameter	**Description**
uid	Unique identifier for the switch used by the SDN Controller to map the switch to the nodes existing in their discovery database.
agent-ip	IP address of the switch where the Agent is running (string).
agent-port	TCP port number of the switch at which the Agent is listening (string).
agent-sw-version	Software version number for the Agent (string).

class ClearStats()

This class provides a function to clear the forwarding table utilization counters.

get_fwd_clear_stats()

Clears all forwarding table utilization counters.

Syntax

```
get_fwd_clear_stats()
```

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully cleared table utilization counters ● `FALSE`: Operation failed
error type	The error type: ● `0` - None ● `1` - Invalid Message Body ● `2` - Server Error
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

class ClearThresholds()

This class provides a function to clear the forwarding table utilization thresholds.

get_fwd_clear_thresholds()

Clears all forwarding table utilization thresholds.

Syntax

```
get_fwd_clear_thresholds()
```

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully cleared table utilization thresholds ● `FALSE`: Operation failed
error type	The error type: ● `0` - None ● `1` - Invalid Message Body ● `2` - Server Error
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

class FWD_Feature()

This class provides functions to manage the telemetry forwarding table utilization feature.

set_fwd_feature()

Configures the forwarding table utilization feature parameters.

Syntax

`set_fwd_feature``(`*dict_fwd_feature*`)`

where:

Parameter	**Description**
feature-enable	Enable or disable the feature (integer). Valid values: `0` for disable, `1` for enable. Default value: `0`.
collection-interval	The frequency of collecting statistics, in seconds (integer). Valid values: `0` for no periodic collection, or an integer from 10-3600. Default value: `60`.
trigger-rate-limit-interval	The interval in seconds to perform rate limiting for trigger reports. An integer from 10-60. Default value: `10`.

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully configured table utilization features ● `FALSE`: Operation failed
error type	The error type: ● `0` - None ● `1` - Invalid Message Body ● `2` - Server Error
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

get_fwd_feature()

Gets the forwarding table utilization feature parameters.

Syntax

```
get_fwd_feature()
```

Returns

A dictionary containing the following elements:

Parameter	Description
feature-enable	Enable or disable the feature (integer). Valid values: `0` for disable, `1` for enable. Default value: `0`.
collection-interval	The frequency of collecting statistics, in seconds (integer). Valid values: `0` for no periodic collection, or an integer from 10-3600. Default value: `60`.
trigger-rate-limit-interval	The interval in seconds to perform rate limiting for trigger reports. An integer from 10-60. Default value: `10`.

class FWD_Report()

This class provides a function to access the forwarding table utilization feature counters.

get_fwd_report()

Gets the valid (greater than zero) forwarding table utilization counters for all counter groups (realms).

Syntax

```
get_fwd_report()
```

Returns

A dictionary containing the following elements:

Parameter	Description
method	The method used to deliver information (string). Valid values: `"on-demand"`, `"periodic-report"`, `"trigger-report"`.
time-stamp	The collection time stamp. A string containing the date and time in 24-hour format.
feature	The telemetry feature associated to the counters (string). Valid values: “`fwd-table-utilization`”.
switch-ip	The IP address of device. A string in dotted decimal format.
host-name	The system’s network name. A string up to 64 characters long.

Parameter	**Description**
report	A list of counters per counter group.
realm	The counter group of the report (string). Valid values: `"acl"`, `"arp"`, `"ipv4-rt"`, `"ipv6-rt"`, `"mac"`, `"mc-grp"`, `"nd"`.
usage	The absolute value of current resource utilization for the specific counter (integer). Valid values `0`, `-1`.
max	The absolute value of maximum resource utilization supported for the specific counter group (integer). **Note:** Values can vary based on system configuration.
percent	The percentage value of resource utilization. An integer from 1-100.

class FWD_Threshold()

This class provides functions to manage the forwarding table utilization feature thresholds.

set_fwd_threshold()

Configures the forwarding table utilization feature thresholds.

Syntax

`set_fwd_threshold``(`*fwd_threshold_cfg*`)`

where *fwd_threshold_cfg* contains the following elements:

Parameter	**Description**
realm	The counter group of the report (string). Valid values: `"acl"`, `"arp"`, `"ipv4-rt"`, `"ipv6-rt"`, `"mac"`, `"mc-grp"`, `"nd"`.
threshold	The threshold upon which we must generate triggered reports. An integer from 1-100. Default value: `0`.

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully set table utilization feature thresholds ● `FALSE`: Operation failed
error type	The error type: ● `0` - None ● `1` - Invalid Message Body ● `2` - Server Error
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

get_fwd_threshold()

Gets the forwarding table utilization feature thresholds for all realms.

Syntax

```
get_fwd_threshold()
```

Returns

dict_report is a list of thresholds per counter group:

Parameter	**Description**
realm	The counter group of the report (string). Valid values: `"acl"`, `"arp"`, `"ipv4-rt"`, `"ipv6-rt"`, `"mac"`, `"mc-grp"`, `"nd"`.
threshold	The threshold upon which we must generate triggered reports. An integer from 1-100.

class TelemetryINTF_ClearStats()

This class provides a function to clear the interface statistics counters.

get_intf_clear_stats()

Clears all interface statistics counters.

Syntax

```
get_intf_clear_stats()
```

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully cleared interface statistics counters ● `FALSE`: Operation failed
error type	The error type: ● `0` - None ● `1` - Invalid Message Body ● `2` - Server Error
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

class TelemetryINTF_ClearThresholds()

This class provides a function to clear the interface statistics thresholds.

get_intf_clear_thresholds()

Clears all interface statistics thresholds.

Syntax

```
get_intf_clear_thresholds()
```

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully cleared interface statistics thresholds ● `FALSE`: Operation failed
error type	The error type: ● `0` - None ● `1` - Invalid Message Body ● `2` - Server Error
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

class TelemetryINTF_Feature()

This class provides functions to manage the telemetry interface statistics feature.

set_intf_feature()

Configures the interface statistics feature parameters.

Syntax

`set_intf_feature``(`*dict_intf_feature*`)`

where *dict_intf_feature* contains the following elements:

Parameter	**Description**
feature-enable	Enable or disable the feature (integer). Valid values: `0` for disable, `1` for enable. Default value: `0`.
collection-interval	The frequency of collecting statistics, in seconds (integer). Valid values: `0` for no periodic collection, or an integer from 10-3600. Default value: `60`.
trigger-rate-limit-interval	The interval in seconds to perform rate limiting for trigger reports. An integer from 10-60. Default value: `10`.

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully set interface statistics feature parameters ● `FALSE`: Operation failed
error type	The error type: ● `0` - None ● `1` - Invalid Message Body ● `2` - Server Error
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

get_intf_feature()

Gets the interface statistics feature configuration parameters.

Syntax

```
get_intf_feature()
```

Returns

A dictionary containing the following elements:

Parameter	Description
feature-enable	Enable or disable the feature (integer). Valid values: `0` for disable, `1` for enable. Default value: `0`.
collection-interval	The frequency of collecting statistics, in seconds (integer). Valid values: `0` for no periodic collection, or an integer from 10-3600. Default value: `60`.
trigger-rate-limit-interval	The interval in seconds to perform rate limiting for trigger reports. An integer from 10-60. Default value: `10`.

class TelemetryINTF_Report()

This class provides a function to provide access to the interface statistics counters.

get_intf_report()

Gets the valid (greater than zero) interface utilization counters for all counter groups (realms).

Syntax

```
get_intf_report()
```

Returns

A dictionary containing the following elements:

Parameter	Description
method	The method used to deliver information (string). Valid values: `"on-demand"`, `"periodic-report"`, `"trigger-report"`.
time-stamp	The collection time stamp. A string containing the date and time in 24-hour format.
feature	The telemetry feature associated to the counters (string). Valid values: `“intf-utilization”`.

Parameter	**Description**
switch-ip	The IP address of device. A string in dotted decimal format.
host-name	The system's network name. A string up to 64 characters long.
report	A list of counters per counter group.
realm	The counter group of the threshold (string). Valid values: `"interface-status"`, `"traffic-utilization"`.
data	A dictionary containing utilization data.
inactive-interfaces	The total number of down interface (integer). Valid values: `0`, `-1`.
total-interfaces	The total number of interfaces configured on the switch (integer).
percent	The percentage value of resource utilization. An integer from 1-100. **Note:** Values can vary based on system configuration.
interface	The switch interface (string). Valid values: `"Ethernet1/1"`, `"Vlan10"`, `"mgmt0"`, `"loopback0"`, `"po2"`.
speed	The counter of packets received (integer). Valid values: ● `10000` ● `100000` ● `25000` ● `40000` ● `50000` ● `NA`
rx-packets	The counter of packets received (integer).
rx-bytes	The counter of bytes received (integer).
rx-unicast-packets	The counter of unicast packets received (integer).
rx-multicast-packets	The counter of multicast packets received (integer).
rx-broadcast-packets	The counter of broadcast packets received (integer).
rx-trunk-frames	The counter the tagged frames received (integer).

Parameter	**Description**
rx-discard-errors	The received packets discarded due to errors (integer).
rx-errors	The received packets with errors (integer).
rx-drop-events	The total count of received packets dropped (integer).
rx-down-drops	The received packets discarded due to interface down (integer).
rx-packets-from-0-to-64-bytes	The packets with sizes between 0 and 64 bytes received (integer).
rx-packets-from-65-to-127-bytes	The packets with sizes between 65 and 127 bytes received (integer).
rx-packets-from-128-to-255-bytes	The packets with sizes between 128 and 255 bytes received (integer).
rx-packets-from-256-to-511-bytes	The packets with sizes between 256 and 511 bytes received (integer).
rx-packets-from-512-to-1023-bytes	The packets with sizes between 512 and 1023 bytes received (integer).
rx-packets-from-1024-to-1518-bytes	The packets with sizes between 1024 and 1518 bytes received (integer).
rx-packets-from-1519-to-2047-bytes	The packets with sizes between 1519 and 2047 bytes received (integer).
tx-packets	The counter of packets transmitted (integer).
tx-bytes	The counter of bytes transmitted (integer).
tx-unicast-packets	The counter of unicast packets transmitted (integer).
tx-multicast-packets	The counter of multicast packets transmitted (integer).
tx-broadcast-packets	The counter of broadcast packets transmitted (integer).
tx-trunk-frames	The counter of trunk frames transmitted (integer).
tx-errors	The number of outbound packets that could not be transmitted because of errors (integer).
tx-dropped	The number of packets dropped at TX for any reason (integer).
tx-packets-from-0-to-64-bytes	The packets with sizes between 0 and 64 bytes transmitted (integer).

Parameter	**Description**
tx-packets-from-65-to-127-bytes	The packets with sizes between 65 and 127 bytes transmitted (integer).
tx-packets-from-128-to-255-bytes	The packets with sizes between 128 and 255 bytes transmitted (integer).
tx-packets-from-256-to-511-bytes	The packets with sizes between 256 and 511 bytes transmitted (integer).
tx-packets-from-512-to-1023-bytes	The packets with sizes between 512 and 1023 bytes transmitted (integer).
tx-packets-from-1024-to-1518-bytes	The packets with sizes between 1024 and 1518 bytes transmitted (integer).
tx-packets-from-1519-to-2047-bytes	The packets with sizes between 1519 and 2047 bytes transmitted (integer).

class TelemetryINTF_Threshold()

This class provides functions to manage the interface statistics thresholds.

set_intf_threshold()

Configures the interface statistics feature thresholds.

Syntax

`set_intf_threshold``(`*intf_threshold*`)`

where *intf_threshold* contains the following elements:

Parameter	**Description**
realm	The realm for which we want to configure the threshold (string). Valid values: `"all"`, `"status"`.
threshold	The threshold upon which we must generate triggered reports. An integer from 1-100. Default value: `0`.

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully set interface statistics feature thresholds ● `FALSE`: Operation failed
error type	The error type: ● `0` - None ● `1` - Invalid Message Body ● `2` - Server Error
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

get_intf_threshold()

Gets the interface statistics thresholds for all counter groups.

Syntax

```
get_intf_threshold()
```

Returns

A dictionary containing the following elements:

Parameter	**Description**
realm	The realm for which we want to configure the threshold (string). Valid values: `"all"`, `"status"`.
threshold	The threshold upon which we must generate triggered reports. An integer from 1-100.

class TelemetrySYS_ClearStats()

This class provides a function to clear the system statistics counters.

get_sys_clear_stats()

Clears all system statistics counters.

Syntax

```
get_sys_clear_stats()
```

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully cleared system statistics counters ● `FALSE`: Operation failed
error type	The error type: ● `0` - None ● `1` - Invalid Message Body ● `2` - Server Error
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

class TelemetrySYS_ClearThresholds()

This class provides a function to clear the system statistics thresholds.

get_sys_clear_thresholds()

Clears all system statistics thresholds.

Syntax

```
get_sys_clear_thresholds()
```

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully cleared system statistics thresholds ● `FALSE`: Operation failed
error type	The error type: ● `0` - None ● `1` - Invalid Message Body ● `2` - Server Error
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

class TelemetrySYS_Feature()

This class provides functions to manage the telemetry system statistics feature.

set_sys_feature()

Configures the system statistics feature parameters.

Syntax

`set_sys_feature``(`*dic_sys_feature*`)`

where *dic_sys_feature* contains the following elements:

Parameter	**Description**
feature-enable	Enable or disable the feature (integer). Valid values: `0` for disable, `1` for enable. Default value: `0`.
collection-interval	The frequency of collecting statistics, in seconds (integer). Valid values: `0` for no periodic collection, or an integer from 10-3600. Default value: `60`.
trigger-rate-limit-interval	The interval in seconds to perform rate limiting for trigger reports. An integer from 10-60. Default value: `10`.

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully set system statistics feature parameters. ● `FALSE`: Operation failed
error type	The error type: ● `0` - None ● `1` - Invalid Message Body ● `2` - Server Error
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

get_sys_feature()

Gets the system statistics feature configuration parameters.

Syntax

```
get_sys_feature()
```

Returns

A dictionary containing the following elements:

Parameter	**Description**
feature-enable	Enable or disable the feature (integer). Valid values: `0` for disable, `1` for enable. Default value: `0`.
collection-interval	The frequency of collecting statistics, in seconds (integer). Valid values: `0` for no periodic collection, or an integer from 10-3600. Default value: `60`.
trigger-rate-limit-interval	The interval in seconds to perform rate limiting for trigger reports. An integer from 10-60. Default value: `10`.

class TelemetrySYS_Report()

This class provides a function to access the system statistic feature counters.

get_sys_report()

Gets the system statistics counters for all counter groups.

Syntax

```
get_sys_report()
```

Returns

A dictionary containing the following elements:

Parameter	**Description**
method	The method used to deliver information (string). Valid values: `"on-demand"`, `"periodic-report"`, `"trigger-report"`.
time-stamp	The collection time stamp. A string containing the date and time in 24-hour format.
feature	The telemetry feature associated to the counters (string). Valid values: `“sys-utilization”`.
switch-ip	The IP address of device. A string in dotted decimal format.
host-name	The system’s network name. A string up to 64 characters long.
report	A list of counters per counter group.
realm	The counter group of the report (string). Valid values: `"cpu"`, `"fan"`, `"memory"`, `"power"`, `"system-memory"`, `"process-memory"`, `"temperature"`.
data	A list of dictionaries containing utilization data.
active-fans	The number of fans that are in operation (integer). Valid values: `0`, `-1`.
total-fans	The total number of fans specific counter group (integer).
percent	The percentage value of resource utilization. An integer from 1-100.
average-fan-speed	The average fan speed value of speed in rotations per minute (integer).
average-percent-speed	The average rotations per minute percent value (integer).
warning	The temperature over which an warning event is generated (integer). Default value: `85`.
shutdown	The temperature over which an alert event is generated and the switch is turned off after three alerts (integer). Default value: `95`.

Parameter	**Description**
set-point	The temperature below which the switch is considered to work optimally after a warning or shutdown event has occurred (integer). Default value: `70`.
temp	The temperature sensor values, in Celsius. An integer from 0-100.
state	The temperature sensor status (string). Valid values: `"Ok"`, `"Fault"`.
power1-name	The name of power supply one (string). Default value: `“Power Supply 1”`.
power1-manufacturer	The name of manufacturer for power supply one (string).
power1-model	The name of model for power supply one (string).
power1-state	The state of the power supply one (string). Valid values: `"Normal ON"`, `"12V Output Fault"`.
power1-voltage	The voltage of power supply one. An integer from 1-12.
power1-watts	The power consumed by power supply one (integer).
power2-name	The name of power supply two (string). Default value: `“Power Supply 2”`.
power2-manufacturer	The name of manufacturer for power supply two (string).
power2-model	The name of model for power supply two (string).
power2-state	The state of the power supply two (string). Valid values: `"Normal ON"`, `"22V Output Fault"`.
power2-voltage	The voltage of power supply two. An integer from 1-22.
power2-watts	The power consumed by power supply two (integer).
load-1-min	The system CPU load in the last minute. Valid values 1-100.
load-5-min	The system CPU load in the last five minutes. Valid values 1-100.
load-15-min	The system CPU load in the last 15 minutes. Valid values 1-100.
number-of-cores	The number of cores per processor. An integer from 0-4.
percent	The percentage value of CPU load. An integer from 1-100.
memory-usage	The memory used (integer). Valid values: `0`, `-1`.
total-memory	The the total memory of the device (integer).
percent	The percentage value of memory utilization. An integer from 1-100.

Parameter	**Description**
process	The process name (string).
pid	The process identifier (integer).
mem-alloc	The memory allocated for this process (integer).
stk-size	The stack size in Kilobytes for this process (integer).
rss-mem	The resident set size memory allocated for the process in kilobytes. An accurate representation of how much actual physical memory a process is consuming (integer).

© Copyright Lenovo 2019

class TelemetrySYS_Threshold()

This class provides functions to manage the system statistics feature thresholds.

set_sys_threshold()

Configures the system statistics feature thresholds.

Syntax

`set_sys_threshold``(`*sys_threshold_cfg*`)`

where *sys_threshold_cfg* contains the following elements:

Parameter	**Description**
realm	The counter group of the threshold (string). Valid values: `"all"`, `"cpu"`, `"memory"`.
threshold	The threshold upon which we must generate triggered reports. An integer from 1-100. Default value: `0`.

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully set system statistics feature thresholds ● `FALSE`: Operation failed
error type	The error type: ● `0` - None ● `1` - Invalid Message Body ● `2` - Server Error
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

get_sys_threshold()

Gets the system statistics feature thresholds for all counter groups.

Syntax

```
get_sys_threshold()
```

Returns

A dictionary containing the following elements:

Parameter	Description
realm	The counter group of the threshold (string). Valid values: `"all"`, `"cpu"`, `"memory"`.
threshold	The threshold upon which we must generate triggered reports. An integer from 1-100.

UFP Module

This module manages Unified Fabric Port (UFP).

To use this module, in the Python file or in the Python interpreter, enter:

`import ufpApi`

class UfpCapRx()

This class gets received UFP Capability Discovery TLV information.

ufp_get_cap_rx()

Gets received UFP Capability Discovery TLV information.

Syntax

`ufp_get_cap_rx``(`*if_name*`)`

where:

Parameter	Description
if_name	The Ethernet interface name (string). Valid values: `“ethernet”`.

Returns

A dictionary containing UFP Capability Discovery TLV information:

Parameter	Description
if_name	The Ethernet interface name (string). Valid values: `“ethernet”`.
type	UFP Capability Discovery TLV type in LLDP (integer). Valid value: `127`. Default value: `127`.
len	UFP Capability Discovery TLV string length (integer). Valid value: `7`. Default value: `7`.
oui	Organizationally Unique ID (string). Valid values: `"00-18-b1"`. Default value: `"00-18-b1"`.
subtype	Organizationally Defined Subtype (integer). Valid value: `1`. Default value: `1`.
max_ver	Maximum version supported by the host (integer). Valid value: `1`. Default value: `1`.
oper_ver	The operation version (integer). Valid values: `0` for disable, `1` for enable. Default value: `1`.
cna_req	Whether CNA is in UFP mode (integer). Valid vales: `0` for disable, `1` for enable. Default value: `1`.

Parameter	**Description**
cna_oper	The UFP operational state of CNA (integer). Valid vales: `0` for disable, `1` for enable.
switch_cap	Whether the switch is in UFP mode. Valid vales: `0` for disable, `1` for enable.
switch_oper	The UFP operational state of switch. Valid vales: `0` for disable, `1` for enable.

class UfpCapTx()

This class gets transmitted UFP Capability Discovery TLV information.

ufp_get_cap_tx()

Gets transmitted UFP Capability Discovery TLV information.

Syntax

`ufp_get_cap_tx``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.

Returns

A dictionary containing UFP Capability Discovery TLV information:

Parameter	**Description**
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.
type	UFP Capability Discovery TLV type (integer). Valid value: `127`. Default value: `127`.
len	UFP Capability Discovery TLV string length (integer). Valid value: `7`. Default value: `7`.
oui	Organizationally Unique ID (string). Valid value: `"00-18-b1"`. Default value: `"00-18-b1"`.
subtype	Organizationally Defined Subtype (integer). Valid value: `1`. Default value: `1`.
max_ver	Maximum version supported by host (integer). Valid value: `1`. Default value: `1`.
oper_ver	Operation version (integer). Valid values: `0` for disable, `1` for enable. Default value: `1`.
cna_req	Whether CNA is in UFP mode (integer). Valid vales: `0` for disable, `1` for enable.
cna_oper	The UFP operational state of CNA (integer). Valid vales: `0` for disable, `1` for enable.
switch_cap	Whether the switch is in UFP mode (integer). Valid vales: `0` for disable, `1` for enable. Default value: `1`.
switch_oper	The UFP operational state of the switch (integer). Valid vales: `0` for disable, `1` for enable.

class UfpCdcpRx()

This class gets received UFP CDCP TLV information.

ufp_get_cdcp_rx()

Gets received UFP CDCP TLV information.

Syntax

`ufp_get_cdcp_rx``(`*if_name*`)`

where:

Parameter	Description
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.

Returns

A dictionary containing UFP CDCP TLV information:

Parameter	Description
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.
type	UFP Capability Discovery TLV type (integer). Valid value: `127`. Default value: `127`.
len	UFP Capability Discovery TLV string length. An integer from 8-32.
oui	Organizationally Unique ID (string). Valid value: `"00-18-b1"`. Default value: `"00-18-b1"`.
subtype	Organizationally Defined Subtype (integer). Valid value: `14`. Default value: `14`.
role	The sender role (integer). Valid values `1` for NIC, `0` for switch.
scomp	The S-Component as Port-mapping S-VLAN component (integer). Valid vales: `0` for disable, `1` for enable. Default value: `1`.
chn_cap	S-channel capacity. For example the number of S-channels. An integer from 0-8.
scid	The S-channel ID. An integer from 1-9. Default value: `1`.
svid	The S-channel default VLAN ID. An integer from 2-3999 or 4002-4009. Default value: `1`.

class UfpCdcpTx()

This class gets transmitted UFP CDCP TLV information.

ufp_get_cdcp_tx()

Gets transmitted UFP CDCP TLV information.

Syntax

`ufp_get_cdcp_tx``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.

Returns

A dictionary containing UFP CDCP TLV information:

Parameter	**Description**
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.
type	UFP CDCP TLV type in LLDP (integer). Valid value: `127`. Default value: `127`.
len	UFP CDCP TLV string length string length. An integer from 8-32.
oui	Organizationally Unique ID (string). Valid value: `"00-18-b1"`. Default value: `"00-18-b1"`.
subtype	Organizationally Defined Subtype (integer). Valid value: `14`. Default value: `14`.
role	The sender role (integer). Valid values `1` for NIC, `0` for switch.
scomp	The S-Component as Port-mapping S-VLAN component (integer). Valid vales: `0` for disable, `1` for enable. Default value: `1`.
chn_cap	S-channel capacity. For example the number of S-channels. An integer from 0-8.
scid	The S-channel ID. An integer from 1-9. Default value: `1`.
svid	The S-channel default VLAN ID. An integer from 2-3999 or 4002-4009. Default value: `1`.

class UfpEcp()

This class gets ECP information.

ufp_get_ecp()

Gets UFP ECP information.

Syntax

`ufp_get_ecp(`*if_name***`)`**

where:

Parameter	Description
if_name	The Ethernet interface name (string). Valid value: `"ethernet"`.

Returns

A dictionary containing UFP ECP information:

Parameter	Description
if_name	The Ethernet interface name (string). Valid value: `"ethernet"`.
state	The current link state of ECP (string). Valid values: `"up"`, `"down"`. Default value: `"down"`.
ulpid	The Upper Level Protocol ID in ECP (integer).
chnl_id	The channel ID of ECP in the given interface (integer).

class UfpGlobal()

This class gets and sets UFP global configuration.

ufp_get_config()

Gets UFP global configuration.

Syntax

`ufp_get_config()`

Returns

A dictionary containing UFP global operation information:

Parameter	Description
ena	Whether UFP is enabled or not (string). Valid values: `"yes"`, `"no"`. Default value: `"no"`.

ufp_set_ena()

Enables or disables UFP global operation.

Syntax

`ufp_set_ena``(`*ena*`)`

where:

Parameter	**Description**
ena	Whether UFP is enabled or not (string). Valid values: “`yes`”, “`no`”. Default value: “`no`”.

Returns

Boolean (`True` on success, otherwise `False`).

class UfpGlobalInfo()

This class gets UFP global information.

ufp_get_info()

Gets UFP global operation.

Syntax

`ufp_get_info``()`

Returns

A dictionary containing UFP information:

Parameter	**Description**
ena	Whether UFP is enabled or not (string). Valid values: “`yes`”, “`no`”. Default value: “`no`”.

class UfpIfInfo()

This class gets UFP interface and status information.

ufp_get_if_info()

Gets UFP interface and status information.

Syntax

`ufp_get_if_info``(`*if_name*`)`

where:

Parameter	Description
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.

Returns

A dictionary containing UFP interface information:

Parameter	Description
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.
state	The current state of UFP interface (string). Valid values: `“enable”`, `“disable”`.
vports	The number of enabled vPorts. An integer from 0-8. Default value: `0`.
vpidx	The virtual port index. An integer from 1-8.
status	The vPort status (string). Valid values: `"linkup"`, `"linkdown"`, `"mismatch"`, `"failover"`.

class UfpInterface()

This class gets and sets UFP interface configurations.

ufp_get_interface()

Gets UFP interface configurations.

Syntax

`ufp_get_interface``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The Ethernet interface name (string). Valid value: “`ethernet`”.

Returns

A dictionary containing UFP interface configurations:

Parameter	**Description**
if_name	The Ethernet interface name (string). Valid value: “`ethernet`”.
ena	Whether UFP interface is enabled or not (string). Valid values: “`yes`”, “`no`”. Default value: “`no`”.
qos_mode	The Qos mode (string). Valid values: “`ets`” - for ETS mode, “`bw`” - for Bandwidth mode. Default value: “`bw`”.

ufp_get_all_interface()

Gets UFP configurations for all interfaces.

Syntax

`ufp_get_all_interface``()`

Returns

A dictionary containing UFP interface configurations:

Parameter	**Description**
if_name	The Ethernet interface name (string). Valid value: “`ethernet`”.
ena	Whether UFP interface is enabled or not (string). Valid values: “`yes`”, “`no`”. Default value: “`no`”.
qos_mode	The Qos mode (string). Valid values: “`ets`” - for ETS mode, “`bw`” - for Bandwidth mode. Default value: “`bw`”.

ufp_set_interface()

Sets UFP interface configurations.

Syntax

`ufp_set_interface``(`*if_name, enable,qos_mode*`)`

where:

Parameter	**Description**
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.
ena	Whether UFP interface is enabled or not (string). Valid values: `“yes”`, `“no”`. Default value: `“no”`.
qos_mode	The Qos mode (string). Valid values: `“ets”` - for ETS mode, `“bw”` - for Bandwidth mode. Default value: `“bw”`.

Returns

Boolean (`True` on success, otherwise `False`).

ufp_delete_interface()

Deletes UFP interface configurations.

Syntax

`ufp_delete_interface``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.

Returns

Boolean (`True` on success, otherwise `False`).

class UfpLnkdnRx()

This class gets received UFP linkdown TLV information.

ufp_get_linkdown_rx()

Gets received UFP linkdown TLV information.

Syntax

`ufp_get_linkdown_rx``(`*if_name*`)`

where:

Parameter	Description
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.

Returns

A dictionary containing UFP TLV information:

Parameter	Description
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.
type	The UFP Control TLV type of Linkdown message (integer). Valid value: 2. Default value: `2`.
len	UFP Linkdown TLV string length. An integer from 1-13. Default value: `1`.
flags	Indicates the request or response (integer). Valid values: `0` for request, `1` for acknowledgement. Default value: `1`.
status	The status (integer). Valid values: `0` for success, 1-6 for failure. Default value: `0`.
scid	The S-channel ID. An integer from 2-9.

class UfpLnkdnTx()

This class gets transmitted UFP linkdown TLV information.

ufp_get_linkdown_tx()

Gets transmitted UFP linkdown TLV information.

Syntax

`ufp_get_linkdown_tx``(`*if_name*`)`

where:

Parameter	Description
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.

Returns

A dictionary containing UFP TLV information:

Parameter	Description
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.
type	The UFP Control TLV type of Linkdown message (integer). Valid value: `2`. Default value: `2`.
len	UFP Linkdown TLV string length. An integer from 1-13. Default value: `1`.
flags	Indicates the request or response (integer). Valid values: `0` for request, `1` for acknowledgement. Default value: `0`.
status	The status (integer). Valid values: `0` for success, 1-6 for failure. Default value: `0`.
scid	The S-channel ID. An integer from 2-9.

class UfpLnkupRx()

This class gets received UFP linkup TLV information.

ufp_get_linkup_rx()

Gets received UFP linkup TLV information.

Syntax

`ufp_get_linkup_rx``(`*if_name*`)`

where:

Parameter	Description
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.

Returns

A dictionary containing UFP TLV information:

Parameter	Description
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.
type	The UFP Control TLV type of Linkdown message (integer). Valid value: `3`. Default value: `3`.
len	UFP Linkdown TLV string length. An integer from 1-13. Default value: `1`.
flags	Indicates the request or response (integer). Valid values: `0` for request, `1` for acknowledgement. Default value: `0`.
status	The status (integer). Valid values: `0` for success, 1-6 for failure. Default value: `0`.
scid	The S-channel ID. An integer from 2-9.

class UfpLnkupTx()

This class gets transmitted UFP linkup TLV information.

ufp_get_linkup_tx()

Gets transmitted UFP linkup TLV information.

Syntax

`ufp_get_linkup_tx``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.

Returns

A dictionary containing UFP TLV information:

Parameter	**Description**
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.
type	The UFP Control TLV type of Linkdown message (integer). Valid value: `3`. Default value: `3`.
len	UFP Linkdown TLV string length. An integer from 1-13. Default value: `1`.
flags	Indicates the request or response (integer). Valid values: `0` for request, `1` for acknowledgement. Default value: `0`.
status	The status (integer). Valid values: `0` for success, 1-6 for failure. Default value: `0`.
scid	The link up S-Channel ID. An integer from 2-9.

class UfpPropRx()

This class receives UFP NIC-PROPS TLV information.

ufp_get_prop_rx()

Gets received UFP NIC-PROPS TLV information.

Syntax

`ufp_get_prop_rx``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.

Returns

A dictionary containing UFP NIC-PROPS TLV information:

Parameter	**Description**
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.
type	The UFP Control TLV type of NIC-PROPS message (integer). Valid value: `1`. Default value: `1`.
len	The UFP NIC-PROPS TLV string length. An integer from 3-59. Default value: `3`.
flags	Indicates the request or response (integer). Valid values: `0` for request, `1` for acknowledgement. Default value: `1`.
status	The status (integer). Valid values: ● 0 - for success ● 1 - for unsupported ● 2 - for invalid request ● 3 - for invalid state ● 4 - for insufficient resource ● 5 - for unspecified error ● 6 - for configuration mismatch Default value: `0`.
chnl_type	The channel type (integer). Valid values: `0` for VLAN partition, `1` for S-Tagged. Default value: `1`.
sched_type	The schedule type. Valid values: `0` for QoS ETS, `1` for vPort bandwidth. Default value: `0`.
num_vlan	Maximum VLANs in Partition (integer). Valid Values: `0` for S-Tagged). Default value: `0`.

Parameter	Description
scid	The enabled vPort's S-Channel ID. An integer from 2-9.
svid	The vPort's S-Channel default VLAN ID. An integer from 2-3999 or 4002-4009.
iscsi	Whether the switch is iSCSI capable. Valid values: `0` for no, `1` for yes. Default value: `0`.
host_pri	Whether the vPort is host priority control or not. Valid values: `0` for no, `1` for yes. Default value: `0`.
fcoe	Whether the vPort is FCoE capable. Valid values: `0` for no, `1` for yes. Default value: `0`.
minbw	The minimum guaranteed bandwidth, in percent. An integer from 10-100, or `0` for QoS ETS mode. Default value: `25`.
maxbw	The maximum allowed bandwidth, in percent. An integer from 10-100, or `0` for QoS ETS mode. Default value: `100`.

class UfpPropTx()

This class gets transmitted UFP NIC-PROPS TLV information.

ufp_get_prop_tx()

Gets transmitted UFP NIC-PROPS TLV information.

Syntax

`ufp_get_prop_tx``(`*if_name*`)`

where:

Parameter	**Description**
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.

Returns

A dictionary containing UFP NIC-PROPS TLV information:

Parameter	**Description**
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.
type	The UFP Control TLV type of NIC-PROPS message (integer). Valid value: `1`. Default value: `1`.
len	The UFP NIC-PROPS TLV string length. An integer from 3-59. Default value: `3`.
flags	Indicates the request or response (integer). Valid values: `0` for request, `1` for acknowledgement. Default value: `0`.
status	The status (integer). Valid values: ● 0 - for success ● 1 - for unsupported ● 2 - for invalid request ● 3 - for invalid state ● 4 - for insufficient resource ● 5 - for unspecified error ● 6 - for configuration mismatch Default value: `0`.
chnl_type	The channel type (integer). Valid values: `0` for VLAN partition, `1` for S-Tagged. Default value: `1`.
sched_type	The schedule type. Valid values: `0` for QoS ETS, `1` for vPort bandwidth. Default value: `0`.
num_vlan	Maximum VLANs in Partition (integer). Valid Values: `0` for S-Tagged). Default value: `0`.

Parameter	Description
scid	The enabled vPort's S-Channel ID. An integer from 2-9.
svid	The vPort's S-Channel default VLAN ID. An integer from 2-3999 or 4002-4009.
iscsi	Whether the switch is iSCSI capable. Valid values: `0` for no, `1` for yes. Default value: `0`.
host_pri	Whether the vPort is host priority control or not. Valid values: `0` for no, `1` for yes. Default value: `0`.
fcoe	Whether the vPort is FCoE capable. Valid values: `0` for no, `1` for yes. Default value: `0`.
minbw	The minimum guaranteed bandwidth, in percent. An integer from 10-100, or `0` for QoS ETS mode. Default value: `25`.
maxbw	The maximum allowed bandwidth, in percent. An integer from 10-100, or `0` for QoS ETS mode. Default value: `100`.

class UfpQoSInfo()

This class gets UFP QoS information.

ufp_get_qos_info()

Gets UFP QoS information.

Syntax

`ufp_get_qos_info``(`*if_name*`)`

where:

Parameter	Description
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.

Returns

A dictionary containing UFP QoS information:

Parameter	Description
if_name	The Ethernet interface name (string). Valid value: `“ethernet”`.
vport	The virtual port index. An integer from 1-8.
mode	The current QoS mode on interface (string). Valid values: `"ets"`, `"bw"`. Default value: `"ets"`.
minbw	The minimum guaranteed bandwidth, in percent. An integer from 10-100, or `0` for QoS ETS mode. Default value: `25`.
maxbw	The maximum allowed bandwidth, in percent. An integer from 10-100, or `0` for QoS ETS mode. Default value: `100`.
priority	802.1p class priority of vPort, in Qos ETS mode. An integer from 0-7. Default value: `0`.
host_pri	Whether OS provides priority or not (integer). Valid values: `0` for no, `1` for yes. Default value: `0`.

class UfpVlanInfo()

This class gets UFP VLAN information.

ufp_get_vlan_info()

Gets UFP VLAN information.

Syntax

`ufp_get_vlan_info``(`*vid*`)`

where:

Parameter	**Description**
vid	The VLAN ID. An integer from 2-3999.

Returns

A dictionary containing UFP VLAN information:

Parameter	**Description**
vid	The VLAN ID. An integer from 2-3999.
vport_list	A list of dictionaries of UFP vPorts.
vport	The UFP Virtual interface name (string). Valid values: "`ufp-virtual 1/x/y`".
non_ufp_list	A list of dictionaries of non-UFP interfaces.
if_name	The interface name of non-UFP ports (string). Valid values: "`Ethernet 1/X `".
ufp_list	A list of dictionaries of UFP interfaces.
if_name	The interface name of UFP ports (string). Valid values: "`Ethernet 1/X `".

class UfpVport()

This class gets and sets UFP virtual interface configurations.

ufp_get_vport()

Gets UFP virtual interface configurations.

Syntax

`ufp_get_vport``(`*vif_name*`)`

where:

Parameter	**Description**
vif_name	The UFP virtual interface name (string). Valid values: `"ufp-virtual 1/x/y"`.

Returns

A dictionary containing UFP virtual interface information:

Parameter	**Description**
vif_name	The UFP virtual interface name (string). Valid values: `"ufp-virtual 1/x/y"`.
ena	Whether UFP operation is enabled or not (string). Valid values: `"yes"`, `"no"`. Default value: `"no"`.
mode	UFP virtual interface network mode (string). Valid values: `“access”`, `"fcoe"`, `"trunk"`, `"tunnel"`. Default value: `"tunnel"`.
tag_defvlan	Whether tag on default VLAN is enabled or not. Valid values: `"yes"`, `"no"`. Default value: `"no"`.
defvlan	The default VLAN of the UFP virtual interface. An integer from 2-3999.
minbw	The minimum guaranteed bandwidth percentage for a UFP virtual interface. An integer from 0-100. Default value: `25`.
maxbw	The maximum allowed bandwidth percentage for a UFP virtual interface. An integer from 0-100. Default value: `100`.
priority	Set 802.1p traffic class priority of QoS ETS mode of a UFP virtual interface. An integer from 0-7. Default value: `0`.
hostctrl_ena	Whether host COS control of a UFP virtual interface is enabled or not (string). Valid values: `"yes"`, `"no"`. Default value: `"no"`.

Parameter	Description
allowed_vlans	A list of trunk mode accepted VLANs of a UFP virtual interface.
vid	The trunk mode accepted VLAN ID of a UFP Virtual Interface. An integer from 2-3999.

ufp_get_all_vport()

Gets all UFP virtual interface configurations.

Syntax

```
ufp_get_all_vport()
```

Returns

A dictionary containing UFP virtual interface information:

Parameter	Description
vif_name	The UFP virtual interface name (string). Valid values: `"ufp-virtual 1/x/y"`.
ena	Whether UFP operation is enabled or not (string). Valid values: `"yes"`, `"no"`. Default value: `"no"`.
mode	UFP virtual interface network mode (string). Valid values: `“access”`, `"fcoe"`, `"trunk"`, `"tunnel"`. Default value: `"tunnel"`.
tag_defvlan	Whether tag on default VLAN is enabled or not. Valid values: `"yes"`, `"no"`. Default value: `"no"`.
defvlan	The default VLAN of the UFP virtual interface. An integer from 2-3999.
minbw	The minimum guaranteed bandwidth percentage for a UFP virtual interface. An integer from 0-100. Default value: `25`.
maxbw	The maximum allowed bandwidth percentage for a UFP virtual interface. An integer from 0-100. Default value: `100`.
priority	802.1p traffic class priority of QoS ETS mode of a UFP virtual interface. An integer from 0-7. Default value: `0`.
hostctrl_ena	Whether host COS control of a UFP virtual interface is enabled or not (string). Valid values: `"yes"`, `"no"`. Default value: `"no"`.
allowed_vlans	A list of trunk mode accepted VLANs of a UFP virtual interface.
vid	The trunk mode accepted VLAN ID of a UFP Virtual Interface. An integer from 2-3999.

ufp_set_vport()

Sets UFP virtual interface configurations.

Syntax

`ufp_set_vport``(`*dict_ufp_vport_new, create*`)`

where:

Parameter	**Description**
vif_name	The UFP virtual interface name (string). Valid values: `"ufp-virtual 1/x/y"`.
ena	Whether UFP operation is enabled or not (string). Valid values: `"yes"`, `"no"`. Default value: `"no"`.
mode	UFP virtual interface network mode (string). Valid values: `“access”`, `"fcoe"`, `"trunk"`, `"tunnel"`. Default value: `"tunnel"`.
tag_defvlan	Whether tag on default VLAN is enabled or not. Valid values: `"yes"`, `"no"`. Default value: `"no"`.
defvlan	The default VLAN of the UFP virtual interface. An integer from 2-3999. **Note:** VLAN must be configured.
minbw	The minimum guaranteed bandwidth percentage for a UFP virtual interface. An integer from 0-100. Default value: `25`.
maxbw	The maximum allowed bandwidth percentage for a UFP virtual interface. An integer from 0-100. Default value: `100`.
priority	802.1p traffic class priority of QoS ETS mode of a UFP virtual interface. An integer from 0-7. Default value: `0`.
hostctrl_ena	Whether host COS control of a UFP virtual interface is enabled or not (string). Valid values: `"yes"`, `"no"`. Default value: `"no"`.
allowed_vlans	A list of trunk mode accepted VLANs of a UFP virtual interface. **Note:** VLANs must be configured.
vid	The trunk mode accepted VLAN ID of a UFP Virtual Interface. An integer from 2-3999.
create	Create a new UFP Virtual Interface or update an existing one (boolean). Valid values: `True`, `False`.

Returns

Boolean (`True` on success, otherwise `False`).

ufp_delete_all_vport()

Deletes all UFP virtual interface configurations.

Syntax

`ufp_delete_all_vport``()`

Returns

Boolean (`True` on success, otherwise `False`).

ufp_delete_vport()

Deletes the specified UFP virtual interface configurations.

Syntax

`ufp_delete_all_vport``(`*vif_name*`)`

where:

Parameter	**Description**
vif_name	The UFP virtual interface name (string). Valid values: `"ufp-virtual 1/x/y"`.

Returns

Boolean (`True` on success, otherwise `False`).

class UfpVportInfo()

This class gets UFP vPort information.

ufp_get_vport_info()

Gets UFP vPort information.

Syntax

`ufp_get_vport_info``(`*if_name, vpidx*`)`

where:

Parameter	Description
if_name	The interface name (string). Valid values: `“Ethernet 1/X”`.

Returns

A dictionary containing UFP vPort information:

Parameter	Description
if_name	The interface name (string). Valid values: `“Ethernet 1/X”`. Default value: `none`.
vport	The UFP virtual port index. An integer from 1-8. Default value: `none`.
state	The current state of vPort (string). Valid values: `"linkup”`, `"linkdown"`, `"mismatch"`, `"failover"`.
mode	The vPort network mode (string). Valid values: `"access"`, `"fcoe"`, `"trunk"`, `"tunnel"`. Default value: `"tunnel"`.
svid	The UFP vPort’s S-Channel default VLAN ID. An integer from 2-3999 or 4002-4009.
defvlan	The default VLAN of UFP Virtual Interface. An integer from 2-3999.
deftag	Whether tagging on default VLAN is enabled or not (string). Valid values: `“yes”`, `“no”`. Default value: `“no”`.
l2f	Whether UFP Virtual Interface is L2 Failover controlled (string). Valid values: `“yes”`, `“no”`. Default value: `“no”`.
vlans	A list of accepted VLAN IDs of a UFP Virtual Interface.
vid	The accepted VLAN ID of a UFP Virtual Interface. An integer from 2-3999.

vLAG Module

This module manages Virtual Link Aggregation Group (vLAG) information.

To use this module, in the Python file or in the Python interpreter, enter:

`import vlagApi`

class VlagGlobal()

This class provides functions to get and set vLAG global parameters.

set_vlag_state()

Sets the vLAG global state.

Syntax

`set_vlag_state``(`*state*`)`

where:

Parameter	**Description**
state	The source subnet prefix (boolean). Valid values: `True`, `False`.

Returns

Boolean (`True` on success, otherwise `False`).

set_vlag_startup_delay()

Sets the vLAG startup delay, in seconds.

Syntax

`set_vlag_state``(`*startup_delay*`)`

where:

Parameter	**Description**
startup_delay	The vLAG startup delay, in seconds. An integer from 0-3600. Default value: `120`.

Returns

Boolean (`True` on success, otherwise `False`).

set_vlag_auto_recover()

Sets the time interval, in seconds, after which the switch can assume the Primary role from an unresponsive ISL peer and bring up the vLAG ports.

Syntax

`set_vlag_auto_recover``(`*auto_recover*`)`

where:

Parameter	**Description**
auto_recover	The vLAG auto recovery time, in seconds. An integer from 240-3600. Default value: `300`.

Returns

Boolean (`True` on success, otherwise `False`).

set_vlag_tier_id()

Sets the vLAG tier ID.

Syntax

`set_vlag_tier_id``(`*tier_id*`)`

where:

Parameter	**Description**
tier_id	The vLAG tier ID. An integer from 0-512. Default value: `0`.

Returns

Boolean (`True` on success, otherwise `False`).

set_vlag_priority()

Sets the vLAG priority.

Syntax

`set_vlag_priority``(`*priority*`)`

where:

Parameter	Description
priority	The vLAG priority. An integer from 0-65535. Default value: `0`.

Returns

Boolean (`True` on success, otherwise `False`).

get_vlag_config()

Gets vLAG global configuration values.

Syntax

`get_vlag_config``()`

Returns

A dictionary containing vLAG global configuration details:

Parameter	Description
status	Whether the vLAG is enabled or disabled (string). Valid values: “`enable`”, “`disable`”. Default value: “`disable`”.
tier_id	vLAG tier ID value. An integer from 1-512. Default value: `0`.
priority	vLAG priority value. An integer from 0-65535. Default value: `0`.
auto_recover	Time interval, in seconds. An integer from 240-3600. Default value: `300`.
startup_delay	Delay time, in seconds. An integer from 0-3600. Default value: `120`.

get_vlag_info()

Gets vLAG configuration and information.

Syntax

`get_vlag_info``()`

Returns

A dictionary containing vLAG information:

Parameter	**Description**
status	Whether the vLAG is enabled or disabled (string). Valid values: `“enable”`, `“disable”`. Default value: `“disable”`.
system_mac	Unique vLAG system MAC used for LACP negotiation on the vLAG ports so the access switch forms a single LAG (string). The vLAG `tier_id` is used to form this vLAG system MAC.
fdb_refresh	Whether FDB refresh is configured (string). Valid values: `”yes”`, `”no”`.
fdb_synch	Whether FDB is synchronized (string). Valid values: `”yes”`, `”no”`.
auto_recovery	A dictionary consisting of the following values: ● `interval`: Time interval, in seconds. An integer from 240-3600. Default value: `300`. ● `state`: Auto-recovery state (string). Valid values: `”unstarted”`, `”running”`, `”finished”`.
startup_delay	A dictionary consisting of the following values: ● `interval`: Delay time, in seconds. An integer from 0-3600. Default value: 120. ● `state`: Startup delay state; one of `unstarted`, `running`, `finished`.

Parameter	**Description**
local	Dictionary containing the following values: ● `tier_id`: vLAG tier ID of the local switch. An integer from 1-512. Default value: `0`. ● `sys_type`: Lenovo hardware model number (string). ● `os_version` CNOS version (string). ● `admin_role`: One of `”Primary”`, `”Secondary”`, `”Unselected”`. ● `oper_role`: One of `”Primary”`, `”Secondary”`, `”Unselected”`. ● `priority`: The local vLAG priority. An integer from 0-65535. Default value: `0`. ● `system_mac`: Local switch MAC (string). ● `match`: Whether there is an ISL local match or mis-match (string). Valid values: `”Match”`, `”Mis-Match”`.
peer	Dictionary containing the following values: ● `tier_id`: vLAG tier ID of the local switch. An integer from 1-512. Default value: `0`. ● `sys_type`: Lenovo hardware model number (string). ● `os_version` CNOS version (string). ● `admin_role`: One of `”Primary”`, `”Secondary”`, `”Unselected”`. ● `oper_role`: One of `”Primary”`, `”Secondary”`, `”Unselected”`. ● `priority`: The local vLAG priority. An integer from 0-65535. Default value: `0`. ● `system_mac`: Local switch MAC (string). ● `match`: Whether there is an ISL local match or mis-match (string). Valid values: `”Match”`, `”Mis-Match”`.

class VlagIsl()

This class provides functions to set/reset vLAG Inter-Switch Link (ISL) parameters.

set_vlag_isl()

Sets the vLAG ISL by configuring a port-aggregator.

Syntax

`set_vlag_isl``(`*agg_id*`)`

where:

Parameter	Description
agg_id	Aggregator identifier. An integer from 1-4096.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

reset_vlag_isl()

Resets the vLAG ISL.

Syntax

`reset_vlag_isl`

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

get_vlag_isl()

Gets vLAG ISL information.

Syntax

`get_vlag_isl``()`

Returns

A dictionary containing vLAG ISL information:

Parameter	**Description**
port_aggregator	Aggregator identifier. An integer from 1-4096.
if_index	ISL interface index (integer).
state	ISL state (string). Valid values: `"Down"`, `"Inactive"`, `"Active"`.
prev_state	Previous ISL state (string). Valid values: `"Down"`, `"Inactive"`, `"Active"`.

class VlagHlthChk()

This class provides functions to get and set vLAG health check information.

get_vlag_hc()

Gets or updates vLAG health check information.

Syntax

`get_vlag_hc``()`

Returns

A dictionary containing vLAG health check information:

Parameter	**Description**
status	The vLAG health check status (string). Valid values: `"up"`, `"down"`.
peer_ip	The vLAG health check peer IP address (string). Valid values: `"IPv4"`, `"IPv6"`.
local_ip	The vLAG health check local interface IP address (string). Valid values: `"IPv4"`, `"IPv6"`.
vrf	The VRF context string. A string up to 64 characters long. Valid values: `"default"`, `"management"`, `"VRFNAME"`. Default value: `"default"`.
retry_interval	The retry interval, in seconds. An integer from 1-300.

Parameter	Description
keepalive_attempts	Number of keepalive attempts made before declaring the peer is down. An integer from 1-24.
keepalive_interval	Keepalive interval, in seconds. An integer from 2-300.

set_vlag_hc_ka_attempts()

Sets the number of vLAG health check keepalive attempts.

Syntax

`set_vlag_hc_ka_attempts``(`*ka_attempts*`)`

where:

Parameter	Description
ka_attempts	Keepalive attemps. An integer from 1-24.

Returns

Boolean (`True` on success, otherwise `False`).

set_vlag_hc_ka_interval()

Sets the number of vLAG health check keepalive attempts.

Syntax

`set_vlag_hc_ka_interval``(`*ka_interval*`)`

where:

Parameter	Description
ka_interval	The keepalive interval, in seconds. An integer from 2-300.

Returns

Boolean (`True` on success, otherwise `False`).

set_vlag_hc_peer_ip()

Sets the health check peer IP address.

Syntax

`set_vlag_hc_peer_ip``(`*peer_ip*`,`*vrf_name*`)`

where:

Parameter	**Description**
peer_ip	The vLAG peer IP address (string). Valid values: `“IPv4”`, `“IPv6”`.
vrf_name	(Optional) The VRF name. A string up to 64 characters long. Valid values: `“default”`, `“management”`, `“VRFNAME”`. Default value: `“default”`.

Returns

Boolean (`True` on success, otherwise `False`).

set_vlag_hc_retry_interval()

Sets the vLAG health check retry interval time.

Syntax

`set_vlag_hc_peer_ip``(`*retry_interval*`)`

where:

Parameter	**Description**
retry_interval	Retry interval, in seconds. An integer from 1-300.

Returns

Boolean (`True` on success, otherwise `False`).

class VlagInstance()

This class provides functions to add, remove, and get information about vLAG instances.

add_vlag_instance()

Adds a vLAG instance.

Syntax

`add_vlag_instance(``*inst_id*,*agg_id*)`

where:

Parameter	**Description**
inst_id	vLAG instance ID number. An integer from 1-64.
agg_id	The aggregator identifier. An integer from 1-4096.

Returns

Boolean (`True` on success, otherwise `False`).

del_vlag_instance()

Deletes a vLAG instance.

Syntax

`del_vlag_instance(``*inst_id*)`

where:

Parameter	**Description**
inst_id	The vLAG instance ID number. An integer from 1-64.

Returns

Boolean (`True` on success, otherwise `False`).

get_vlag_all_inst_cfg()

Gets all vLAG instance configuration values.

Syntax

`get_vlag_all_inst_cfg``()`

Returns

A list of dictionaries containing all vLAG instance configuration values:

Parameter	**Description**
inst_id	The vLAG instance ID number. An integer from 1-64.
port_aggregator	The vLAG instance port-aggregator. An integer from 1-4096.
status	The vLAG instance status (string). Valid values: `“enabled”`, `”disabled”`.

get_vlag_all_inst_info()

Gets all vLAG instance information.

Syntax

`get_vlag_all_inst_info`

Returns

A list of dictionaries containing all vLAG instance configuration values:

Parameter	**Description**
inst_id	The vLAG instance ID number. An integer from 1-64.
port_aggregator	The vLAG instance port-aggregator. An integer from 1-4096.
status	The vLAG instance status (string). Valid values: one of `”Down”`, `”Local Up”`, `”Remote Up”`, `”Formed”`.
prev_state	Previous vLAG instance state (string). Valid values: one of `”Down”`, `”Local Up”`, `”Remote Up”`, `”Formed”`.

get_vlag_instance_cfg()

Gets vLAG configuration information for the specified instance.

Syntax

`get_vlag_instance_cfg``(`*inst_id*`)`

where:

Parameter	**Description**
inst_id	The vLAG instance ID number. An integer from 1-64.

Returns

A dictionary containing vLAG configuration values for the specified instance:

Parameter	**Description**
inst_id	The vLAG instance ID number. An integer from 1-64.
port_aggregator	The vLAG instance port-aggregator. An integer from 1-4096.
status	The vLAG instance status (string). Valid values: `“enabled”,”disabled”`.

get_vlag_instance_info()

Gets information for the specified vLAG instance.

Syntax

`get_vlag_instance_info``(`*inst_id*`)`

where:

Parameter	**Description**
inst_id	The vLAG instance ID number. An integer from 1-64.

Returns

A dictionary containing information for the specified vLAG instance:

Parameter	**Description**
inst_id	The vLAG instance ID number. An integer from 1-64.
port_aggregator	The vLAG instance port-aggregator. An integer from 1-4096.

Parameter	Description
status	The vLAG instance status (string). Valid values: one of `"Down"`, `"Local Up"`, `"Remote Up"`, `"Formed"`.
prev_state	Previous vLAG instance state (string). Valid values: one of `"Down"`, `"Local Up"`, `"Remote Up"`, `"Formed"`.

remove_pag_vlag_instance()

Removes the port-aggregator associated with the specified vLAG instance.

Syntax

`remove_pag_vlag_instance``(`*inst_id*`)`

where:

Parameter	Description
inst_id	The vLAG instance ID number. An integer from 1-64.

Returns

Boolean (`True` on success, otherwise `False`).

set_vlag_instance_state()

Sets the vLAG instance state.

Syntax

`set_vlag_instance_state``(`*inst_id*`,` *state*`)`

where:

Parameter	Description
inst_id	vLAG instance ID number. An integer from 1-64.
state	vLAG instance state (boolean). Valid values: `True`, `False`.

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `True` for success or `False` for failure

class VlagOrphanPort()

This class provides functions to manage vLAG orphan port suspend state for interface.

get_vlag_orphan_port_state()

Gets the vLAG orphan port suspend status for a specified interface.

Syntax

`get_vlag_orphan_port_state`(*if_name*)

where:

Parameter	Description
if_name	The Ethernet or LAG interface name (string).

Returns

A dictionary containing vLAG information:

Parameter	Description
status	The vLAG orphan port configuration status (string). Valid values: `“enabled”`, `”disabled”`.

set_vlag_orphan_port_state()

Sets the vLAG orphan port suspend status for a specified interface.

Syntax

`set_vlag_orphan_port_state`(*if_name, suspend_state*)

where:

Parameter	Description
if_name	The Ethernet or LAG interface name (string).
suspend_state	The vLAG orphan port suspend status (string). Valid values: `“enabled”`, `”disabled”`.

Returns

Boolean (`True` on success, otherwise `False`).

class VlagSVI()

This class provides functions to manage SVI exclude interface VLAN list.

get_vlag_svi_exclude()

Gets the vLAG SVI exclude interface VLAN list.

Syntax

`get_vlag_svi_exclude``()`

Returns

A dictionary containing VLAN list:

Parameter	**Description**
vlan_id	The VLAN ID. An integer from 1-4094.

set_vlag_svi_exclude()

Sets the vLAG SVI exclude interface VLAN list.

Syntax

`set_vlag_svi_exclude``(`*vlan_json_list*`)`

vlan_json_list contains the following information:

Parameter	**Description**
vlans	A list of VLAN ID.
vlan_id	The VLAN ID. An integer from 1-4094.

Returns

Boolean (`True` on success, otherwise `False`).

VLAN Module

This module configures VLAN properties.

To use this module, in the Python file or in the Python interpreter, enter:

`import vlanApi`

class VlanSystem()

This class provides functions to get and set VLAN configurations.

python_get_vlan()

Gets properties of a VLAN if the VLAN identifier (*vid*) is a valid VLAN ID or of all VLANs if *vid* is `None`.

Syntax

`python_get_vlan``(`*vid*`)`

where:

Parameter	Description
vid	(Optional) The VLAN ID. An integer from 1-4093.

Returns

A dictionary with VLAN properties if (*vid*) is a valid VLAN ID or of all VLANs if *vid* is `None`:

Parameter	Description
vlan_name	The name of the VLAN (string).
vlan_id	The VLAN identifier. An integer from 1-4093.
admin_state	The admin status of the VLAN (string). Valid values: `”up”`, `”down”`.
ipmc_flood	The VLAN IPMC flood (string). Valid values: `"disable"`, `"ipv4"`, `"ipv6"`, `"ipv4ipv6"`.
mst_inst_id	MST instance identifier. An integer from 0-64. 0 refers to CIST. Default value: `0`.
interfaces	List of interface members of the VLAN containing the following properties: ● `pvid` ● `bridgeport_mode` ● `if_name`
pvid	Native VLAN ID of the port. An integer from 1-4093. Default value: 1.

Parameter	**Description**
bridgeport_mode	The ridge port mode (string). Valid values: `“access”`, `”trunk”`, `“hybrid”`.
if_name	Ethernet interface name (string).

python_create_vlan()

Creates a VLAN.

Syntax

`python_create_vlan``(`*vlan_info*`)`

where:

Parameter	**Description**
vlan_info	A dictionary with the following VLAN information: ● `vlan_name` ● `vlan_id` ● `admin_state` If `vlan_name` is null, a VLAN with a default name will be created.
vlan_name	The VLAN name (string). To create a VLAN with the default name, this field must be null.
vlan_id	The VLAN identifier. An integer from 2-3999.
admin_state	The admin status (string). Valid values: `“up”`, `“down”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_delete_vlan()

Deletes a VLAN

Syntax

`python_delete_vlan``(`*vid*`)`

where:

Parameter	Description
vid	VLAN ID (Int) If `vid=all`, all user-created VLANs are deleted.

Returns

Boolean (`True` on success, otherwise `False`).

python_update_vlan_name()

Updates VLAN name.

Syntax

`python_update_vlan_name``(`*vid, vlan_name*`)`

where:

Parameter	Description
vid	VLAN ID (integer).
vlan_name	The VLAN name (string).

Returns

Boolean (`True` on success, otherwise `False`).

python_update_vlan_admin_state()

Updates VLAN admin state.

Syntax

`python_update_vlan_admin_state``(`*vid, admin_state*`)`

where:

Parameter	**Description**
vid	VLAN ID (integer).
admin_state	The administrative state (string). Valid values: `“up”`, `“down”`.

Returns

Boolean (`True` on success, otherwise `False`).

python_update_vlan_ipmc_flood()

Updates the VLAN IPMC flood.

Syntax

`python_update_vlan_ipmc_flood``(`*vid, flood*`)`

where:

Parameter	**Description**
vid	VLAN ID. An integer from 2-4093.
flood	The VLAN IPMC FLOOD (string). Valid values: `"disable"`, `"ipv4"`, `"ipv6"`, `"ipv4ipv6"`.

Returns

Boolean (`True` on success, otherwise `False`).

class VlanEthIf()

This class provides functions to affect VLAN properties of Ethernet interfaces.

python_get_vlan_properties()

Gets the VLAN properties of an Ethernet Interface or of all Ethernet Interfaces if the interface name (*if_name* argument) is `None`.

Syntax

`python_get_vlan_properties``(`*if_name*`)`

where:

Parameter	Description
if_name	The interface name (string). A valid interface name or `None`.

Returns

A dictionary with VLAN properties of the specified interface or of all interfaces:

Parameter	Description
if_name	Ethernet interface name (string).
bridgeport_mode	The switch bridgeport mode (string). Valid values: `“access”`, `”trunk”`, `“hybrid”`.
pvid	The Access VLAN for access ports or the Native VLAN for trunk/hybrid ports. An integer from 1-4093.
vlans	A list of all VLANs attached to this interface. Valid values: `"all"`,`"none"`, 1-4093.
vlan_id	The VLAN ID. An integer from 1-4093.
egress_type	Whether traffic is egress tagged when the interface is in hybrid mode (string). Valid values: `”tagged”`, `”untagged”`.
egress_type_vlans	A list of VLANs on which the switch tags egress traffic. An integer from 1-4093.
tag_native	The tag native VLAN (string). Valid values: `“enable”`, `”disable”`, `“egress_only”`, `“none”`.
ingress_tagging	The port mode dot1q tunnel (string). Valid values: `“enable”`, `”disable”`.

python_update_vlan_properties()

Updates VLAN interface properties.

Syntax

`python_update_vlan_properties``(`*dict_if_new_info*`)`

dict_if_new_info is a dictionary containing the following parameters:

Parameter	**Description**
if_name	The switch interface name (string).
bridgeport_mode	The switch bridgeport mode (string). Valid values: `“access”`, `”trunk”`, `“hybrid”`.
pvid	The Access VLAN for access ports or the Native VLAN for trunk/hybrid ports. An integer from 1-4093.
vlans	A list of all VLANs attached to this interface. Valid values: `"all"`,`"none"`, 1-4093.
operation	(Optional) The VLAN list operation type (string). Valid values: `“add”`, `”remove”`, `“except”`.
vlan_id	(Optional) The VLAN ID. An integer from 1-4093.
egress_type	(Optional) Whether traffic is egress tagged when the interface is in hybrid mode (string). Valid values: `”tagged”`, `”untagged”`.
egress_type_vlans	(Optional) A list of VLANs on which the switch tags egress traffic. An integer from 1-4093.
tag_native	(Optional) The tag native VLAN (string). Valid values: `“enable”`, `”disable”`, `“egress_only”`, `“none”`.
ingress_tagging	(Optional) The port mode dot1q tunnel (string). Valid values: `“enable”`, `”disable”`.

Returns

Boolean (`True` on success, otherwise `False`).

class GlobalVlanTagNative()

This class provides functions to get and set global VLAN tag native properties.

python_get_vlan_global_tag_native()

Gets the global VLAN tag native properties.

Syntax

`python_get_vlan_global_tag_native``()`

Returns

A dictionary with global VLAN properties:

Parameter	**Description**
global_tag_native	The global tag native VLAN (string). Valid values: `"enable"`, `"disable"`, `"eggress_only"`.
if_name	The switch interface name (string). **Note:** The interface information will be displayed only if the interface is in trunk mode.
tag_native	The tag native VLAN of the a specified interface or the global tag native value (string). Valid values: `"enable"`, `"disable"`, `"eggress_only"`.

python_set_vlan_global_tag_native()

Updates the global VLAN tag native properties.

Syntax

`python_set_vlan_global_tag_native``(`*tag_native*`)`

where:

Parameter	**Description**
tag_native	The global tag native VLAN (string). Valid values: `"enable"`, `"disable"`, `"eggress_only"`.

Returns

Boolean (`True` on success, otherwise `False`).

class VlanReserved()

This class provides functions to get and set system reserved VLAN properties.

set_reserved_vlan_range()

Sets system reserved VLAN properties.

Syntax

`set_reserved_vlan_range``(`*lower, higher*`)`

where:

Parameter	Description
lower	The lower ID of reserved VLAN range. An integer from 1-4094.
higher	The higher ID of reserved VLAN range. An integer from 1-4094.

Returns

Boolean (`True` on success, otherwise `False`).

reset_reserved_vlan_range()

Resets system reserved VLAN properties to the default range (4000-4093).

Syntax

`reset_reserved_vlan_range``()`

Returns

Boolean (`True` on success, otherwise `False`).

get_reserved_vlan_range()

Gets system reserved VLAN properties.

Syntax

```
get_reserved_vlan_range()
```

Returns

A dictionary with reserved VLAN properties:

Parameter	**Description**
cfg_lower_id	The current configured lower ID of the reserved VLAN range. An integer from 1-4094.
cfg_higher_id	The current configured higher ID of the reserved VLAN range. An integer from 1-4094.
cur_lower_id	The current operational lower ID of the reserved VLAN range. An integer from 1-4094.
cur_higher_id	The current operational higher ID of the reserved VLAN range. An integer from 1-4094.

VRF Module

This module manages Virtual Routing and Forwarding (VRF).

To use this module, in the Python file or in the Python interpreter, enter:

`import vrfApi`

class VRF()

This class provides functions to manage VRF.

create_vrf ()

Creates a VRF instance.

Syntax

`create_vrf``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	The VRF name. A string up to 63 characters long.

Returns

Boolean (`True` on success, otherwise `False`).

delete_vrf ()

Deletes a VRF instance.

Syntax

`delete_vrf``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	The VRF name. A string up to 63 characters long.

Returns

Boolean (`True` on success, otherwise `False`).

get_vrf_entry()

Gets all VRF details.

Syntax

`get_vrf_entry``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	(Optional) The VRF name. A string up to 63 characters long.

Returns

The following VRF details:

Parameter	**Description**
vrf_name	The VRF name. A string up to 63 characters long.
interfaces	A list of interfaces belonging to the specified VRF.

set_vrf_entry ()

Associates an interface with a VRF.

Syntax

`set_vrf_entry``(`*vrf_name, if_name*`)`

where:

Parameter	**Description**
vrf_name	The VRF name. A string up to 63 characters long.
if_name	The interface name (string).

Returns

Boolean (`True` on success, otherwise `False`).

delete_vrf_entry ()

Unbinds an interface from a VRF.

Syntax

`delete_vrf_entry``(`*vrf_name, if_name*`)`

where:

Parameter	**Description**
vrf_name	The VRF name. A string up to 63 characters long.
if_name	The interface name (string).

Returns

Boolean (`True` on success, otherwise `False`).

class VRFDescription()

This class provides functions to manage VRF descriptions.

get_vrf_description()

Gets VRF descriptions.

Syntax

`get_vrf_description``(`*vrf_name*`)`

where:

Parameter	**Description**
vrf_name	(Optional) The VRF name. A string up to 64 characters long.

Returns

Multiple possible return values:

- `False` if an error occurred
- a list of dictionaries of VRF descriptions:

Parameter	**Description**
vrf_name	The VRF entry name. A string up to 64 characters long.
description	The VRF description (string).

set_vrf_description()

Sets the description of a VRF.

Syntax

`set_vrf_description``(`*vrf_name, vrf_description*`)`

where:

Parameter	Description
vrf_name	The VRF name entry. A string up to 64 characters long.
vrf_description	The VRF description. A string up to 255 characters long.

Returns

Boolean (`True` on success, otherwise `False`).

unset_vrf_description()

Unsets the description of a VRF.

Syntax

`unset_vrf_description``(`*vrf_name*`)`

where:

Parameter	Description
vrf_name	The VRF name entry. A string up to 64 characters long.

Returns

Boolean (`True` on success, otherwise `False`).

class VRFRd()

This class provides functions to manage VRF route distinguishers.

get_vrf_rd()

Gets all VRF route distinguishers.

Syntax

`get_vrf_rd(`*vrf_name***`)`**

where:

Parameter	**Description**
vrf_name	(Optional) The VRF name. A string up to 64 characters long.

Returns

Multiple possible return values:

- `False` if an error occurred
- a list of dictionaries containing the following information:

Parameter	**Description**
vrf_name	The VRF entry name. A string up to 64 characters long.
route_distinguisher	The VRF route distinguisher (string). Valid values: `"ANS2:NN"`, `"ANS4:NN"`, `"IPV4:NN"`.

set_vrf_rd()

Sets the route distinguisher of a VRF.

Syntax

`set_vrf_rd(`*vrf_name, route_distinguisher***`)`**

where:

Parameter	**Description**
vrf_name	The VRF name. A string up to 64 characters long.
route_distinguisher	The VRF route distinguisher (string). Valid values: `"ANS2:NN"`, `"ANS4:NN"`, `"IPV4:NN"`.

Returns

Boolean (`True` on success, otherwise `False`).

unset_vrf_rd()

Unsets the route distinguisher of a VRF.

Syntax

`unset_vrf_rd``(`*vrf_name*`)`

where:

Parameter	Description
vrf_name	The VRF name. A string up to 64 characters long.

Returns

Boolean (`True` on success, otherwise `False`).

class VRFVni()

This class provides functions to manage VRF Virtual Network Identifiers (VNIs).

get_vrf_vni()

Gets all VRF VNIs.

Syntax

`get_vrf_vni``(`*vrf_name*`)`

where:

Parameter	Description
vrf_name	(Optional) The VRF name. A string up to 64 characters long.

Returns

Multiple possible return values:

- `False` if an error occurred
- a list of dictionaries containing the following information:

Parameter	Description
vrf_name	The VRF entry name. A string up to 64 characters long.
vni	The VRF Virtual Network Identifier. An integer from 1-16777214.

set_vrf_vni()

Sets the VNI of a VRF.

Syntax

`set_vrf_vni``(`*vrf_name, vni*`)`

where:

Parameter	**Description**
vrf_name	The VRF name. A string up to 64 characters long.
vni	The VRF Virtual Network Identifier. A string or an integer from 1-16777214.

Returns

Boolean (`True` on success, otherwise `False`).

unset_vrf_vni()

Unsets the VNI of a VRF.

Syntax

`unset_vrf_vni``(`*vrf_name, vni*`)`

where:

Parameter	**Description**
vrf_name	The VRF name. A string up to 64 characters long.
vni	The VRF Virtual Network Identifier. A string or an integer from 1-16777214.

Returns

Boolean (`True` on success, otherwise `False`).

class VRFRt()

This class provides functions to get, set and unset VRF route-target.

get_vrf_rt()

Gets all VRF route-targets.

Syntax

`get_vrf_rt(`*vrf_name***`)`**

where:

Parameter	Description
vrf_name	(Optional) The VRF name. A string up to 64 characters long.

Returns

Multiple possible return values:

- `False` if an error occurred
- a list of dictionaries containing the following information:

Parameter	Description
vrf_name	The VRF entry name. A string up to 64 characters long.
route_target	The list of Route-Targets assigned to the VRF (string). Valid values: `"import"`, `"export"`, `"both"`.

set_vrf_rt()

Sets the Route-Target of a VRF.

Syntax

`set_vrf_rt(`*vrf_name, route_target_option, route_target_value***`)`**

where:

Parameter	Description
vrf_name	The VRF name. A string up to 64 characters long.
route_target_option	The route-Target option of the VRF (string). Valid values: `"import"`, `"export"`, `"both"`.
route_target_value	The route-Target Value (string). Valid values: `"ASN2:NN"`, `"ASN4:NN"`, `"IPV4:NN"`.

Returns

Boolean (`True` on success, otherwise `False`).

unset_vrf_rt()

Unsets the Route-Target of a VRF.

Syntax

`unset_vrf_rt``(`*vrf_name, route_target_option, route_target_value*`)`

where:

Parameter	Description
vrf_name	The VRF name. A string up to 64 characters long.
route_target_ option	The route-Target option of the VRF (string). Valid values: `"import"`, `"export"`, `"both"`.
route_target_ value	The route-Target Value (string). Valid values: `"ASN2:NN"`, `"ASN4:NN"`, `"IPV4:NN"`.

Returns

Boolean (`True` on success, otherwise `False`).

VRRP Module

This module manages Virtual Router Redundancy Protocol (VRRP).

To use this module, in the Python file or in the Python interpreter, enter:

`import vrrpApi`

class VRRP()

This class provides functions to manage Virtual Router Redundancy Protocol.

get_vrrp()

Gets properties of all VRRP Virtual Routers (VRs) for all interfaces or for the specified interface.

Syntax

`get_vrrp``(`*vr_id, if_name*`)`

where:

Parameter	Description
vr_id	(Optional) The VRRP session Virtual Router (VR) ID; an integer from 1-255. Default value: `None`.
if_name	(Optional) Interface name used for communication (string). Default value: `None`. **Note:** The interface must exist.

Returns

A list of VRRP VR information:

Parameter	Description
if_name	The Ethernet interface name (string).
vr_id	The VRRP session Virtual Router (VR) ID; an integer from 1-255. Default value: `0`.
ip_addr	The IP address of the VR. A valid IPv4 address.
ad_intvl	Advertisement interval (The number of centi-seconds between advertisements for VRRPv3). A multiple of 5 from 5-4095. Default value: 100 centi-seconds.
preempt	Enable the preemption of a lower priority master (string). Valid values: `“yes”`, `”no”`. Default value: `“yes”`.
prio	The priority of the VR on the switch. An integer from 1-254. Default value: `100`.

Parameter	Description
admin_state	Enable the VR (string). Valid values: `“up”`, `”down”`. Default value: `“up”`.
oper_state	The operation state of the VR (string). Valid values: `“master”`, `“backup”`, `“init”`.
track_if	The interface to track by this VR (string). Default value: `none`. **Note:** If an interface is specified, it must exist.
accept_mode	Enables or disables the accept mode for this session (string). Valid values: `“yes”`, `”no”`. Default value: `“yes”`.
switch_back_delay	The switch back delay interval. An integer from 1-500000, or `0` to disable (default).
v2_compt	Enables backward compatibility for VRRPv2 for the VR (string). Valid values: `“yes”`, `”no”`. Default value: `“no”`.

get_vrrp_intf()

Gets properties of all VRRP VRs of specified interfaces.

Syntax

`get_vrrp_intf``(`*if_name*`)`

where:

Parameter	Description
if_name	The Ethernet interface name (string). **Note:** The interface must exist.

Returns

A list of VRRP properties:

Parameter	Description
if_name	The Ethernet interface name (string).
vr_id	The VRRP session Virtual Router (VR) ID. An integer from 1-255. Default value: `0`.
ip_addr	The IP address of the VR. A valid IPv4 address.
ad_intvl	Advertisement interval (The number of centi-seconds between advertisements for VRRPv3). A multiple of 5 from 5-4095. Default value: 100 centi-seconds.

Parameter	**Description**
preempt	Enable the preemption of a lower priority master (string). Valid values: `"yes"`, `"no"`. Default value: `"yes"`.
prio	The priority of the VR on the switch. An integer from 1-254. Default value: `100`.
admin_state	Enable the VR (string). Valid values: `"up"`, `"down"`. Default value: `"up"`.
oper_state	The operation state of the VR (string). Valid values: `"master"`, `"backup"`, `"init"`.
track_if	The interface to track by this VR (string). Default value: `none`. **Note:** If an interface is specified, it must exist.
accept_mode	Enables or disables the accept mode for this session (string). Valid values: `"yes"`, `"no"`. Default value: `"yes"`.
switch_back_delay	The switch back delay interval. An integer from 1-500000, or `0` to disable (default).
v2_compt	Enables backward compatibility for VRRPv2 for the VR (string). Valid values: `"yes"`, `"no"`. Default value: `"no"`.

get_vrrp_accept_mode()

Determines whether a virtual router in Master state will accept packets.

Syntax

`get_vrrp_accept_mode``(`*vr_id, af_type, if_name*`)`

where:

Parameter	**Description**
vr_id	The VRRP session Virtual Router (VR) ID. An integer from 1-255.
af_type	The Address Family type (integer). `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (string). **Note:** The interface must exist.

Returns

The `accept_mode` for the session (string). Valid values: `"yes"`, `"no"`. Default value: `"yes"`.

get_vrrp_advt_interval()

Gets the IGMP snooping status for a VLAN.

Syntax

`get_vrrp_advt_interval`(*vr_id, af_type, if_name*)

where:

Parameter	Description
vr_id	The VRRP session Virtual Router (VR) ID. An integer from 1-255.
af_type	The Address Family type (integer). `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (string). **Note:** The interface must exist.

`where:`

Parameter	Description
vid	The VLAN ID (integer).

Returns

The advertisement interval:

Parameter	Description
ad_intvl	Advertisement interval (The number of centi-seconds between advertisements for VRRPv3). A multiple of 5 from 5-4095. Default value: 100 centi-seconds.

get_vrrp_preempt_mode()

Gets whether a higher priority virtual router can preempt a lower priority master.

Syntax

`get_vrrp_preempt_mode``(`*vr_id, af_type, if_name*`)`

where:

Parameter	Description
vr_id	The VRRP session Virtual Router (VR) ID. An integer from 1-255.
af_type	The Address Family type (integer). `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	Ethernet interface name used for communication (string) **Note:** The interface must exist.

Returns

Whether the preemption of a lower priority master is enabled:

Parameter	Description
preempt	Enable the preemption of a lower priority master (string). Valid values: `“yes”`, `“no”`. Default value: `“yes”`.

get_vrrp_priority()

Gets the priority to be used for the virtual router master election process.

Syntax

`get_vrrp_priority`(*vr_id, af_type, if_name*)

where:

Parameter	**Description**
vr_id	The VRRP session Virtual Router (VR) ID. An integer from 1-255.
af_type	The Address Family type (integer). `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (string). **Note:** The interface must exist.

Returns

VRRP priority:

Parameter	**Description**
prio	The priority of the VR on the switch. An integer from 1-254. Default value: `100`.

set_vrrp_accept_mode()

Sets the accept mode for a VRRP session when VRPP V3 is enabled.

Syntax

`set_vrrp_accept_mode`(*vr_id, af_type, if_name, accept_mode*)

where:

Parameter	**Description**
vr_id	The VRRP session Virtual Router (VR) ID. An integer from 1-255.
af_type	The Address Family type (integer). `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (string). **Note:** The interface must exist.
accept_mode	Whether to enable Accept mode for this VRRP session (string). Valid values: `“yes”`, `“no”`. Default value: `“yes”`.

Returns

Boolean (`True` on success, otherwise `False`).

set_vrrp_advt_interval()

Sets the advertisement interval of a virtual router.

Syntax

`set_vrrp_advt_interval``(`*vr_id, af_type, if_name, interval*`)`

where:

Parameter	**Description**
vr_id	The VRRP session Virtual Router (VR) ID. An integer from 1-255.
af_type	The Address Family type (integer). `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (string). **Note:** The interface must exist.
interval	Advertisement interval in centi-seconds (integer). A multiple of 5 from 5-4095. Default value: `100` centi-seconds.

Returns

Boolean (`True` on success, otherwise `False`).

set_vrrp_preempt_mode()

Enables or disables the preempt mode for a session.

Syntax

`set_vrrp_preempt_mode``(`*vr_id, af_type, if_name, preempt*`)`

where:

Parameter	**Description**
vr_id	The VRRP session Virtual Router (VR) ID. An integer from 1-255.
af_type	The Address Family type (integer). `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (string). **Note:** The interface must exist.
preempt	Enable the preemption of a lower priority master (string). Valid values: `“yes”`, `“no”`. Default value: `“yes”`.

Returns

Boolean (`True` on success, otherwise `False`).

set_vrrp_priority()

Enables the configuration of the priority of the VRRP router for a session.

Syntax

`set_vrrp_priority``(`*vr_id, af_type, if_name, prio*`)`

where:

Parameter	**Description**
vr_id	The VRRP session Virtual Router (VR) ID. An integer from 1-255.
af_type	The Address Family type (integer. `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (string). **Note:** The interface must exist.
prio	The priority of the VR on the switch. An integer from 1-254. Default value: `100`.

Returns

Boolean (`True` on success, otherwise `False`).

set_vrrp_switch_back_delay()

Gets the IGMP snooping status for a VLAN.

Syntax

`set_vrrp_switch_back_delay``(`*vr_id, af_type, if_name, switch_back_delay*`)`

where:

Parameter	**Description**
vr_id	The VRRP session Virtual Router (VR) ID. An integer from 1-255.
af_type	The Address Family type (integer). `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (string). **Note:** The interface must exist.
switch_back_ delay	The switch back delay interval. An integer from 1-500000 or `0` to disable. Default value: `0`.

Returns

Boolean (`True` on success, otherwise `False`).

set_vrrp_oper_primary_ipaddr()

Sets the primary IP address of the VRRP virtual router.

Syntax

`set_vrrp_oper_primary_ipaddr``(`*vr_id, af_type, if_name, ipaddr*`)`

where:

Parameter	**Description**
vr_id	The VRRP session Virtual Router (VR) ID. An integer from 1-255.
af_type	The Address Family type (integer). `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (string). **Note:** The interface must exist.
ipaddr	The primary IP address (string).

Returns

Boolean (`True` on success, otherwise `False`).

set_vrrp_monitored_circuit()

Gets the IGMP snooping status for a VLAN.

Syntax

`set_vrrp_monitored_circuit``(`*vr_id, af_type, if_name, track_if*`)`

where:

Parameter	**Description**
vr_id	The VRRP session Virtual Router (VR) ID. An integer from 1-255.
af_type	The Address Family type (integer). `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (string). **Note:** The interface must exist.
track_if	The interface to track by this VR. **Note:** If an interface is specified, it must exist.

Returns

Boolean (`True` on success, otherwise `False`).

delete_vrrp_vr()

Deletes a VRRP VR.

Syntax

`delete_vrrp_vr``(`*vr_id, af_type, if_name*`)`

where:

Parameter	**Description**
vr_id	The VRRP session Virtual Router (VR) ID. An integer from 1-255.
af_type	The Address Family type (integer); `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (string). **Note:** The interface must exist.

Returns

Boolean (`True` on success, otherwise `False`).

set_vrrp_vr()

Creates a new VRRP session on the specified interface and allocate resources for the session.

Syntax

`set_vrrp_vr``(`*vr_id, if_name*`)`

where:

Parameter	**Description**
vr_id	The VRRP session Virtual Router (VR) ID. An integer from 1-255.
if_name	The Ethernet interface name (string). **Note:** The interface must exist.

Returns

Boolean (`True` on success, otherwise `False`).

VXLAN Module

This module configures Virtual Extensible LAN (VXLAN) properties.

To use this module, in the Python file or in the Python interpreter, enter:

`import vxlanApi`

python_vxlan_interface_ena()

Enables VXLAN mode on a specific interface.

Syntax

`python_vxlan_interface_ena``(`*interface_name*`)`

where:

Parameter	**Description**
interface_name	Whether the VXLAN is enabled on a specific interface (string). For example: “`Ethernet1/1`”.

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully enabled VXLAN mode. ● `FALSE`: Setting VXLAN mode failed.
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

python_vxlan_interface_dis()

Disables VXLAN mode on a specific interface.

Syntax

`python_vxlan_interface_dis``(`*interface_name*`)`

where:

Parameter	**Description**
interface_name	Whether the VXLAN is disabled on a specific interface (string). For example: `“Ethernet1/1”.`

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully disabled VXLAN mode. ● `FALSE`: Setting VXLAN mode failed.
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

python_vxlan_interface_get()

Gets the VXLAN status for a specific interface.

Syntax

`python_vxlan_interface_get``(`*interface_name*`)`

where:

Parameter	**Description**
interface_name	Whether the VXLAN is disabled on a specific interface (string). For example: `“Ethernet1/1”.`

Returns

A dictionary containing the following:

Parameter	**Description**
vxlan	The VXLAN interface status (string). Valid values: `“Enabled”`, `“Disabled”`.

python_vxlan_get()

Gets the VXLAN configuration.

Syntax

`python_vxlan_get``()`

Returns

A dictionary containing the following:

Parameter	**Description**
tunnel_ip_addr	The local Virtual Tunnel End Point (VTEP), the IP address. A string in the following format: “`10.1.2.1`”.
vlan_bindings	A list of Virtual Network Instance (VNI) VLAN maps.
remote_vtep	A list of VTEP with VNI.

python_vxlan_vni_cnt_get()

Gets all of the VXLAN VNI counters.

Syntax

`python_vxlan_vni_cnt_get``()`

Returns

A dictionary containing the following:

Parameter	**Description**
bin	Bytes received. A positive integer.
pktin	The number of packets received. A positive integer.
bout	Bytes sent. A positive integer.
pkout	The number of packets sent. A positive integer.
vni	The VNI identifier. An integer from 1-16000000.

python_vxlan_vp_get()

Gets all of the VXLAN Virtual Ports (VP).

Syntax

`python_vxlan_vp_get``()`

Returns

A dictionary containing the following:

Parameter	**Description**
count	The number of virtual ports (integer).
virtual port	A list of virtual ports.
remoteTEP	The type of VTEP (string). Valid values: `“LOCAL”`, `“REMOTE”`.
vlan	The VLAN number. The valid VLAN values range from 1-4000.
port	The switch interface name (string).

python_vxlan_vp_cnt_get()

Gets all of the VXLAN VP counters.

Syntax

`python_vxlan_vp_cnt_get``()`

Returns

A dictionary containing the following:

Parameter	**Description**
bin	Bytes received. A positive integer.
pktin	The number of packets received. A positive integer.
bout	Bytes sent. A positive integer.
pkout	The number of packets sent. A positive integer.
vni	The VNI identifier. An integer from 1-16000000.

python_vxlan_vp_cnt_del()

Clears all the VXLAN VP counters.

Syntax

```
python_vxlan_vp_cnt_del()
```

Returns

A dictionary containing the following elements:

Parameter	Description
return status	Return status of the API call: ● TRUE: Successfully cleared VXLAN VP counters. ● FALSE: Clearing VXLAN VP counters failed
error message	String value containing the reason for the API failure: ● none: When Return Status is TRUE. ● error string: When Return Status is FALSE.

python_vxlan_vn_cnt_del()

Clears all the VXLAN VNI counters.

Syntax

```
python_vxlan_vn_cnt_del()
```

Returns

A dictionary containing the following elements:

Parameter	Description
return status	Return status of the API call: ● TRUE: Successfully cleared VXLAN VNI counters. ● FALSE: Clearing VXLAN VNI counters failed.
error message	String value containing the reason for the API failure: ● none: When Return Status is TRUE. ● error string: When Return Status is FALSE.

python_vxlan_tunnel_get()

Gets all of the VXLAN VTEP tunnels.

Syntax

`python_vxlan_tunnel_get``()`

Returns

A dictionary containing the following:

Parameter	**Description**
count	The number of elements in the list (integer).
tunnel	The VTEP description. A list of dictionaries.
status	The BFD status of VTEP (string). Valid values: `“Up”`, `“Down”`.
remote	The remote IP address. A string in the following format: `“1.1.2.3”`.
local	The local IP address. A string in the following format: `“1.2.3.4”`.
type	The VTEP type (string). Valid values: `“VTEP”`, `“local”`.

python_vxlan_mac_address_get()

Gets all of the VXLAN MAC addresses.

Syntax

`python_vxlan_mac_address_get``()`

Returns

A dictionary containing the following:

Parameter	**Description**
local-count	The number of local MAC addresses (integer).
remote-count	The number of remote MAC addresses (integer).
local-umac *remote-umac*	The local/remote MAC. A list of dictionaries.
tunnel	The VTEP IP address. A string in the following format: “`1.1.2.3`”.
mac	The MAC address. A string in the following format: “`00:50:56:84:E3:2E`”.
local	The local IP address. A string in the following format: “`1.2.3.4`”.
type	The VTEP type (string). Valid values: “`SN Active`”, “`SN Backup`”, “`VTEP`”, “`local`”.

python_vxlan_vni_get()

Gets all of the VXLAN VNI.

Syntax

`python_vxlan_vni_get``()`

Returns

A dictionary containing the following:

Parameter	**Description**
count	The number of VNI elements in the list (integer).
vni	The list of VNI (integer).

python_vxlan_interface_vni_map_set()

Gets all of the VXLAN VNI mappings.

Syntax

`python_vxlan_interface_vni_map_set``(`*global, action, vlan, vni*`)`

where:

Parameter	**Description**
global	Wether the VNI-VLAN mappings are global (string). Default value: `“GLOBAL”`.
action	Indicates if a VTEP is added or removed (integer). Valid values: `0` to add a VTEP, or `2` to delete a VTEP.
vlan	The list of VLAN. A valid VLAN list.
vni	The list of VNI. A valid VNI list.

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully get VXLAN VNI mappings. ● `FALSE`: Getting VXLAN VNI mappings failed.
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

class VxlanCfg()

This class provides functions to set VXLAN configurations.

set_tunnel_ip()

Sets a Network Virtualization VXLAN tunnel IP.

Syntax

`set_tunnel_ip``(`*tunnel_ip*`)`

where:

Parameter	**Description**
tunnel_ip	The new tunnel IP (string). A valid IP address in the following format: `“10.0.1.1”.`

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● TRUE: Successfully set a Network Virtualization VXLAN tunnel IP. ● FALSE: Setting a Network Virtualization VXLAN tunnel IP failed.
error message	String value containing the reason for the API failure: ● none: When Return Status is TRUE. ● error string: When Return Status is FALSE.

del_tunnel_ip()

Deletes the Network Virtualization VXLAN local tunnel IP.

Syntax

`del_tunnel_ip` ()

Returns

A dictionary containing the following elements:

Parameter	**Description**
return status	Return status of the API call: ● `TRUE`: Successfully deleted a Network Virtualization VXLAN local tunnel IP. ● `FALSE`: Deleting a Network Virtualization VXLAN local tunnel IP failed.
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

set_remote_vtep()

Sets the Remote VTEP.

Syntax

`set_remote_vtep``(`*action, vni, ip_list*`)`

where:

Parameter	Description
action	Indicates if a VTEP is added or removed (integer). Valid values: `0` to add a VTEP, or `2` to delete a VTEP.
vni	Indicates the VNI. A positive integer from 1-16000000.
ip_list	The list of IP addresses. A string in the following format: `[“10.0.1.1”, “10.2.2.1”]`.

Returns

A dictionary containing the following elements:

Parameter	Description
return status	Return status of the API call: ● `TRUE`: Successfully set the Remote VTEP. ● `FALSE`: Setting the Remote VTEP failed.
error message	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

class VxlanInfo()

This class provides functions to get VXLAN information.

get_info_l3vnid()

Gets L3VNID information.

Syntax

`get_info_l3vnid``()`

Returns

A dictionary containing the following elements:

Parameter	**Description**
count	The number of items available in the l3vnid-table (integer).
vni	The VXLAN network identifier. An integer from 1-16777214.
vrfid	The VRF ID (integer).

get_info_static_route()

Gets the VXLAN static route information.

Syntax

`get_info_static_route``()`

Returns

A dictionary containing the following elements:

Parameter	**Description**
vlan	The VLAN ID (integer).
vni	The VXLAN network identifier. An integer from 1-16777214.
port_name	The port name (string).
tunnel_ip	The tunnel IP address (string).
subnet	The destination subnet. A string in the following format: "`xxx.xxx.xxx.xxx/<mask num>`".
gw_ip	The gateway IP address (string).
gw_mac	The gateway MAC address. A string in the following format: "`xxxx.xxxx.xxxx`".

Appendix A. Error Messages

Error messages thrown by the Lenovo Cloud NOS Python API fall into the following categories:

- Validation errors

  If the argument for a function is not of a valid type, such as a string when an integer is expected or a string other than an expected string, an error will be thrown. For example, in the vlanApi module, when validating arguments for `vlanApi.VlanSystem().python_update_vlan_admin_state()`, if the value of *admin_state* is not `up` or `down`, the Python interpreter will throw the following error message:

  ```
  Error: Invalid admin state. Valid options (up, down)
  ```

- Operating system errors

  If the internal module or server functionality encounters an error, it will throw an operating system error. For example, if you call the LLDP module function `lldpApi.LldpStats().python_lldp_get_interface(`*ifname*`)` with an interface name that does not exist, the following error is thrown:

  ```
  Error: Interface name not valid
  ```

Appendix B. Getting help and technical assistance

If you need help, service, or technical assistance or just want more information about Lenovo products, you will find a wide variety of sources available from Lenovo to assist you.

Use this information to obtain additional information about Lenovo and Lenovo products, and determine what to do if you experience a problem with your Lenovo system or optional device.

Note: This section includes references to IBM web sites and information about obtaining service. IBM is Lenovo's preferred service provider for the System x, Flex System, and NeXtScale System products.

Before you call, make sure that you have taken these steps to try to solve the problem yourself.

If you believe that you require warranty service for your Lenovo product, the service technicians will be able to assist you more efficiently if you prepare before you call.

- Check all cables to make sure that they are connected.
- Check the power switches to make sure that the system and any optional devices are turned on.
- Check for updated software, firmware, and operating-system device drivers for your Lenovo product. The Lenovo Warranty terms and conditions state that you, the owner of the Lenovo product, are responsible for maintaining and updating all software and firmware for the product (unless it is covered by an additional maintenance contract). Your service technician will request that you upgrade your software and firmware if the problem has a documented solution within a software upgrade.
- If you have installed new hardware or software in your environment, check the Lenovo ServerProven website to make sure that the hardware and software is supported by your product.
- Go to the Lenovo Support portal to check for information to help you solve the problem.
- Gather the following information to provide to the service technician. This data will help the service technician quickly provide a solution to your problem and ensure that you receive the level of service for which you might have contracted.
  - Hardware and Software Maintenance agreement contract numbers, if applicable
  - Machine type number (if applicable–Lenovo 4-digit machine identifier)
  - Model number
  - Serial number
  - Current system UEFI and firmware levels
  - Other pertinent information such as error messages and logs
- Start the process of determining a solution to your problem by making the pertinent information available to the service technicians. The IBM service technicians can start working on your solution as soon as you have completed and submitted an Electronic Service Request.

You can solve many problems without outside assistance by following the troubleshooting procedures that Lenovo provides in the online help or in the Lenovo product documentation. The Lenovo product documentation also describes the diagnostic tests that you can perform. The documentation for most systems, operating systems, and programs contains troubleshooting procedures and explanations of error messages and error codes. If you suspect a software problem, see the documentation for the operating system or program.

Appendix C. Notices

Lenovo may not offer the products, services, or features discussed in this document in all countries. Consult your local Lenovo representative for information on the products and services currently available in your area.

Any reference to a Lenovo product, program, or service is not intended to state or imply that only that Lenovo product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Lenovo intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any other product, program, or service.

Lenovo may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Lenovo (United States), Inc.
1009 Think Place - Building One
Morrisville, NC 27560
U.S.A.

Attention: Lenovo Director of Licensing

LENOVO PROVIDES THIS PUBLICATION "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. Lenovo may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

The products described in this document are not intended for use in implantation or other life support applications where malfunction may result in injury or death to persons. The information contained in this document does not affect or change Lenovo product specifications or warranties.

Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of Lenovo or third parties. All information contained in this document was obtained in specific environments and is presented as an illustration. The result obtained in other operating environments may vary.

Lenovo may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any references in this publication to non-Lenovo Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this Lenovo product, and use of those Web sites is at your own risk.

Any performance data contained herein was determined in a controlled environment. Therefore, the result obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Trademarks

Lenovo, the Lenovo logo, Flex System, System x, NeXtScale System, and X-Architecture are trademarks of Lenovo in the United States, other countries, or both.

Intel and Intel Xeon are trademarks of Intel Corporation in the United States, other countries, or both.

Internet Explorer, Microsoft, and Windows are trademarks of the Microsoft group of companies.

Linux is a registered trademark of Linus Torvalds.

Other company, product, or service names may be trademarks or service marks of others.

Important Notes

Processor speed indicates the internal clock speed of the microprocessor; other factors also affect application performance.

CD or DVD drive speed is the variable read rate. Actual speeds vary and are often less than the possible maximum.

When referring to processor storage, real and virtual storage, or channel volume, KB stands for 1 024 bytes, MB stands for 1 048 576 bytes, and GB stands for 1 073 741 824 bytes.

When referring to hard disk drive capacity or communications volume, MB stands for 1 000 000 bytes, and GB stands for 1 000 000 000 bytes. Total user-accessible capacity can vary depending on operating environments.

Maximum internal hard disk drive capacities assume the replacement of any standard hard disk drives and population of all hard-disk-drive bays with the largest currently supported drives that are available from Lenovo.

Maximum memory might require replacement of the standard memory with an optional memory module.

Each solid-state memory cell has an intrinsic, finite number of write cycles that the cell can incur. Therefore, a solid-state device has a maximum number of write cycles that it can be subjected to, expressed as total bytes written (TBW). A device that has exceeded this limit might fail to respond to system-generated commands or might be incapable of being written to. Lenovo is not responsible for replacement of a device that has exceeded its maximum guaranteed number of program/erase cycles, as documented in the Official Published Specifications for the device.

Lenovo makes no representations or warranties with respect to non-Lenovo products. Support (if any) for the non-Lenovo products is provided by the third party, not Lenovo.

Some software might differ from its retail version (if available) and might not include user manuals or all program functionality.

Open Source Information

This Lenovo Switch may include software made publicly available by Lenovo, including software licensed under the General Public License and/or the Lesser General Public License (the "open source software").

You may obtain the corresponding machine-readable copy for any such open source software licensed under the General Public License and/or the Lesser General Public License (or any other license requiring us to make a written offer to provide corresponding source code to you) from Lenovo for a period of three years without charge except for the cost of media, shipping, and handling, upon written request to Lenovo. This offer is valid to anyone in receipt of this Lenovo Switch. You may send your request in writing to the address below accompanied by a check or money order for $5 to:

Lenovo Legal Department
8001 Development Dr.
Morrisville, NC 27560
U.S.A.

Attention: Open Source Team / Source Code Requests

Please include both a "NOS" Release version and model number or Machine Type (MT) of your Lenovo Switch as part of your request. Be sure to provide a return address.

The open source software is distributed in hope it will be useful, but WITHOUT ANY WARRANTY; without even the implied warranty of MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See for example the GNU General Public License and/or the Lesser General Public License for more information.

Visit https://datacentersupport.lenovo.com/us/en/ and enter the model number or Machine Type (MT) for your Switch to view additional information regarding licenses, acknowledgments and required copyright notices for the open source software used on your Switch.

Recycling Information

Lenovo encourages owners of information technology (IT) equipment to responsibly recycle their equipment when it is no longer needed. Lenovo offers a variety of programs and services to assist equipment owners in recycling their IT products. For information on recycling Lenovo products, go to:

http://www.lenovo.com/recycling

Particulate Contamination

Attention: Airborne particulates (including metal flakes or particles) and reactive gases acting alone or in combination with other environmental factors such as humidity or temperature might pose a risk to the device that is described in this document.

Risks that are posed by the presence of excessive particulate levels or concentrations of harmful gases include damage that might cause the device to malfunction or cease functioning altogether. This specification sets forth limits for particulates and gases that are intended to avoid such damage. The limits must not be viewed or used as definitive limits, because numerous other factors, such as temperature or moisture content of the air, can influence the impact of particulates or environmental corrosives and gaseous contaminant transfer. In the absence of specific limits that are set forth in this document, you must implement practices that maintain particulate and gas levels that are consistent with the protection of human health and safety. If Lenovo determines that the levels of particulates or gases in your environment have caused damage to the device, Lenovo may condition provision of repair or replacement of devices or parts on implementation of appropriate remedial measures to mitigate such environmental contamination. Implementation of such remedial measures is a customer responsibility..

Contaminant	**Limits**
Particulate	• The room air must be continuously filtered with 40% atmospheric dust spot efficiency (MERV 9) according to ASHRAE Standard 52.2[1]. • Air that enters a data center must be filtered to 99.97% efficiency or greater, using high-efficiency particulate air (HEPA) filters that meet MIL-STD-282. • The deliquescent relative humidity of the particulate contamination must be more than 60%[2]. • The room must be free of conductive contamination such as zinc whiskers.
Gaseous	• Copper: Class G1 as per ANSI/ISA 71.04-1985[3] • Silver: Corrosion rate of less than 300 Å in 30 days

[1] ASHRAE 52.2-2008 - *Method of Testing General Ventilation Air-Cleaning Devices for Removal Efficiency by Particle Size*. Atlanta: American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc.

[2] The deliquescent relative humidity of particulate contamination is the relative humidity at which the dust absorbs enough water to become wet and promote ionic conduction.

[3] ANSI/ISA-71.04-1985. *Environmental conditions for process measurement and control systems: Airborne contaminants*. Instrument Society of America, Research Triangle Park, North Carolina, U.S.A.

Telecommunication Regulatory Statement

This product may not be certified in your country for connection by any means whatsoever to interfaces of public telecommunications networks. Further certification may be required by law prior to making any such connection. Contact a Lenovo representative or reseller for any questions.

Electronic Emission Notices

When you attach a monitor to the equipment, you must use the designated monitor cable and any interference suppression devices that are supplied with the monitor.

Federal Communications Commission (FCC) Statement

Note: This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference, in which case the user will be required to correct the interference at his own expense.

Properly shielded and grounded cables and connectors must be used to meet FCC emission limits. Lenovo is not responsible for any radio or television interference caused by using other than recommended cables and connectors or by unauthorized changes or modifications to this equipment. Unauthorized changes or modifications could void the user’s authority to operate the equipment.

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that might cause undesired operation.

Industry Canada Class A Emission Compliance Statement

This Class A digital apparatus complies with Canadian ICES-003.

Avis de Conformité à la Réglementation d'Industrie Canada

Cet appareil numérique de la classe A est conforme à la norme NMB-003 du Canada.

Australia and New Zealand Class A Statement

Attention: This is a Class A product. In a domestic environment this product may cause radio interference in which case the user may be required to take adequate measures.

European Union - Compliance to the Electromagnetic Compatibility Directive

This product is in conformity with the protection requirements of EU Council Directive 2004/108/EC (until April 19, 2016) and EU Council Directive 2014/30/EU (from April 20, 2016) on the approximation of the laws of the Member States relating to electromagnetic compatibility. Lenovo cannot accept responsibility for any failure to satisfy the protection requirements resulting from a non-recommended modification of the product, including the installation of option cards from other manufacturers.

This product has been tested and found to comply with the limits for Class A equipment according to European Standards harmonized in the Directives in compliance. The limits for Class A equipment were derived for commercial and industrial environments to provide reasonable protection against interference with licensed communication equipment.

Lenovo, Einsteinova 21, 851 01 Bratislava, Slovakia

Warning: This is a Class A product. In a domestic environment this product may cause radio interference in which case the user may be required to take adequate measures.

Germany Class A Statement

Deutschsprachiger EU Hinweis:

Hinweis für Geräte der Klasse A EU-Richtlinie zur Elektromagnetischen Verträglichkeit

Dieses Produkt entspricht den Schutzanforderungen der EU-Richtlinie 2014/30/EU (früher 2004/108/EC) zur Angleichung der Rechtsvorschriften über die elektromagnetische Verträglichkeit in den EU-Mitgliedsstaaten und hält die Grenzwerte der Klasse A der Norm gemäß Richtlinie.

Um dieses sicherzustellen, sind die Geräte wie in den Handbüchern beschrieben zu installieren und zu betreiben. Des Weiteren dürfen auch nur von der Lenovo empfohlene Kabel angeschlossen werden. Lenovo übernimmt keine Verantwortung für die Einhaltung der Schutzanforderungen, wenn das Produkt ohne Zustimmung der Lenovo verändert bzw. wenn Erweiterungskomponenten von Fremdherstellern ohne Empfehlung der Lenovo gesteckt/eingebaut werden.

Deutschland:

Einhaltung des Gesetzes über die elektromagnetische Verträglichkeit von Betriebsmitteln

Dieses Produkt entspricht dem „Gesetz über die elektromagnetische Verträglichkeit von Betriebsmitteln“ EMVG (früher „Gesetz über die elektromagnetische Verträglichkeit von Geräten“). Dies ist die Umsetzung der EU-Richtlinie 2014/30/EU (früher 2004/108/EC) in der Bundesrepublik Deutschland.

Zulassungsbescheinigung laut dem Deutschen Gesetz über die elektromagnetische Verträglichkeit von Betriebsmitteln, EMVG vom 20. Juli 2007 (früher Gesetz über die elektromagnetische Verträglichkeit von Geräten), bzw. der EMV EU Richtlinie 2014/30/EU (früher 2004/108/EC), für Geräte der Klasse A.

Dieses Gerät ist berechtigt, in Übereinstimmung mit dem Deutschen EMVG das EG-Konformitätszeichen - CE - zu führen. Verantwortlich für die Konformitätserklärung nach Paragraf 5 des EMVG ist die Lenovo (Deutschland) GmbH, Meitnerstr. 9, D-70563 Stuttgart.

Informationen in Hinsicht EMVG Paragraf 4 Abs. (1) 4:

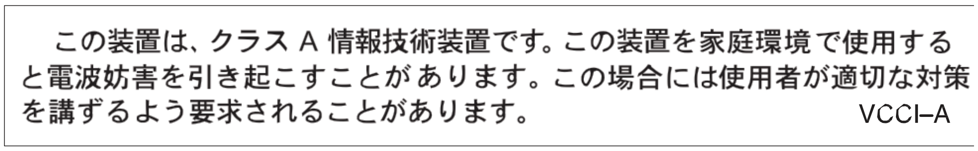
Das Gerät erfüllt die Schutzanforderungen nach EN 55024 und EN 55022 Klasse A.

Nach der EN 55022: „Dies ist eine Einrichtung der Klasse A. Diese Einrichtung kann im Wohnbereich Funkstörungen verursachen; in diesem Fall kann vom Betreiber verlangt werden, angemessene Maßnahmen durchzuführen und dafür aufzukommen."

Nach dem EMVG: „Geräte dürfen an Orten, für die sie nicht ausreichend entstört sind, nur mit besonderer Genehmigung des Bundesministers für Post und Telekommunikation oder des Bundesamtes für Post und Telekommunikation betrieben werden. Die Genehmigung wird erteilt, wenn keine elektromagnetischen Störungen zu erwarten sind." (Auszug aus dem EMVG, Paragraph 3, Abs. 4). Dieses Genehmigungsverfahrenist nach Paragraph 9 EMVG in Verbindung mit der entsprechenden Kostenverordnung (Amtsblatt 14/93) kostenpflichtig.

Anmerkung: Um die Einhaltung des EMVG sicherzustellen sind die Geräte, wie in den Handbüchern angegeben, zu installieren und zu betreiben.

Japan VCCI Class A Statement



この装置は、クラス A 情報技術装置です。この装置を家庭環境で使用すると電波妨害を引き起こすことがあります。この場合には使用者が適切な対策を講ずるよう要求されることがあります。 VCCI–A

This is a Class A product based on the standard of the Voluntary Control Council for Interference (VCCI). If this equipment is used in a domestic environment, radio interference may occur, in which case the user may be required to take corrective actions.

Japan Electronics and Information Technology Industries Association (JEITA) Statement

高調波ガイドライン適合品

Japan Electronics and Information Technology Industries Association (JEITA) Confirmed Harmonics Guidelines (products less than or equal to 20 A per phase)

高調波ガイドライン準用品

Japan Electronics and Information Technology Industries Association (JEITA) Confirmed Harmonics Guidelines with Modifications (products greater than 20 A per phase).

Korea Communications Commission (KCC) Statement

이 기기는 업무용(A급)으로 전자파적합기기로
서 판매자 또는 사용자는 이 점을 주의하시기
바라며, 가정외의 지역에서 사용하는 것을 목
적으로 합니다.

This is electromagnetic wave compatibility equipment for business (Type A). Sellers and users need to pay attention to it. This is for any areas other than home.

Russia Electromagnetic Interference (EMI) Class A statement

ВНИМАНИЕ! Настоящее изделие относится к классу А.
В жилых помещениях оно может создавать радиопомехи, для
снижения которых необходимы дополнительные меры

People's Republic of China Class A electronic emission statement

中华人民共和国"A类"警告声明

声 明

此为A级产品，在生活环境中，该产品可能会造成无线电干扰。在这种情况下，可能需要用户对其干扰采取切实可行的措施。

Taiwan Class A compliance statement

警告使用者:
這是甲類的資訊產品，在
居住的環境中使用時，可
能會造成射頻干擾，在這
種情況下，使用者會被要
求採取某些適當的對策。