Lenovo Network

Python Programming Guide

For Cloud Network Operating System 10.2

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 (November 2016)

Preface

The *Lenovo Network Python Programming Guide for Cloud NOS 10.2* describes how to configure and use the Cloud Network Operating System 10.2 software on the RackSwitch G8272 (referred to as G8272 throughout this document). For documentation on installing the switch physically, see the *Installation Guide* for your G8272.

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 G8272 is available in the following guides:

- *Lenovo RackSwitch G8272 Installation Guide*
- *Lenovo Network Application Guide for Lenovo Cloud Network Operating System 10.2*
- *Lenovo Network Command Reference for Lenovo Cloud Network Operating System 10.2*

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. It also depicts on-screen computer output and prompts.	View the `readme.txt` file. `Main#`
`ABC123`	This bold type appears in command examples. It shows text that must be typed in exactly as shown.	`Main#` **`sys`**
<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. This also shows book titles, special terms, or words to be emphasized.	To establish a Telnet session, enter: `host#` **`telnet`** *<IP address>* Read your *User's Guide* thoroughly.
[ ]	Command items shown inside brackets are optional and can be used or excluded as the situation demands. Do not type the brackets.	`host#` **`ls`** [**`-a`**]
|	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.	`host#` **`set left|right`**
AaBbCc123	This block type depicts menus, buttons, and other controls that appear in Web browsers and other graphical interfaces.	Click the **Save** button.

Contents

Chapter 1. Introduction to Python Scripting

The Lenovo Cloud Network Operating System (CNOS) version 10.2 Python function API is a set of libraries of API functions that are embedded into the Lenovo RackSwitch G8272 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 G8272 version 10.2.

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 G8272 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 G8272, 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 RackSwitch G8272 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 G8272, see Chapter 3, "Managing Scripts.

Running a Basic Script

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

```
SWITCH#
SWITCH# display script helloWorld.py

print "Hello world!"

SWITCH# python helloWorld.py
Hello world!
SWITCH#
```

Running a Basic Script with Arguments

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

```
SWITCH# display 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
SWITCH#
```

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 **`exit()`** or press **Ctrl+D** (end of file).

Chapter 3. Managing Python Scripts

Script files are saved in persistent storage on the G8272, 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 G8272 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://<IPv4 address>/<remote script> obs <local script>
[vrf {<VRF name>|default|management}]
```

where:

Parameter	Description
IPv4 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 name	The VRF name.

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://<IPv4 address>/<remote script path>
[vrf {<VRF name>|default|management}]
```

where:

Parameter	Description
IPv4 address	The IPv4 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.

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# display script
```

Viewing Script File Content

To view a script file, use the following command

```
SWITCH# display script <script filename>
```

Viewing Configured Scheduler Jobs

To view the scheduler jobs configured on the G8272, use the following command:

```
SWITCH# display 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 RackSwitch G8272 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.

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 <time parameter>
```

where:

Parameter	Description
Python script	The name of the Python script.
arguments	Any arguments the script needs.
time parameter	When the script is to be run.

The following time parameters are available:

Parameter	Description
Crontab format	Run a script periodically, in the format of the UNIX `crontab` utility. See xx for more information on this argument.
daily	Run once a day at 12:00:01 AM.
hourly	Run once an hour at the beginning of the hour.
monthly	Run once a month at 12:00:01 AM the first day of the month.
reboot	Run at startup.
weekly	Run once a week at 12:00:01 AM Sunday.
yearly	run once a year at 12:00:01 AM on January 1.

For example, to create a job to run the Python script `myScript.py` with arguments "`1 2 3`" on a daily basis, enter:

```
SWITCH(config)# script-job myScript.py "1 2 3" time daily
```

Using Crontab Format Arguments

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

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

where:

Parameter	Description
minute	An integer from 0-59.
hour	An integer from 0-23.
day of month	An integer from 1-31.
month	An integer from 1-12 or the three-letter abbreviation for the month: • Jan • Feb • Mar • Apr • May • Jun • Jul • Aug • Sep • Oct • Nov • Dec
day of week	An integer from 1-7 or the three-letter abbreviation for the day of the week: • Sun • Mon • Tues • Wed • Thu • Fri • Sat

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` with arguments "`1 2 3`" every July 4th at 10:55 AM, enter:

```
SWITCH(config)# script-job myScript.py “1 2 3” time “55 10 4 7 *”
```

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` with arguments "`1 2 3`" every July 4th-July 11th at 10:55 AM and 10:55 PM, enter:

```
SWITCH(config)# script-job myScript.py “1 2 3” 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` with arguments "`1 2 3`" every July 4th every two hours, enter:

```
SWITCH(config)# script-job myScript.py “1 2 3” time “0 */2 4 7 *”
```

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 command'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 command to be run at 4:30 AM 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# display script running
```

Stopping a Running Job

To stop a running job:

1. :Check the list of running scripts:

```
SWITCH# display 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# display script-log
```

To view an individual log file, enter:

```
SWITCH# display 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 RackSwitch G8272. 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 an SMI client connection.

  Note: You must import this module before importing anything else.

- `bootinfoApi`: Functions that get and update switch boot properties.
- `hostpCpyApi`: Functions that update image and configuration files via TFTP.
- `vlanApi`: Functions that configure VLAN properties.
- `lldpApi`: Functions that get and set LLDP configurations and LLDP interface configurations, and functions that get LLDP statistics and LLDP neighbor information.
- `lacpApi`: Functions that provide system and interface LACP configuration.
- `lagApi`: Classes and functions that configure LAGs and find information about LAGs and associated interfaces.
- `mstpApi`: Classes and functions that configure and get information about MSTP.
- `igmpApi`: classes and functions in this module manage Internet Group Management Protocol (IGMP) snooping.
- `ipApi`: Functions that manage IP addresses.
- `routeApi`: Functions that manage static routes.
- `vrrpApi`: Functions that manage Virtual Router Redundancy Protocol (VRRP).
- `arpApi`: Functions that manage Address Resolution Protocol (ARP).
- `vrfApi`: A function that manage Virtual Routing and Forwarding (VRF).
- `platformApi`: Functions that get and set port information.
- `bgpApi`: Functions that get and set Border Gateway Protocol (BGP) information.
- `dhcpApi`: Functions that get and set Configuration Protocol (DHCP) information.

The following is a sample Python shell session:

```
SWITCH# python
>>> import systemApi
>>> systemApi.client_connect()
>>> systemApi.SystemInfo().get_systemInfo()
{'switch_type': 'SWITCH', 'fw_version': '10.1.0.0'}
>>> 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 10.1.0.0'}]
>>> 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 G8272.

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:

System Module

These classes and functions manage and monitor 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()

The function in this class returns basic system properties.

get_systemInfo()

Returns switch type and version number.

Syntax

```
get_systemInfo()
```

Returns

A dictionary containing the switch type and version number.

get_env_fan()

Returns environment fan information for the switch.

Syntax

```
get_env_fan()
```

Returns

One or more dictionaries with fan properties:

Element	Description
`Fan` *<n>*	A dictionary containing the following: • `module` • `air-flow` • `speed-percentage` • `speed-rpm`
`module`	Module type
`air-flow`	Air flow direction; one of `Front-to-Back`, `Back-to-Front`, `Not Installed`
`speed percent`	Percent of RPMs; an integer from 0-100.
`speed-rpm`	Speed in Revolutions Per Minute; a positive integer.

get_env_power()

Returns power supply information for the switch.

Syntax

```
get_env_power()
```

Returns

One or more dictionaries with power supply properties:

Element	Description
`Power Supply` *<n>*	A dictionary containing the following: • `Name` • `Manufacturer` • `Model` • `State`
`Name`	Power supply name (str).
`Manufacturer`	Power supply manufacturer (str).
`Model`	Power supply model (str).
`State`	Power supply state (str).

get_env_temperature()

Returns power supply information for the switch.

Syntax

```
get_env_temperature()
```

Returns

A dictionary with switch temperature properties:

Element	Description
`CPU Local`	A dictionary containing the following: • `Temp` • `State`
`Ambient`	A dictionary containing the following: • `Temp` • `State`
`Hotspot`	A dictionary containing the following: • `Temp` • `State`
`Temp`	Temperature, in Celsius (int).
`State`	Power supply state; one of `OK`, `FAULT`.
`Temperature threshold`	A dictionary containing the following: • `System warning` • `System shutdown` • `System set point`
`System warning`	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:

Element	Description
`Serial Number`	The system serial number (str).

get_system_inventory()

Returns system inventory details.

Syntax

```
get_system_inventory()
```

Returns

A dictionary containing system inventory information::

Element	Description
`Uptime`	The system uptime, in seconds.
`Name`	System name.
`Service LED`	Whether or not the Service LED is enabled; one of `enabled`, `disabled`.
`sysObjID`	
`Description`	System description.
`Model`	System model.
`Manufacture Date`	System Manufacture Date.
`Serial Number`	System Serial Number.
`PCB Assembly`	System PCB Assembly.
`Electronic Serial Number`	System Electronic Serial Number.
`Firmware Revision`	System Firmware Revision.
`Software Revision`	System Software Revision.
`Uuid`	System UUID.
`Last reset Reason`	System last reset reason.

get_hostname()

Returns the hostname of the system.

Syntax

```
get_hostname()
```

Returns

The system host name:

Element	Description
`hostname`	The system host name; a string up to 64 characters long.

set_hostname()

Sets the hostname of the system.

Syntax

`set_hostname(`*<hostname>*`)`

where:

Element	Description
hostname	The system host name; a string up to 64 characters long.

Returns

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

get_system_core()

Get system core details.

Syntax

```
get_system_core()
```

Returns

A dictionary containing system core details:

Element	Description
`File` *n*	A dictionary containing elements `name` and `date`.
`name`	File name (Str).
`date`	Date (Str).

Boot Information Module

The class in 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()

The functions in this class manage boot properties.

get_boot()

Get detailed boot information.

Syntax

```
get_boot()
```

Returns

A dictionary containing boot information:

Element	Description
`ztp`	Zero Touch Provisioning (ZTP) status; one of `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`	A string; one of `empty` or a string containing the version and time downloaded.
`boot software`	Boot image setting; one of `active`, `standby`.
`scheduled reboot`	A string; one of `none` or a string containing the date and time of the next scheduled reboot.
`port mode`	The port mode (Str). Default value: `default mode`.

get_boot_ztp()

Get detailed Zero Touch Provisioning (ZTP) boot information.

Syntax

```
get_boot_ztp()
```

Returns

A dictionary containing ZTP boot information:

Element	Description
`ztp`	Zero touch feature status; one of `Enable`, `Forcedly Enabled`, `Forcedly Disabled`. Default value: `Enable`.

set_boot_ztp()

Set detailed Zero Touch Provisioning (ZTP) boot information.

Syntax

`set_boot_ztp(`*state*`)`

where:

Element	Description
state	Zero touch feature status; one of `Enable`, `Forcedly Enabled`, `Forcedly Disabled`.

Returns

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

get_boot_image()

Get boot image status.

Syntax

```
get_boot_image()
```

Returns

A dictionary containing boot software status:

Element	Description
`boot software`	Boot image status; one of `active`, `standby`.

set_boot_image()

Set next boot image.

Syntax

`set_boot_image(`*image*`)`

where:

Element	Description
image	Next boot image (Str); one of `active`, `standby`.

Returns

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

HostpCpy Module

The following module updates 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 has methods for updating the image and configuration files via TFTP.

update_startup_cfg_tftp()

Update the startup configuration using TFTP.

Syntax

`update_startup_cfg_tftp(`*serverip*`,`*cfgfile*`,`*vrf_name*`)`

where:

Variable	Description
serverip	The server IP address (Str); 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: `none`.

Returns

A dictionary that indicates the startup configuration update status.:

Element	Description
`status`	Configuration update status (Str).

update_image_tftp()

Update the image using TFTP.

Syntax

`update_image_tftp(`*serverip*`,`*imgfile*`,`*imgtype*`,`*vrf_name*`)`

where:

Variable	Description
serverip	The server IP address (Str); a valid IP address.
imgfile	The image file name; a string up to 256 characters long.
imgtype	System image type (Str); one of `all`, `boot`, `onie`, `os`.
vrf_name	(Optional) The VRF name; a string up to 64 characters long. Default value: `none`.

Returns

A dictionary that indicates the image update status.:

Element	Description
`status`	Image update status (Str).

get_update_image_status()

Get the image update status.

Syntax

```
get_update_image_status()
```

Returns

A dictionary that indicates the image update status.:

Element	Description
`status`	Image update status (Str).

switch_reboot()

Halt the system and perform a cold restart.

Syntax

```
switch_reboot()
```

Returns

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

VLAN Module

The following classes configure VLAN properties. To use this module, in the Python file or in the Python interpreter, enter:

```
import vlanApi
```

class VlanSystem()

This class has methods for getting and setting VLAN configurations.

python_get_vlan()

Get 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:

Variable	Description
vid	The VLAN ID (Int)

Returns

A dictionary with VLAN properties if (*vid*) is a valid VLAN ID or of all VLANs if *vid* is `None`.:

Element	Description
`vlan_name`	The name of the VLAN (Str).
`interfaces`	List of interface members of the VLAN containing the following properties: • `pvid` • `bridgeport_mode` • `if_name`
`pvid`	Native VLAN ID for a trunk port; an integer from 1-3999. Default value: 1.
`bridgeport_mode`	Bridge port mode (Str); one of `access`, `trunk`.
`if_name`	Ethernet interface name (Str)
`admin_state`	The admin status of the VLAN (Str); one of `up`, `down`.
`vlan_id`	The VLAN identifier; an integer from 1-3999.
`mst_inst_id`	MST instance identifier; an integer from 1-64. 0 refers to CIST. Default value: 0.

python_create_vlan()

Create a VLAN.

Syntax

`python_create_vlan(`*vlan_info*`)`

where:

Variable	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 (Str). 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 (Str); one of up, down.

Returns

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

python_delete_vlan()

Delete a VLAN

Syntax

`python_delete_vlan(`*vid*`)`

where:

Variable	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()

Update VLAN name

Syntax

`python_update_vlan_name(`*vid*`,` *vlan_name*`)`

where:

Variable	Description
vid	VLAN ID (Int)
vlan_name	The VLAN name (Str)

Returns

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

python_update_vlan_admin_state()

Update VLAN admin state

Syntax

`python_update_vlan_admin_state(`*vid*`,` *admin_state*`)`

where:

Variable	Description
vid	VLAN ID (Int)
admin_state	The administrative state (Str); one of `up`, `down`.

Returns

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

class VlanEthIf

The methods in this class affect VLAN properties of Ethernet interfaces.

python_get_vlan_properties()

Get the VLAN properties of an Ethernet Interface or of all Ethernet Interfaces if the interface name (*ifname* argument) is `None`.

Syntax

`python_get_vlan_properties(`*ifname*`)`

where:

Variable	Description
ifname	The interface name (Str)

Returns

A dictionary with VLAN properties of the specified interface or of all interfaces:

Element	Description
`if_name`	Ethernet interface name (Str)
`bridgeport_mode`	Bridge port mode (Str); one of `access`, `trunk`.
`pvid`	Native VLAN ID for a trunk port; an integer from 1-3999. Default value: 1.
`vlans`	A list of all VLANs attached to this interface.

python_update_vlan_properties()

Update VLAN Interface Propeties

Syntax

`python_update_vlan_properties(`*if_new_info*`)`

`where:`

Variable	Description
if_new_info	A dictionary with information that is being updated. At least one of the following properties must be defined in the dictionary: • `if_name` • `bridgeport_mode` • `pvid` • `vlans` (each with a `vlan_id`)
if_name	Ethernet interface name (Str)

Variable	Description
bridgeport_mode	Bridge port mode (Str); one of `access`, `trunk`.
pvid	Native VLAN ID for a trunk port; an integer from 1-3999. Default value: 1.
vlans	A list of all VLANs attached to this interface.

Returns

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

LLDP Module

The following classes provide functions for getting and setting LLDP configurations and LLDP interface configurations, and getting LLDP statistics and LLDP neighbor information. To use this module, in the Python file or in the Python interpreter, enter:

```
import lldpApi
```

class LldpSystem

The functions in this class get and set LLDP configurations and LLDP interface configurations.

python_lldp_get_reinit_delay()

Get the number of seconds until LLDP re-initialization is attempted on an interface

Syntax

```
python_lldp_get_reinit_delay()
```

Returns

The delay value, in seconds (Int).

python_lldp_get_msg_tx_interval()

Get the time interval in seconds between transmissions of LLDP messages

Syntax

```
python_lldp_get_msg_tx_interval()
```

Returns

The transmit interval value, in seconds (Int).

python_lldp_get_tx_delay()

Get the number of seconds for LLDP transmission delay

Syntax

```
python_lldp_get_tx_delay()
```

Returns

The transmit delay value, in seconds, (Int).

python_lldp_set_reinit_delay()

Set the time to wait, in seconds, before initializing an interface.

Syntax

`python_lldp_set_reinit_delay(`*reinit_delay*`)`

where:

Variable	Description
reinit_delay	The time to wait, in seconds (Int).

Returns

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

python_lldp_set_msg_tx_interval()

Set the rate, in seconds, for LLDP packets to be sent.

Syntax

`python_lldp_set_msg_tx_interval(`*tx_interval*`)`

where:

Variable	Description
tx_interval	The transmission rate, in seconds (Int).

Returns

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

python_lldp_set_tx_delay()

Set the delay time in seconds.

Syntax

`python_lldp_set_tx_delay(`*tx_delay*`)`

where:

Variable	Description
tx_delay	The transmission delay, in seconds (Int). **Note:** This value must not be greater than 25% of the transmit interval value.

Returns

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

class LldpNeighbor

The following methods get neighbor port information.

python_lldp_get_neighbor()

Get neighbor information of a port

Syntax

`python_lldp_get_neighbor(`*ifname*`)`

where:

Variable	Description
ifname	The interface port name (Str).

Returns

A dictionary containing information about the specified port.

Element	Description
`if_name`	The ethernet interface name (Str).
`capability`	Remote switch capability (Str). One or more of: `B` (Bridge) `R` (Router).
`rx ttl`	The receiving time-to-live (TTL) value (Long).
`system name`	Remote system name (Str).
`system description`	Remote system description (Str).

python_lldp_get_all_neighbor()

Get neighbor information of all ports

Syntax

`python_lldp_get_all_neighbor()`

Returns

A list of dictionaries containing information about all LLDP neighbor ports:

Element	Description
`if_name`	The ethernet interface name (Str).
`capability`	Remote switch capability (Str). One or more of: `B` (Bridge) `R` (Router).
`rx ttl`	The receiving time-to-live (TTL) value (Long).

Element	Description
`system name`	Remote system name (Str).
`system description`	Remote system description (Str).

class LldpStats

The method in this class gets LLDP statistics.

python_lldp_get_statistics()

Get LLDP port statistics

`python_lldp_get_statistics(`*ifname*`)`

where:

Variable	Description
ifname	The interface port name (Str).

Returns

A dictionary of LLDP statistics:

Element	Description
`total frames`	The total number of LLDP frames received (Int).
`total tlvs discarded`	The total number of LLDP TLVs discarded (Int).
`total frames transmitted`	The total number of LLDP frames transmitted (Int).
`total errrored frames`	The total number of frames received with errors (Int).
`total frames discarded`	The total number of discarded frames (Int).
`total entries aged`	The total number of aged-out entries (Int).
`total tlvs unrecognized`	The total number of unrecognized LLDP TLVs (Int).

class LldpInterface

The methods in this class get and set LLDP interface information.

python_lldp_get_interface()

Get LLDP interface admin status of a specific interface

Syntax

```
python_lldp_get_interface(ifname)
```

where:

Variable	Description
ifname	The interface port name (Str).

Returns

A dictionary of LLDP status information for the specified interface:

Element	Description
`if_name`	The ethernet interface name (Str).
`ena_lldp_rx`	Whether LLDP frame reception is enabled on a physical interface (Str); one of `yes`, `no`. Default value: `Yes`.
`ena_lldp_tx`	Whether LLDP frame transmission is enabled on a physical interface (Str); one of `yes`, `no`. Default value: `Yes`.

python_lldp_get_all_interface()

Get 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:

Element	Description
`if_name`	The ethernet interface name (Str).
`ena_lldp_rx`	Whether LLDP frame reception is enabled on a physical interface (Str); one of `yes`, `no`. Default value: `Yes`.
`ena_lldp_tx`	Whether LLDP frame transmission is enabled on a physical interface (Str); one of `yes`, `no`. Default value: `Yes`.

python_lldp_set_interface()

Set LLDP interface admin status based on the lldp_interface_admin_status dictionary values

Syntax

`python_lldp_set_interface(`*lldp_interface_port_status*`)`

where:

Variable	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 (Str).
ena_lldp_rx	Whether LLDP frame reception is enabled on a physical interface (Str); one of `yes`, `no`. Default value: `Yes`.
ena_lldp_tx	Whether LLDP frame transmission is enabled on a physical interface (Str); one of `yes`, `no`. Default value: `Yes`.

Returns

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

LACP Module

The class in this module has functions that provide system and interface LACP configuration. To use this module, in the Python file or in the Python interpreter, enter:

```
import lacpApi
```

class LacpSystem

This class contains methods to get and set an LACP system configuration.

python_lacp_get_sys_priority()

Get LACP system priority.

Syntax

```
python_lacp_get_sys_priority()
```

Returns

The LACP system priority (Int)

python_lacp_get_max_bundle()

Get the LACP max bundle, which is the supported maximum number of links per LAG .

Syntax

```
python_lacp_get_max_bundle()
```

Returns

The supported maximum number of links per LAG (Int)

python_lacp_get_all_link_details()

Get all LACP interface details.

Syntax

```
python_lacp_get_all_link_details()
```

Returns

A list of dictionaries containing LACP link details, where:

Element	Description
`if_name`	Ethernet interface name (Str). **Note:** The interface must exist.
`lag_mode`	LAG mode; one of `lacp_active`, `lacp_passive`

Element	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; one of `short`, `long` (default).

python_lacp_set_system()

Set LACP system priority

Syntax

`python_lacp_set_system(`*sys_prio*`)`

where:

Variable	Description
sys_prio	The interface system priority (Int).

Returns

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

LAG Module

The following module has a class and functions that configure LAGs and find 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 contains methods to configure and get information about LAG.

python_get_lag()

Get 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:

Element	Description
`lag_name`	LAG name (Str)
`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`
`if_name`	Ethernet interface name (Str). **Note:** The interface must exist.
`lag_mode`	LAG mode (Str); one of `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 (Str); one of `short`, `long`. Default value: `long`.

python_get_lag_id()

Get a list of all the LAG information for the specified LAG ID.

Syntax

`python_get_lag(<lag_id>)`

where:

Element	Description
lag_id	LAG identifier; a positive integer from 1-65535.

Returns

A dictionary containing LAG information for the device:

Element	Description
`lag_name`	LAG name (Str)
`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.
`if_name`	Ethernet interface name (Str). **Note:** The interface must exist.
`lag_mode`	LAG mode (Str); one of `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 (Str); one of `short`, `long`. Default value: `long`.
`suspend_individual`	Whether the LACP state is suspended or individual (Str); one of `Individual`, `Suspended`, `N/A`.

python_update_lag_id()

Update LAG information for the specified LAG ID.

Syntax

`python_update_lag_id(<lag_id>)`

where:

Element	Description
lag_id	LAG identifier; a positive integer from 1-65535.

where:

Returns

A dictionary containing LAG information for the specified LAG ID:

Element	Description
`lag_name`	LAG name (Str).
`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 (Str). **Note:** The interface must exist.
`lag_mode`	LAG mode (Str); one of `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 (Str); one of `short`, `long`. Default value: `long`.
`suspend_individual`	Whether the LACP state is suspended or individual (Str); one of `Individual`, `Suspended`, `N/A`.

python_update_lag_id_details()

Update LAG information for the specified LAG ID.

Syntax

`python_update_lag_id_details(<lag>)`

where *lag* is a dictionary containing the following elements:

where:

Element	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 (Str). **Note:** The interface must exist.
`lag_mode`	LAG mode (Str); one of `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 (Str); one of `short`, `long`. Default value: `long`.

where:

Returns

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

A dictionary containing LAG information for the specified LAG ID:

python_create_lag_id()

Creates a new LAG with the information provided.

Syntax

`python_create_lag_id(`*lag*`)`

where:

Variable	Description
LAG	A dictionary containing LAG information for the specified LAG.

This dictionary contains:

Element	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 (Str). **Note:** The interface must exist.
`lag_mode`	LAG mode (Str); one of `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 (Str); one of `short`, `long`. Default value: `long`.

Returns

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

python_delete_lag_all()

Delete 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:

Element	Description
lag_id	LAG identifier; a positive integer from 1-65535.

Returns

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

MSTP Module

The following module has classes and functions that configure and get information about MSTP. To use this module, in the Python file or in the Python interpreter, enter:

```
import mstpApi
```

class MSTP

This class contains methods that 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:

Variable	Description
region_name	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; string up to 32 characters long.

python_mstp_get_revision()

Gets the revision number for the MSTP bridge

Syntax

```
python_mstp_set_revision()
```

Returns

The MSTP revision number (Int).

python_mstp_set_revision()

Sets the revision number for the MSTP bridge

Syntax

`python_mstp_set_revision(`*<revision>*`)`

where:

Variable	Description
revision	The MSTP revision number (Int).

Returns

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

class MstpInstance

This class contains methods that control MSTP instances.

python_mstp_add_instance()

Adds an MSTP instance.

Syntax

`python_mstp_add_instance(`*<instance_id>*`,`*<vlan_list>*`)`

where:

Element	Description
instance_id	MST 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 1-3999.

Returns

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

python_mstp_delete_instance()

Deletes an MSTP instance.

Syntax

`python_mstp_delete_instance([<instance_id>])`

where:

Variable	Description
instance_id	(Optional) MST 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:

Variable	Description
instance_id	MST 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 1-3999.

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:

Variable	Description
instance_id	MST 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. Default value: 32768.

Returns

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

python_mstp_check_instance_exist()

Check whether the specified MSTP instance exists.

Syntax

`python_mstp_check_instance_exist(`<*instance_id*>`)`

where:

Variable	Description
instance_id	MST 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 If the instance ID is invalid, returns an error along with `False`.

class MstpInterface

This class contains methods that 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:

Variable	Description
ifname	The name of the interface (Str)
instance_id	MST instance ID; an integer from 0-64. Instance 0 refers to the CIST.

Returns

The MSTP path cost (Int).

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:

Variable	Description
ifname	The name of the interface (Str). **Note:** The interface must exist.
instance_id	MST instance ID; an integer from 0-64. Instance 0 refers to the CIST.
path_cost	The port path-cost value on the specified MST instance; either an integer from 1-200000000 or `auto` (default) to base the path-cost on port speed.

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:

Variable	Description
ifname	The name of the interface (Str). **Note:** The interface must exist.
instance_id	MST instance ID; an integer from 0-64. Instance 0 refers to the CIST.

Returns

The port priority (Int):

Element	Description
`port_prio`	The port priority, in increments of 32, on the specified MST instance; a multiple of 32 from 0-224. Default value: 128.

python_mstp_set_port_priority()

Syntax

`python_mstp_set_port_priority(`*<ifname>*`,`*<instance_id>*`,`*<port_prio>*`)`

where:

Variable	Description
ifname	The name of the interface (Str). **Note:** The interface must exist.
instance_id	MST 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 MST instance; a multiple of 32 from 0-224. Default value: 128.

Returns

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

IGMP Module

The classes and functions in this module manage Internet Group Management Protocol (IGMP) snooping. To use this module, in the Python file or in the Python interpreter, enter:

```
import igmpApi
```

class IgmpSnooping

This class contains methods for getting and setting IGMP snooping status.

python_igmp_snoop_get()

Get the IGMP snooping status on the device.

Syntax

```
python_igmp_snoop_get()
```

Returns

A dictionary containing IGMP status information where:

Element	Description
`ena_igmp_snoop`	Whether IGMP snooping is enabled (Str); either `yes` (default) or `no`.

python_igmp_snoop_set()

Set the IGMP snooping status on the device

Syntax

`python_igmp_snoop_set(`*dict_igmp_snoop_status*`)`

where:

Variable	Description
dict_igmp_snoop_status	A dictionary that indicates whether IGMP snooping is enabled; contains *ena_igmp_snoop*.
ena_igmp_snoop	Indicates whether IGMP snooping is enabled (Str); either `yes` (default) or `no`.

Returns

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

class IgmpMcVlan

This class contains methods for getting and setting IGMP snooping status for VLANs.

python_igmp_snoop_all_if_get()

Get the IGMP snooping status for all VLANs

Syntax

```
python_igmp_snoop_all_if_get()
```

Returns

A list of dictionaries containing the IGMP snooping status for all VLANs:

Element	Description
`vid`	The VLAN ID; an integer from 1-3999.
`ena_igmp_snoop`	Whether IGMP snooping is enabled (Str); one of `yes`, `no`. Default value: `yes`.
`fast_leave`	Whether IGMP fast leave is enabled (Str); one of `yes`, `no`. Default value: `no`.
`query_interval`	IGMP query interval, in seconds; an integer from 1-18000. Default value: 125.
`version`	IGMP version (Str); one of `V1`, `V2`, `V3`.

python_igmp_snoop_if_get()

Get the IGMP snooping status for a specified VLAN

Syntax

`python_igmp_snoop_if_get(`*<vid>*`)`

where:

Variable	Description
vid	The VLAN ID; an integer from 1-3999.

Returns

A dictionary containing the IGMP snooping status for the specified VLAN:

Element	Description
`vid`	The VLAN ID; an integer from 1-3999.
`ena_igmp_snoop`	Whether IGMP snooping is enabled (Str); one of `yes`, `no`. Default value: `yes`.

Element	Description
`fast_leave`	Whether IGMP fast leave is enabled (Str); one of `yes`, `no`. Default value: `no`.
`query_interval`	IGMP query interval, in seconds; an integer from 1-18000. Default value: 125.
`version`	IGMP version (Str); one of `V1`, `V2`, `V3`.

python_igmp_snoop_all_if_get()

Get the IGMP snooping status for all VLANs

Syntax

```
python_igmp_snoop_all_if_get()
```

Returns

A list of dictionaries, each containing the IGMP snooping status:

Element	Description
`vid`	The VLAN ID; an integer from 1-3999.
`ena_igmp_snoop`	Whether IGMP snooping is enabled (Str); one of `yes`, `no`. Default value: `yes`.

python_igmp_snoop_vlan_set()

Set the IGMP snooping status for a specified VLAN

Syntax

`python_igmp_snoop_vlan_set(`*<dict_vlan_igmp_snoop_status>*`)`

where *dict_vlan_igmp_snoop_status* is a dictionary containing the following elements:

Element	Description
vlan_id	The VLAN ID; an integer from 1-3999.
ena_igmp_snoop	Whether IGMP snooping is enabled (Str); one of `yes`, `no`. Default value: `yes`.
fast_leave	Whether IGMP fast leave is enabled (Str); one of `yes`, `no`. Default value: `no`.
query_interval	IGMP query interval, in seconds; an integer from 1-18000. Default value: 125.
version	IGMP version (Str); one of `V1`, `V2`, `V3`.

where:

Variable	Description
dict_vlan_igmp_snoop_status	A dictionary containing the IGMP snoop status with the following parameters: • *vid* • *ena_igmp_snoop*
vid	The VLAN ID; an integer from 1-3999.
ena_igmp_snoop	Whether IGMP snooping is enabled (Str); either `yes` or `no`.

Returns

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

IP Module

The class and functions in this module manage 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()

Get the IP properties of a specified interface

Syntax

```
get_ipinfo([<if_name>])
```

where:

Variable	Description
if_name	(Optional) The interface name (Str); default value is `None`.

Returns

A dictionary containing the IP details:

Element	Description
`if_name`	IP interface name. **Note:** The interface must exist.
`switchport`	Whether or not the port is a switchport; one of `yes` (default), `no`.
`mtu`	The maximum transmission unit, in bytes; an integer from 64-9216. Default value: 1500.
`ip_addr`	IP address for the interface.
`ip_prefix_len`	IP address mask; a positive integer from 1-32.
`vrf_name`	The name of the VRF to which the interface belongs (if applicable).
`admin_state`	The admin status; one of `up`, `down`.

set_ip_addr()

Set the IP address of a specified interface

Syntax

`set_ip_addr(`*<if_name>*`,`*<ip_addr>*`,`*<secondary>*`)`

where:

Variable	Description
if_name	IP interface name. **Note:** The interface must exist.
ip_addr	IP address for the interface.
secondary	Secondary value for an interface (Int).

Returns

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

set_bridgeport()

Make the specified interface a bridge port

Syntax

`set_bridgeport(`*<if_name>*`)`

where:

Variable	Description
if_name	IP interface name. **Note:** The interface must exist.

Returns

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

unset_bridgeport()

Change the specified interface from a bridge port to a routed port

Syntax

`unset_bridgeport(<if_name>)`

where:

Variable	Description
if_name	IP interface name (Str). **Note:** The interface must exist.

Returns

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

set_if_flagup()

Set the interface flag to make it operational

Syntax

`set_if_flagup(<if_name>)`

where:

Variable	Description
if_name	IP interface name. **Note:** The interface must exist.

Returns

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

unset_if_flagup()

Unset the interface flag to make it non-operational

Syntax

`unset_if_flagup(`<*if_name*>`)`

where:

Variable	Description
if_name	IP interface name (Str). **Note:** The interface must exist.

Returns

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

Route Module

The class and functions in this module manage static routes. To use this module, in the Python file or in the Python interpreter, enter:

```
import routeApi
```

class Route

This class contains functions that manage static routes.

set_route()

Add 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:

Variable	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 (Str).
vrf_name	The VRF name (Str). **Note:** The named VRF must exist.
if_name	Interface name used for communication (Str). **Note:** The interface must exist.

Returns

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

delete_route()

Delete a static IPv4 route

Syntax

`delete_route(`*<ip_addr>*`,`*<ip_prefix_len>*`,`*<ip_gw>*`,`*<dist>*`,`*<tag>*`,`*<desc>*`,`*<vrf_name>*`,`*<if_name>*`)`

where:

Variable	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 (Str)
vrf_name	The VRF name (Str). **Note:** The named VRF must exist.
if_name	Interface name used for communication (Str). **Note:** The interface must exist.

Returns

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

get_route()

Get all IPv4 routes from the routing table

Syntax

`get_route(<vrf_name>,[<ip_dest>],[<ip_prefix_len>])`

where:

Variable	Description
vrf_name	The VRF name (Str). **Note:** The named VRF must exist.
ip_dest	(Optional) The destination IP address (Str); default value is `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:

Element	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 (Str).
`vrf_name`	The VRF name (Str).
`if_name`	Interface name used for communication (Str).

VRRP Module

The class and functions in this module manage 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 contains functions for managing Virtual Router Redundancy Protocol (VRRP).

get_vrrp()

Get properties of all VRRP Virtual Routers (VRs) for all interfaces or for the specified interface.

Syntax

`get_vrrp([`*<vr_id>*`], [`*<if_name>*`])`

where:

Variable	Description
vr_id	(Optional) The VRRP session Virtual Router (VR) ID; an integer from 1-255. Default value is `None`.
if_name	(Optional) Interface name used for communication (Str); default value is `None`. **Note:** The interface must exist.

Returns

A list of VRRP VR information:

Element	Description
`if_name`	The Ethernet interface name (Str).
`vr_id`	The VRRP session Virtual Router (VR) ID; an integer from 1-255. Default value is `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; one of `yes` (default) , `no`.
`prio`	The priority of the VR on the switch; an integer from 1-254. Default value: 100.

Element	Description
admin_state	Enable the VR (Str); one of up (default), down.
oper_state	The operation state of the VR (Str); one of master, backup, init.
track_if	The interface to track by this VR (Str). Default value: none. **Note:** If an interface is specified, it must exist.
accept_mode	Enables or disables the accept mode for this session (Str); one of yes (default), no.
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 (Str); one of yes, no (default).

get_vrrp_intf()

Get properties of all VRRP VRs of specified interfaces

Syntax

`get_vrrp_intf(`*<if_name>*`)`

where:

Variable	Description
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.

Returns

A list of VRRP properties:

Element	Description
if_name	The Ethernet interface name (Str).
vr_id	The VRRP session Virtual Router (VR) ID; an integer from 1-255. Default value is 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; one of yes (default) , no.

Element	Description
`prio`	The priority of the VR on the switch; an integer from 1-254. Default value: 100.
`admin_state`	Enable the VR (Str); one of `up` (default), `down`.
`oper_state`	The operation state of the VR (Str); one of `master`, `backup`, `init`.
`track_if`	The interface to track by this VR (Str). Default value: `none`. **Note:** If an interface is specified, it must exist.
`accept_mode`	Enables or disables the accept mode for this session (Str); one of `yes` (default), `no`.
`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 (Str); one of `yes`, `no` (default).

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:

Variable	Description
vr_id	The VRRP session Virtual Router (VR) ID; an integer from 1-255.
af_type	The Address Family type (Int); `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.

Returns

The `accept_mode` for the session (Str); one of `yes` (default), `no`.

get_vrrp_advt_interval()

Get the IGMP snooping status for a VLAN

Syntax

`get_vrrp_advt_interval(`*<vr_id>*`,`*<af_type>*`,`*<if_name>*`)`

where:

Variable	Description
vr_id	The VRRP session Virtual Router (VR) ID; an integer from 1-255.
af_type	The Address Family type (Int); `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.

`where:`

Variable	Description
vid	The VLAN ID (Int).

Returns

The advertisement interval:

Element	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()

Get whether a higher priority virtual router can preempt a lower priority master.

Syntax

`get_vrrp_preempt_mode(`*<vr_id>*`,`*<af_type>*`,`*<if_name>*`)`

where:

Variable	Description
vr_id	The VRRP session Virtual Router (VR) ID; an integer from 1-255.
af_type	The Address Family type (Int); `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	Ethernet interface name used for communication (Str). **Note:** The interface must exist.

Returns

Whether the preemption of a lower priority master is enabled:

Element	Description
`preempt`	Enable the preemption of a lower priority master; one of `yes` (default) , `no`.

get_vrrp_priority()

Get the priority to be used for the virtual router master election process.

Syntax

`get_vrrp_priority(`*<vr_id>*`,`*<af_type>*`,`*<if_name>*`)`

where:

Variable	Description
vr_id	The VRRP session Virtual Router (VR) ID; an integer from 1-255.
af_type	The Address Family type (Int); `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.

Returns

VRRP priority:

Element	Description
`prio`	The priority of the VR on the switch; an integer from 1-254. Default value: 100.

set_vrrp_accept_mode()

Set 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:

Variable	Description
vr_id	The VRRP session Virtual Router (VR) ID; an integer from 1-255.
af_type	The Address Family type (Int); `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.
accept_mode	Whether to enable Accept mode for this VRRP session (Str); one of `yes` (default), `no`.

Returns

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

set_vrrp_advt_interval()

Set the advertisement interval of a virtual router.

Syntax

`set_vrrp_advt_interval(`*<vr_id>*`,`*<af_type>*`,`*<if_name>*`,`*<interval>*`)`

where:

Variable	Description
vr_id	The VRRP session Virtual Router (VR) ID; an integer from 1-255.
af_type	The Address Family type (Int); `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.
interval	Advertisement interval in centi-seconds (Int); a multiple of 5 from 5-4095. Default value: `100` centi-seconds.

Returns

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

set_vrrp_preempt_mode()

Enable or disable the preempt mode for a session.

Syntax

`set_vrrp_preempt_mode(`*<vr_id>*`,`*<af_type>*`,`*<if_name>*`,`*<preempt>*`)`

where:

Variable	Description
vr_id	The VRRP session Virtual Router (VR) ID; an integer from 1-255.
af_type	The Address Family type (Int); `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.
preempt	Enable the preemption of a lower priority master; one of `yes` (default) , `no`.

Returns

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

set_vrrp_priority()

Enable the configuration of the priority of the VRRP router for a session.

Syntax

`set_vrrp_priority(`*<vr_id>*`,`*<af_type>*`,`*<if_name>*`,`*<prio>*`)`

where:

Variable	Description
vr_id	The VRRP session Virtual Router (VR) ID; an integer from 1-255.
af_type	The Address Family type (Int); `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (Str). **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()

Get the IGMP snooping status for a VLAN

Syntax

`set_vrrp_switch_back_delay(`*<vr_id>*`,`*<af_type>*`,`*<if_name>*`,`
<switch_back_delay>`)`

where:

Variable	Description
vr_id	The VRRP session Virtual Router (VR) ID; an integer from 1-255.
af_type	The Address Family type (Int); `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.
switch_back_ delay	The switch back delay interval; an integer from 1-500000 or 0 to disable (default).

Returns

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

set_vrrp_oper_primary_ipaddr()

Set the primary IP address of the VRRP virtual router.

Syntax

`set_vrrp_oper_primary_ipaddr(`*<vr_id>*`,`*<af_type>*`,`*<if_name>*`,`*<ipaddr>*`)`

where:

Variable	Description
vr_id	The VRRP session Virtual Router (VR) ID; an integer from 1-255.
af_type	The Address Family type (Int); `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.
ipaddr	The primary IP address (Str).

Returns

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

set_vrrp_monitored_circuit()

Get the IGMP snooping status for a VLAN

Syntax

`set_vrrp_monitored_circuit(`*<vr_id>*`,`*<af_type>*`,`*<if_name>*`,`*<track_if>*`)`

where:

Variable	Description
vr_id	The VRRP session Virtual Router (VR) ID; an integer from 1-255.
af_type	The Address Family type (Int); `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (Str). **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()

Delete a VRRP VR.

Syntax

`delete_vrrp_vr(`*<vr_id>*`,`*<af_type>*`,`*<if_name>*`)`

where:

Variable	Description
vr_id	The VRRP session Virtual Router (VR) ID; an integer from 1-255.
af_type	The Address Family type (Int); `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6).
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.

Returns

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

set_vrrp_vr()

Create a new VRRP session on the specified interface and allocate resources for the session.

Syntax

`set_vrrp_vr(`*<vr_id>*`,`*<if_name>*`)`

where:

Variable	Description
vr_id	The VRRP session Virtual Router (VR) ID; an integer from 1-255.
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.

Returns

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

ARP Module

This module contains a class with functions that manage 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()

Get all static ARP entries.

Syntax

```
get_all_static_arp_entry([<if_name>])
```

where:

Variable	Description
if_name	[Optional) The Ethernet interface name (Str). Default value: `none`. **Note:** If specified, the interface must exist.

Returns

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

Element	Description
`if_name`	The Ethernet interface name (Str).
`ip_addr`	The IP address (Str).
`mac_addr`	The MAC address in xxxx.xxxx,xxxx format (Str).

get_one_static_arp_entry()

Get one static ARP entry for the specified interface.

Syntax

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

where:

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

Returns

The static ARP entry, with the following parameters:

Element	Description
`if_name`	The Ethernet interface name (Str).
`ip_addr`	The IP address (Str).
`mac_addr`	The MAC address in xxxx.xxxx,xxxx format (Str).

set_ip_arp()

Create a static proxy ARP entry.

Syntax

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

where

Variable	Description
ip_addr	The IP address (Str).
mac_addr	The MAC address in xxxx.xxxx,xxxx format (Str).
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.

Returns

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

delete_ip_arp()

Delete an ARP entry.

Syntax

`delete_ip_arp(<ip_addr>,<if_name>)`

where

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

Returns

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

get_arp_sys_pro()

Get the global ARP properties of the system.

Syntax

`get_arp_sys_pro()`

Returns

The global ARP entry age out time:

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

set_arp_sys_pro()

Set the global ARP properties of the system.

Syntax

`set_arp_sys_pro(<ageout_time>)`

where:

Variable	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()

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

Syntax

`get_arp_sys_interfaces([<if_name>])`

where:

Variable	Description
if_name	(Optional) The Ethernet interface name (Str). Default value: `none`. **Note:** If specified, the interface must exist.

Returns

ARP properties for all interfaces or for the specified interface:

Element	Description
`if_name`	The Ethernet interface name (Str).
`ageout_time`	The global ARP entry age out time, in seconds; an integer from 60-28800. Default value: `1500`.

set_arp_sys_pro_interface()

Set the ARP properties of the specified interface.

Syntax

`set_arp_sys_pro_interface(>if_name>,<ageout_time>)`

where:

Variable	Description
if_name	The Ethernet interface name (Str). Default value: `none`. **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`).

VRF Module

The class and function in this module manage 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 a function for managing VRF.

get_vrf_entry()

Get all VRF details.

Syntax

`get_vrf_entry([<vrf_name>])`

where:

Variable	Description
vrf_name	The name of the virtual router (Str). Default value: `none`. **Note:** The VR be either a management or default VR.

Returns

The following VRF details:

Element	Description
`vrf_name`	The name of the virtual router (Str).
`interfaces`	The interface (Str).

Platform Module

The classes in this module contain functions that get and provide 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 provide a physical port's configuration.

get_interface()

Get the properties of one interface.

Syntax

`get_interface([<if_name>])`

where:

Variable	Description
if_name	(Optional) The Ethernet interface name (Str). Default value: `none`. **Note:** If specified, the interface must exist.

Returns

A dictionary containing lists of interface properties:

Element	Description
`duplex`	The communication method of the interface (Str); one of `auto`, `full`, `half`.
`if_name`	The interface name (Str).
`mtu`	The maximum transmission unit, in bytes; a positive integer from 64-9216. Default value: `1500`.
`admin_state`	The admin status (Str); one of `up`, `down`.
`mac_addr`	The MAC address in xxxx.xxxx,xxxx format (Str).
`speed`	The communication speed of the interface (Str); one of: • `auto` (auto negotiate) • `10` (10Mb/s) • `100` (100Mb/s) • `1000` (1Gb/s) • `10000` (10Gb/s) • `40000` (40Gb/s).

get_mac()

Get the MAC address of a physical port.

Syntax

`get_mac(`*<if_name>*`)`

where:

Variable	Description
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.

Returns

The MAC address:

Element	Description
`mac_addr`	The MAC address in xxxx.xxxx,xxxx format (Str).

set_mac()

Set the MAC address of the specified interface.

Syntax

`set_mac(`*if_name, mac_address*`)`

where:

Variable	Description
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.
mac_addr	The MAC address in xxxx.xxxx,xxxx format (Str).

Returns

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

is_enabled()

Check whether the port is enabled

Syntax

`is_enabled(<if_name>)`

where:

Variable	Description
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.

Returns

Enabled status:

Element	Description
`is_enabled`	Port enabled status (Str); one of `up`, `down`

set_enabled()

Enable or disable the specified port.

Syntax

`set_enabled(<if_name>,<flag>)`

where:

Variable	Description
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.
flag	Snooping status (Str); one of `up`, `down`.

Returns

The maximum transmission unit:

Element	Description
`mtu`	The maximum transmission unit; a positive integer from `64-9216`. Default value: `1500`.

set_mtu()

Set the Maximum Transmission Unit (MTU) for the port.

Syntax

`set_mtu(<if_name>,<mtu>)`

where:

Variable	Description
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.
mtu	The maximum transmission unit; a positive integer from `64-9216`. Default value: `1500`.

Returns

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

get_port_speed()

Get the speed of the specified port.

Syntax

`get_port_speed(<if_name>)`

where:

Variable	Description
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.

Returns

The port speed:

Element	Description
`speed`	The communication speed of the interface (Str); one of: • `auto` (auto negotiate) • `10` (10Mb/s) • `100` (100Mb/s) • `1000` (1Gb/s) • `10000` (10Gb/s) • `40000` (40Gb/s).

set_port_speed()

Set the speed of the specified port.

Syntax

`set_port_speed(`*<if_name>*`,`*<speed>*`)`

where:

Variable	Description
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.
speed	The communication speed of the interface (Str); one of: • `auto` (auto negotiate) • `10` (10Mb/s) • `100` (100Mb/s) • `1000` (1Gb/s) • `10000` (10Gb/s) • `40000` (40Gb/s).

Returns

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

get_duplex()

Get the duplex for the port.

Syntax

`get_duplex(`*<if_name>*`)`

where:

Variable	Description
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.

Returns

The duplex of the interface:

Element	Description
`duplex`	The communication method of the interface (Str); one of `auto`, `full`, `half`.

set_duplex()

Set the duplex for the port.

Syntax

`get_duplex(`*<if_name>*`,`*<duplex>*`)`

where:

Variable	Description
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.
duplex	The communication method of the interface (Str); one of `auto`, `full`, `half`.

Returns

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

class PortStatistics

This class contains a function that gets port statistics.

get_stats()

Get the port statistics for the specified interface.

Syntax

`get_stats(`*<if_name>*`)`

where:

Variable	Description
if_name	The Ethernet interface name (Str). **Note:** The interface must exist.

Returns

The following packet statistics:

Element	Description
`good_pkts_sent`	Number of sent packets (Int).
`in_un_pkts`	Number of received unicast packets (Int).
`bad_crc`	Number of bad CRCs (Int).
`brdc_pkts_rcv`	Number of received broadcast packets (Int).
`mc_pkts_sent`	Number of sent multicast packets (Int).

Element	Description
undersize_pkts	Number of undersise packets (Int).
mc_pkts_rcv	Number of received multicast packets (Int).
in_discards	Number of discarded in packets (Int).
good_octets_rcv	Number of received octets (Int).
oversize_pkts	Number of oversize packets (Int).
brdc_pkts_sent	Number of sent broadcast packets (Int).
good_octets_sent	Number of sent octets (Int).
out_uc_pkts	Number of sent unicast packets (Int).
good_pkts_rcv	Number of received packets (Int).

BGP Module

The classes in this module contain functions that get and provide Border Gateway Protocol (BGP) information. To use this module, in the Python file or in the Python interpreter, enter:

```
import bgpApi
```

class BGP()

The functions in this class get and set BGP configurations.

python_bgp_get_global_statistics()

Get BGP global statistics.

Syntax

`python_bgp_get_global_statistics(<vrf_name>)`

where:

Variable	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

A dictionary containing BGP global statistics:

Element	Description
`vrf_name`	Virtual Routing and Forwarding name (Str).
`stats`	A dictionary containing global statistics.
`in_msgs`	Received message number; a positive integer.
`out_msgs`	Send messge 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.
`keepalive_in`	Keepalive input count; a positive integer.
`keepalive_out`	Keepalive output count; a positive integer.

Element	Description
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()

Get BGP global statistics.

Syntax

```
python_bgp_clear_global_statistics([<vrf_name>])
```

where:

Variable	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "default," "all". Default value: default.

Returns

Boolean (True on success, otherwise False).

python_show_bgp_peer_adj_routes()

Show BGP neighbor received and advertised routes.

Syntax

```
python_show_bgp_peer_adj_routes(<in>,<neighbor_ip>,[<vrf_name>],
[<af_name>],[<subaf_name>])
```

where:

Element	Description
in	One of the following: • 1 - adjacent routes • 0 - advertised adjacent routes
neighbor_ip	Neighbor IP address; a valid IPv4 or IPv6 address.
vrf_name	(Optional) Address family name; one of ipv4 or ipv6m. Default value: ipv4.

Element	Description
af_name	(Optional) VRF name; one of the VRF name, `default`, `all`. Default value: `default`.
subaf_name	(Optional) Subaddress family name; one of `unicast`, `multicast`. Default value: `unicast`.

Returns

A dictionary containing BGP neighbor received and advertised routes:

Element	Description
`origin`	Route origin attribute; one of: • `i` - IGP • `e` - EGP • `?` - incomplete
`network`	Route destination IP address; a valid IPv4 or IPv6 address.
`mask_len`	Route mask length; an integer from 0-32.
`weight`	Route weight attribute; an integer from 0-65535.
`metric`	Route Multi-Exit Discriminator attribute; an integer from 0~4294967295.
`nexthop`	Route next hop; a valid IP address.
`aspath4B`	Route 4B AS path; an AS path VTY string.
`status`	Router status; one of: • `s` - suppressed • `d` - damped • `h` - history • `*` - valid • `>` - best • `i` - internal
`local_pref`	Route local preference attribute; an integer from 0-4294967295.
`aspath`	Route AS path attribute; an AS path VTY string.

python_bgp_get_status()

Show whether BGP is enabled or disabled globally.

Syntax

`python_bgp_get_status([<vrf_name>])`

where:

Element	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

The BGP global status: one of `enable`, `disable`.

python_bgp_get_router_id()

Get the BGP router ID.

Syntax

`python_bgp_get_router_id([<vrf_name>])`

where:

Element	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

The BGP router ID (Str); a valid IP address.

python_bgp_get_as_number()

Get 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()

Get the BGP hold down interval.

Syntax

`python_bgp_get_hold_down_timer([<vrf_name>])`

where:

Element	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

The hold down timer value, in MS; an integer from 1-3600. Default value: 180.

python_bgp_get_keep_alive_timer()

Get the BGP keep alive interval.

Syntax

`python_bgp_get_keep_alive_timer([<vrf_name>])`

where:

Element	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

The keep alive timer value, in MS; an integer from 1-3600. Default value: 60.

python_bgp_get_enforce_first_as()

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

Syntax

`python_bgp_get_enforce_first_as([<vrf_name>])`

where:

Element	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

The BGP global enforce-first-AS status: one of `enable`, `disable`.

python_bgp_get_fast_external_failover()

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

Syntax

`python_bgp_get_fast_external_failover([<vrf_name>])`

where:

Element	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

The BGP global fast-external-failover status: one of `enable`, `disable`.

python_bgp_get_log_neighbor_changes()

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

Syntax

`python_bgp_get_log_neighbor_changes([<vrf_name>])`

where:

Element	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

The BGP global log-neighbor-changes status: one of `enable`, `disable`.

python_bgp_get_as_local_cnt()

Get the BGP Autonomous System (AS) local count.

Syntax

`python_bgp_get_as_local_cnt([<vrf_name>])`

where:

Element	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

The BGP local AS count: an integer from 0-64. Default value: 0.

python_bgp_get_maxas_limit()

Get the BGP maximum AS limit.

Syntax

`python_bgp_get_maxas_limit([<vrf_name>])`

where:

Element	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

The maximum number of Autonomous Systems; an integer from 0-2000. Default value: 0.

python_bgp_get_synchronization()

Get whether BGP global synchronization is enabled or disabled.

Syntax

`python_bgp_get_synchronization([<vrf_name>])`

where:

Element	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

The BGP global synchronization status: one of `enable`, `disable`.

python_bgp_get_bestpath_cfg()

Get BGP best path configuration.

Syntax

`python_bgp_get_synchronization([<vrf_name>])`

where:

Element	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

A dictionary containing the best path configuration:

Element	Description
`vrf_name`	VRF name; one of the VRF name, `default`, `all`. Default value: `default`.
`always-compare-med`	Allow comparing MED from different neighbors; one of `enable`, `disable`.
`as-path-ignore`	Ignore as-path length in selecting a route; one of `enable`, `disable`.
`as-path multipath-relax`	Relax AS-Path restriction when choosing multipaths; one of `enable`, `disable`.
`compare-confed-aspath`	Allow comparing confederation AS path length; one of `enable`, `disable`.
`compare-routerid`	Compare router IDs for identical EBGP paths; one of `enable`, `disable`.
`dont-compare-originator-id`	Don't compare originator IDs for BGP; one of `enable`, `disable`.
`med-confed`	Compare MED among confederation paths; one of `enable`, `disable`.
`med-missing-as-worst`	Treat missing MED as the least preferred one; one of `enable`, `disable`.
`med-non-deterministic`	Best MED path among paths not selected from same AS; one of `enable`, `disable`.
`med-remove-recv-med`	Whether to remove received MED attribute; one of `enable`, `disable`.
`med-remove-send-med`	Whether to remove send MED attribute; one of enable, disable.
`tie-break-on-age`	Whether to prefer the old route when `compare-route-id` is not set; one of `enable`, `disable`.

python_bgp_get_confed_id()

Get the BGP confederation identifier.

Syntax

`python_bgp_get_confed_id([<vrf_name>])`

where:

Element	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

The BGP routing domain confederation AS: an integer from 0-65535.

python_bgp_get_confederation_peers()

Get the BGP confederation peers.

Syntax

`python_bgp_get_confederation_peers([<vrf_name>])`

where:

Element	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

The number of peer autonomous systems in the BGP confederation: an integer from 1-65535.

python_bgp_get_graceful_helper_status()

Get whether BGP graceful helper is enabled or disabled.

Syntax

`python_bgp_get_graceful_helper_status([<vrf_name>])`

where:

Element	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

The BGP graceful helper status: one of `enable`, `disable`.

python_bgp_get_graceful_stalepath_time()

Get the BGP stale path time.

Syntax

`python_bgp_get_graceful_stalepath_time([<vrf_name>])`

where:

Element	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of 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.

python_bgp_get_cluster_id()

Get the BGP route reflector cluster ID.

Syntax

`python_bgp_get_cluster_id([<vrf_name>])`

where:

Element	Description
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

The route reflector cluster ID; a valid IP address.

python_show_ip_bgp()

Get BGP Routing Information Base (RIB) information.

Syntax

`python_show_ip_bgp([<af_name>],[<vrf_name>])`

where:

Element	Description
af_name	(Optional) Address family name; one of `ipv4`, `ipv6`. Default value; both.
vrf_name	(Optional) Virtual Routing and Forwarding name; one of the VRF name, "`default`," "`all`". Default value: `default`.

Returns

A dictionary containing RIB information:

Element	Description
`status`	Router status code; one of: • `s` - suppressed • `d` - damped • `h` - history • `*` - valid • `>` - best • `i` – internal
`network`	Route destination IP address; a valid IPv4 or IPv6 address.
`nextHopGlobal`	Route nexthop IPv6 address. Not used for IPv4.

Element	Description
`nextHopLocal`	Route nexthop; a valid IPv4 or IPv6 address.
`weight`	Route weight attribute; an integer from 0-65535.
`pathInfo`	Route path information; a valid AS path VTY string.
`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.
`aspath`	Route AS path attribute; a valid AS path VTY string.
`aspath4B`	Route 4B AS path; a valid AS path VTY string.
`origin`	Route origin attribute; one of the following: • `i` - IGP • `e` - EGP • `?` - incomplete

python_show_ip_bgp_network()

Get detailed information about a BGP route.

Syntax

`python_show_ip_bgp_network(`*<route>*`,`*<network_mask>*`,`
`[`*<af_name>*`],[`*<vrf_name>*`])`

where:

Element	Description
route	Route; a valid IPv4 or IPv6 address.
network_mask	Network mask: • IPv4: An integer from 0-32. • IPv6: An integer from 0-128.
af_name	(Optional) Address family name; one of `ipv4`, `ipv6`. Default value: both.
vrf_name	(Optional) VRF name; one of the VRF name, `default`, `all`. Default value: `default`.

Returns

A dictionary containing BGP route information:

Element	Description
`table entry for`	Route IP address/mask; a valid IP address and net mask.
`paths`	Dictionary; marks the beginning of the path table for the specified route entry.
`as path str`	Route path information; a valid 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; `Yes`. **Note:** This value only appears if it has been set.
`suppressed (damp)`	Suppressed due to dampening; `Yes`. **Note:** This value only appears if it has been set.
`history entry`	History entry; `Yes`. **Note:** This value only appears if it has been set.
`nexthop address`	Route nexthop; a valid IPv4 or IPv6 address.
`peer`	Peer address.
`inaccessible`	Whether the RIB is can be accessed; one of `yes`, `no`.
`igpmetric`	IGP metric value; No. **Note:** This value only appears if it is `No`.
`from peer`	Whether the from peer address can be accessed, `No`. **Note:** This value only appears if it is `No`.
`orig id`	Whether the originator ID can be accessed; No.
`next-hop local ip`	Whether the next-hop IP address can be accessed; a valid IP address or `No`. **Note:** The value `No` only appears if it is inaccessible.
`metric`	Metric; one of the metric value, `removed`.
`local pref`	Local preference value; only appears if set.
`weight`	Route weight attribute; an integer from 0-65535. **Note:** This value only appears if it is set.
`label`	Label; only appears if set.

Element	Description
valid	Whether the path is valid; Yes. **Note:** This value only appears if the path is valid.
stale	Whether the state is stale; Yes. **Note:** This value only appears if the state is stale.
multipath-candidate	Whether this is a multipath candidate; one of yes, no.
installed	Whether installed; one of yes, no.
synchronized	Whether synchronized; one of yes, no.
atomic aggregate	Whether this is an atomic aggregate; Yes. **Note:** This value only appears if it is Yes.
best	Whether this is the best path; one of yes, no.
community	Community string.
Extended community	Extended community string.
originator	Originator ID.
cluster id	Cluster ID.
reuse info	Reuse information.
last update	Last update time.
best is no.	Which path number is best; the maximum number of paths for this destination.
advertised to any peer	Whether the route is advertised to any peers; one of yes, no.
advertised to EBGP peer	Whether the route is advertised to an EBGP peer; one of yes, no.
advertised outside local AS	Whether the route is advertised outside the local AS; one of yes, no.
advertisements suppressed by an agregate	Whether advertisements are suppressed by an aggregate; one of yes, no.
advertised to non peer-group peers	IP address advertised to non peer-group peers.
advertised to peer-groups	IP address addvertised to peer groups.
not advertised	Not advertised to any peer. **Note:** This value only appears if true.

python_show_ip_bgp_summary()

Get BGP summary information.

Syntax

`python_show_ip_bgp_summary([`*<af_name>*`],[`*<vrf_name>*`], [`*<subaf_name>*`])`

where:

Element	Description
af_name	(Optional) Address family name; one of `ipv4`, `ipv6`. Default value: `ipv4`.
vrf_name	(Optional) VRF name; one of the VRF name, `default`, `all`. Default value: `default`.
subaf_name	(Optional) Subsequent Address Family Identifier name; `unicast`. Default value: `unicast`.

Returns

A dictionary containing BGP route information:

Element	Description
`router id`	Router ID; a valid IPv4 or IPv6 address.
`peer`	Peer address; a valid IPv4 or IPv6 address.
`peer version`	Peer version.
`peer AS`	Peer AS.
`open in`	Number of received open messages.
`update in`	Number of received updates.
`keepalive in`	Number of received keepalives.
`refresh in`	Number of received route refresh.
`dynamic cap in`	Dynamic capabilities imput count.
`open out`	Number of sent open messages.
`update out`	Number of sent updates.
`keepalive out`	Number of sent keepalive messages.
`refresh out`	Number of sent route refresh messages.
`dynamic cap out`	Dynamic capabilities output count.

python_show_ip_bgp_neighbor()

Get BGP neighbor details.

Syntax

`python_show_ip_bgp_neighbor([`*<nbr-ip>*`],[`*<vrf_name>*`])`

where:

Element	Description
nbr-ip	(Optional) Neighbor IP address; 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; one of the VRF name, `default`, `all`. Default value: `default`.

Returns

A dictionary containing BGP route information:

Element	Description
`neighbor`	Neighbor address.
`vrfname`	VRF name.
`remote AS`	AS number.
`local AS`	Local AS number.
`address family`	Address family.
`table version`	Table version.
`neighbor version`	Neighbor version.
`index val`	Neighbor index value.
`index offset`	Index offset.
`index mask`	Index mask.
`link type`	Link type; one of `internal`, `external`.
`version`	Version.
`description`	Description.
`remote router-ID`	Remote router ID.
`admin`	Admin state.
`ifbound`	Whether the interface is bound; one of No interface binding, Interface bound.

Element	Description
state	Neighbor state.
dyncap_adv	Dynamic capability advertised, only if advertised.
dyncap_rec	Dynamic capability received, only if received.
refresh_adv	Refresh capability advertised, only if advertised.
refresh_new_rec	Refresh New received, only if received.
refresh_old_rec	Refresh Old received, only if received.
ext_asn_adv	Extended ASN capability advertised.
ext_asn_rec	Extended ASN capability received.
afc_adv	Address family unicast sent.
afc_recv	Address family unicast received.
afc_VPN_adv	Address family VPN sent.
afc_VPN_recv	Address family VPN received.
afc_mcast_adv	Address family multicast sent.
afc_mcast_recv	Address family multicast received.
uptime	Uptime.
peer-group name	Peer IP address.
holdtime	Holdtime.
keepalive	Keepalive time.
conf holdtime	Configured holdtime.
conf keepalive	Configured keepalive time.
recvMsg	Number of received messages.
recvNotf	Number of received notifications.
recvQueue	Received messages queue count.
sentMsg	Number of sent messages.
sentNotf	Number of sent notifications.
sentQueue	Sent messages queue count.
refresh_in	Number of route refresh messages received.
refresh_out	Number of route refresh messages sent.

Element	Description
routeadv	Number of router advertisements.
update_if	Update interface.
update_source	Update source address.
established	Established count.
dropped	Dropped count.
prefix overflow	Whether there is a prefix overflow; one of yes, no.
ttl	Time to live value.
local address	Local neighbor IP address.
local port	Local port number.
remote address	Remote peer IP address.
remote port	Remote port number.
nextHop Address	Next-hop address.
nextHopLocalV6	Next-hop address (link local).
nextHop GlobalV6	Next-hop address (global).
shared network	Shared network.
next conn retry	Number of retries.
err notif	Whether there was an error notification; one of sent, received.
last_reset_ time	Last reset time.
err code	Code string.
err subcode	Subcode string.
rmap_name	Default originating route map.

python_bgp_get_af_distance_config()

Get BGP distance information.

Syntax

`python_bgp_get_af_distance_config(<af_name>,<saf_name>, [<vrf_name>])`

where:

Element	Description
af_name	Address family name; one of `ipv4`, `ipv6`.
saf_name	Subsequent Address Family Identifier name; `unicast`.
vrf_name	(Optional) VRF name; one of the VRF name, `default`, `all`. Default value: `default`.

Returns

A dictionary containing BGP distance information:

Element	Description
`vrf_name`	VRF name; one of 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()

Get BGP global configuration information.

Syntax

`python_bgp_get_af_global_config(<af_name>,<saf_name>, [<vrf_name>])`

where:

Element	Description
af_name	Address family name; one of `ipv4`, `ipv6`.
saf_name	Subsequent Address Family Identifier name; `unicast`.
vrf_name	(Optional) VRF name; one of the VRF name, `default`, `all`. Default value: `default`.

Returns

A dictionary containing BGP global configuration information:

Element	Description
`vrf_name`	VRF name; one of the VRF name, `default`, `all`. Default value: `default`.
`cc_reflection`	Client-to-client reflect; one of `enable`, `disable`.
`synchronization`	Perform IGP synchronization; one of `enable`, `disable`.
`network_ synchronization`	Perform IGP synchronization on network routes; one of `enable`, `disable`.

python_bgp_get_af_maximum_paths_config()

Get BGP multipath ECMP number configuration information.

Syntax

`python_bgp_get_af_maximum_paths_config(`*<af_name>*`,`*<saf_name>*`, [`*<vrf_name>*`])`

where:

Element	Description
af_name	Address family name; one of `ipv4`, `ipv6`.
saf_name	Subsequent Address Family Identifier name; `unicast`.
vrf_name	(Optional) VRF name; one of the VRF name, `default`, `all`. Default value: `default`.

Returns

A dictionary containing BGP global configuration information:

Element	Description
`vrf_name`	VRF name; one of 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()

Get the BGP nexthop trigger-delay configuration.

Syntax

`python_bgp_get_af_nexthop_trigger_delay_config(`*<af_name>*`,` *<saf_name>*`)`

where:

Element	Description
af_name	Address family name; one of `ipv4`, `ipv6`.
saf_name	Subsequent Address Family Identifier name; `unicast`. Default value: `unicast`.

Returns

A dictionary containing BGP nexthop trigger-delay configuration information:

Element	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()

Get the BGP aggregate configuration.

Syntax

`python_bgp_get_af_aggregate_config("`*<af_name>*`","`*<saf_name>*`,"` `["`*<vrf_name>*`"])`

where:

Element	Description
af_name	Address family name; one of `ipv4`, `ipv6`. Default value; both.
saf_name	Subsequent Address Family Identifier name; `unicast`.
vrf_name	(Optional) VRF name; one of the VRF name, `default`. Default value: `default`.

Returns

A dictionary containing BGP aggregate information:

Element	Description
`vrf_name`	VRF name; one of the VRF name, `default`. Default value: `default`.
`prefix`	Aggregate prefix; an IP address in one of the following forms: • A.B.C.D/M • X:X::X:X/M.
`type`	Aggregate type; one of the following: • `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()

Get the BGP dampening configuration.

Syntax

`python_bgp_get_af_dampening_config("<af_name>","<saf_name>","["<vrf_name>"])`

`where:`

Element	Description
af_name	Address family name; one of `ipv4`, `ipv6`.
saf_name	Subsequent Address Family Identifier name; `unicast`.
vrf_name	(Optional) VRF name; one of the VRF name, `default`, `all`. Default value: `default`.

Returns

A dictionary containing BGP dampening information:

Element	Description
`vrf_name`	VRF name; one of the VRF name, `default`, `all`. Default value: `default`.

Element	Description
`prefix`	Aggregate prefix; an IP address in one of the following forms: • A.B.C.D/M • X:X::X:X/M.
`type`	Aggregate type; one of the following: • `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_network_config()

Get the BGP network configuration.

Syntax

`python_bgp_get_af_network_config(`*<af_name>*`,`*<saf_name>*`, [`*<vrf_name>*`])`

where:

Element	Description
af_name	Address family name; one of `ipv4`, `ipv6`.
saf_name	Subsequent Address Family Identifier name; `unicast`.
vrf_name	(Optional) VRF name; one of the VRF name, `default`, `all`. Default value: `default`.

Returns

A dictionary containing BGP network information:

Element	Description
`vrf_name`	VRF name; one of the VRF name, `default`, `all`. Default value: `default`.
`prefix`	Aggregate prefix; an IP address in one of the following forms: • A.B.C.D/M • X:X::X:X/M
`backdoor`	Whether a BGP backdoor route is specified; one of `enable`, `disable`.
`rmap_name`	Route map name; a string up to 63 characters long.

python_bgp_get_af_redistribute_config()

Get the BGP redistribute configuration.

Syntax

`python_bgp_get_af_redistribute_config(`*<af_name>*`,`*<saf_name>*`, [`*<vrf_name>*`])`

where:

Element	Description
af_name	Address family name; one of `ipv4`, `ipv6`.
saf_name	Subsequent Address Family Identifier name; `unicast`.
vrf_name	(Optional) VRF name; one of the VRF name, `default`, `all`. Default value: `default`.

Returns

A dictionary containing BGP redistribute configuration information:

Element	Description
`vrf_name`	VRF name; one of the VRF name, `default`, `all`. Default value: `default`.
`redist_direct`	Whether redistribute direct is enabled; one of `enable`, `disable`.
`direct_rmap_name`	Route map name for redistribute direct; a string up to 63 characters long.
`redist_ospf`	Whether redistribute OSPF is enabled; one of `enable`, `disable`.
`ospf_rmap_name`	Route map name for redistribute OSPF; a string up to 63 characters long.
`redist_static`	Whether redistribute static is enabled; one of `enable`, `disable`.
`static_rmap_name`	Route map name for redistribute static; a string up to 63 characters long.

python_show_ip_bgp_neighbor_stats()

Get BGP neighbor details.

Syntax

`python_show_ip_bgp_neighbor_stats(`*<nbr-ip>*`,`*<item>*`,[`*<vrf_name>*`])`

where:

Element	Description
nbr-ip	IP address; one or more valid IPv4 or IPv6 addresses.
item	The type of statistics; one or more of `keepalive`, `notification`, `open`, `update`, `recv_msgs`, `sent_msgs`. Default values: show all items.
vrf_name	(Optional) VRF name; one of the VRF name, `default`, `all`. Default value: `default`.

Returns

A dictionary containing BGP neighbor detail information:

Element	Description
`statistic type`	Statistic type; one or more of `keepalive`, `notification`, `open`, `update`, `recv_msgs`, `sent_msgs`.
`received`	Number of received messages of the specified type or types.
`sent`	Number of sent messages of the specified type or types.

DHCP Module

The classes in this module contain functions that get and provide Dynamic Host Configuration Protocol (DHCP) information. To use this module, in the Python file or in the Python interpreter, enter:

```
import dhcpApi
```

class DHCP()

The functions in this class get and set DHCP configurations.

get_dhcp_feature()

Determine whether the DHCP client is enabled or disabled.

Syntax

```
get_dhcp_feature()
```

Returns

Whether the DHCP client is enabled or disabled:

Element	Description
`ena_dhcp_feature`	Whether the DHCP client is enabled (Str); one of `yes`, `no`. Default value: `yes`.

set_dhcp_feature()

Enable the DHCP client.

Syntax

```
set_dhcp_feature()
```

Returns

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

unset_dhcp_feature()

Disable the DHCP client.

Syntax

```
unset_dhcp_feature()
```

Returns

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

get_dhcpinfo()

Get the DHCP client configuration.

Syntax

`get_dhcpinfo([<if_name>])`

where:

Variable	Description
if_name	(Optional) The interface port name (Str).

Returns

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

Element	Description
`if_name`	Interface name (Str).
`ena_v4_client`	Whether or not the DHCPv4 client is enabled on the interface; one of `yes`, `no`. Default value: `no`.
`ena_v6_client`	Whether or not the DHCPv6 client is enabled on the interface; one of `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; one of `yes`, `no`. Default value: `no`.
`class_id`	The name of the vendor class-identifier; a string up to 64 characters long.

set_dhcp_client()

Enable the DHCP client on the specified interface.

Syntax

`set_dhcp_client(<if_name>)`

where:

Variable	Description
if_name	The interface port name (Str).

Returns

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

unset_dhcp_client()

Disable the DHCP client on the specified interface.

Syntax

`unset_dhcp_client(<if_name>)`

where:

Variable	Description
if_name	The interface port name (Str).

Returns

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

set_dhcpv6_client()

Enable the DHCPv6 client on the specified interface.

Syntax

`set_dhcpv6_client(<if_name>)`

where:

Variable	Description
if_name	The interface port name (Str).

Returns

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

unset_dhcpv6_client()

Disable the DHCPv6 client on the specified interface.

Syntax

`unset_dhcpv6_client(`<*if_name*>`)`

where:

Variable	Description
if_name	The interface port name (Str).

Returns

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

set_dhcp_option_hostname()

Enable the DHCP hostname option on the specified interface.

Syntax

`set_dhcp_option_hostname(`<*if_name*>`)`

where:

Variable	Description
if_name	The interface port name (Str).

Returns

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

unset_dhcp_option_hostname()

Disable the DHCP hostname option on the specified interface.

Syntax

`unset_dhcp_option_hostname(`<*if_name*>`)`

where:

Variable	Description
if_name	The interface port name (Str).

Returns

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

set_dhcp_option_ntp_server()

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

Syntax

`set_dhcp_option_ntp_server(<if_name>)`

where:

Variable	Description
if_name	The interface port name (Str).

Returns

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

unset_dhcp_option_ntp_server()

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

Syntax

`unset_dhcp_option_ntp_server(<if_name>)`

where:

Variable	Description
if_name	The interface port name (Str).

Returns

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

set_dhcp_option_log_server()

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

Syntax

`set_dhcp_option_log_server(<if_name>)`

where:

Variable	Description
if_name	The interface port name (Str).

Returns

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

unset_dhcp_option_log_server()

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

Syntax

`unset_dhcp_option_log_server(<if_name>)`

where:

Variable	Description
if_name	The interface port name (Str).

Returns

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

set_dhcp_class_id()

Set the specified vendor class ID on the specified interface.

Syntax

`set_dhcp_class_id(<if_name>,<class_id>)`

where:

Variable	Description
if_name	The interface port name (Str).
class_id	The vendor class ID (Str); up to 64 characters long.

Returns

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

unset_dhcp_class_id()

Remove the vendor class ID from the specified interface.

Syntax

`unset_dhcp_class_id(<if_name>)`

where:

Variable	Description
if_name	The interface port name (Str).

Returns

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

class DHCP_Relay()

The functions in this class get and set DHCP relay configurations.

get_relay_serv()

Determine 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:

Element	Description
`ena_v4_relay`	Whether DHCPv4 relay is enabled (Str); one of `yes`, `no`. Default value: `no`.
`ena_v6_relay`	Whether DHCPv6 relay is enabled (Str); one of `yes`, `no`. Default value: `no`.

set_dhcpv4_relay()

Enable DHCPv4 relay service.

Syntax

```
set_dhcpv4_relay()
```

Returns

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

unset_dhcpv4_relay()

Disable DHCPv4 relay service.

Syntax

```
unset_dhcpv4_relay()
```

Returns

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

set_dhcpv6_relay()

Enable DHCPv6 relay service.

Syntax

```
set_dhcpv6_relay()
```

Returns

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

unset_dhcpv6_relay()

Disable DHCPv6 relay service.

Syntax

```
unset_dhcpv6_relay()
```

Returns

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

get_interface_relay_address()

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

Syntax

`get_interface_relay_address([<`*if_name*`>])`

where:

Variable	Description
if_name	(Optional) The interface port name (Str).

Returns

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

Element	Description
`if_name`	(Optional) The interface port name (Str).
`dhcpv4_relay`	A dictionary containing DHCPv4 relay addresses.
`v4_relay_address`	DHCPv4 relay server address (Str); a valid IPv4 address.
`dhcpv6_relay`	A dictionary containing DHCPv6 relay addresses.

Element	Description
`v6_relay_address`	DHCPv6 relay server address (Str); a valid IPv6 address.
`v6_relay_out_if`	DHCPv6 outgoing interface (Str).

set_relay4_address()

Set a DHCPv4 relay address for the specified interface.

Syntax

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

where:

Variable	Description
if_name	The interface port name (Str).
ip_addr	The IP address (Str); a valid IPv4 address.

Returns

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

unset_relay4_address()

Remove a DHCPv4 relay address from the specified interface.

Syntax

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

where:

Variable	Description
if_name	The interface port name (Str).
ip_addr	The IP address (Str); a valid IPv4 address.

Returns

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

set_relay6_address()

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

Syntax

`set_relay6_address(`*<if_name>*`,`*<ipv6_addr>*`,[`*<out_if_name>*`])`

where:

Variable	Description
if_name	The interface port name (Str).
ipv6_addr	The IP address (Str); a valid IPv6 address.
out_if_name	(Optional) Relay outoing interface (Str).

Returns

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

unset_relay6_address()

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

Syntax

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

where:

Variable	Description
if_name	The interface port name (Str).
ipv6_addr	The IP address (Str); a valid IPv6 address.
out_if_name	(Optional) Relay outoing interface (Str).

Returns

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

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 IBM ServerProven website to make sure that the hardware and software is supported by your product.
- Go to the IBM 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.

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 EMC Directive Conformance Statement

This product is in conformity with the protection requirements of EU Council Directive 2004/108/EC 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 Information Technology Equipment according to European Standard EN 55022. 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

Germany Class A Statement

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 EG Richtlinie 2004/108/EC (früher 89/336/EWG), 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, Gropiusplatz 10, 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: Dieses Produkt entspricht den Schutzanforderungen der EU-Richtlinie 2004/108/EG (früher 89/336/EWG) zur Angleichung der Rechtsvorschriften über die elektromagnetische Verträglichkeit in den EU-Mitgliedsstaaten und hält die Grenzwerte der EN 55022 Klasse A ein.

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 Betriebsmittein
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 2004/108/EG (früher 89/336/EWG) 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 EG Richtlinie 2004/108/EC (früher 89/336/EWG), 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, Gropiusplatz 10, 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 Genehmigungsverfahren ist 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。