Lenovo Network

Python Programming Guide

For Lenovo Cloud Network Operating System 10.6

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 (December 2017)

Contents

Who Should Use This Guide 16
Additional References 17
Terminology 18
Typographic Conventions 19

Chapter 1. Introduction to Python Scripting 21
About Python 22
About CNOS Python 23

Chapter 2. Running Python Scripts via the ISCLI 25
  Running a Basic Script 25
  Running a Basic Script with Arguments 25
Entering and Exiting the Python Shell 26

Chapter 3. Managing Python Scripts 27
Downloading a Script from a TFTP Server 28
Uploading a Script to a TFTP Server. 29
Editing a Script Directly on the Switch. 30
Deleting a Script 31
Viewing A List of Script Files. .32
Viewing Script File Content 33
Viewing Configured Scheduler Jobs. 34

Chapter 4. Using the CNOS Python Scheduler 35
Using the Python Scheduler 36
  Creating a Scheduled Python Job 36
  Using Crontab Format Arguments. 37
  Deleting a Job. 38
  Monitoring a Running Job 38
  Stopping a Running Job 38
  Viewing Python Logs 39
Creating Syslog Messages 40

Chapter 5. Writing Python Scripts 41
Script Components 42
  Viewing Online Help 44
  Python API Function Arguments 45
Script Examples 46
  ARP Configuration Example 46
  IP Configuration Example 47
  LAG Configuration Example 48
  LLDP Configuration Example .49
  VLAN Configuration Example .50

Chapter 6. The CNOS Python API 51
AAA Module 53
class AAA 53
get_accounting_def 53
set_accounting_def 53
get_authorization_cmds_def 54
set_authorization_cmds_def 54
get_authorization_conf_cmds_def 55
set_authorization_conf_cmds_def 55
get_authentication_login_con 56
set_authentication_login_con 56
get_authentication_login_def 57
set_authentication_login_def 57
get_authentication_login_err_enable 58
set_authentication_login_err_enable 58
unset_authentication_login_err_enable 58
get_maxfail_attempts 59
set_maxfail_attempts 59
get_user_default_role 59
set_user_default_role 60
unset_user_default_role 60
get_groups 60
ARP Module 61
class ARP 61
get_all_static_arp_entry() 61
get_one_static_arp_entry() 62
set_ip_arp() 62
delete_ip_arp() 63
get_arp_sys_pro() 63
set_arp_sys_pro() 63
get_arp_sys_interfaces() 64
set_arp_sys_pro_interface() 64
get_arp_refresh 65
set_arp_refresh 65

BGP Module 67
class BGP() 67
python_bgp_get_global_statistics() 67
python_bgp_clear_global_statistics() 68
python_show_bgp_peer_adj_routes() 68
python_bgp_get_status() 70
python_bgp_get_router_id() .70
python_bgp_get_as_number(). 70
python_bgp_get_hold_down_timer(). 71
python_bgp_get_keep_alive_timer() 71
python_bgp_get_enforce_first_as(). 71
python_bgp_get_fast_external_failover() 72
python_bgp_get_log_neighbor_changes(). 72
python_bgp_get_as_local_cnt() 72
python_bgp_get_maxas_limit() 73
python_bgp_get_synchronization() 73
python_bgp_get_bestpath_cfg() 73
python_bgp_get_confed_id() .75
python_bgp_get_confederation_peers(). 75
python_bgp_get_graceful_helper_status() 76
python_bgp_get_graceful_stalepath_time() 76
python_bgp_get_cluster_id() .77
python_show_ip_bgp(). 77
python_show_ip_bgp_network() 78
python_show_ip_bgp_summary() 81
python_show_ip_bgp_neighbor() 82
python_bgp_get_af_distance_config() 84
python_bgp_get_af_global_config() 85
python_bgp_get_af_maximum_paths_config() 86
python_bgp_get_af_nexthop_trigger_delay_config() 86
python_bgp_get_af_aggregate_config(). 87
python_bgp_get_af_dampening_config() 88
python_bgp_get_af_network_config() 89
python_bgp_get_af_redistribute_config() 89
python_show_ip_bgp_neighbor_stats(). 90
python_show_ip_bgp_neighbors_cfg. 91
python_put_ip_bgp_neighbors_cfg 94
python_bgp_set_unnumbered. .97
python_bgp_unset_unnumbered 97
class IP 98
set_if_bgp_unnumbered 98
unset_if_bgp_unnumbered 98
Boot Information Module 99
class BootInfo() 99
get_boot() 99
get_boot_ztp(). 100
set_boot_ztp() 100
get_boot_image() 100
set_boot_image() 101

CEE Module 103
class DCB 103
python_dcbx_get_interface_state 103
python_dcbx_set_state 104
python_dcbx_pfc_set_advt 104
python_dcbx_ets_set_advt 105
python_dcbx_app_set_advt. 105
python_dcbx_get_ctrl 106
python_dcbx_get_dcbxstate 106
python_dcbx_pfc_get_interface 107
python_dcbx_ets_get_interface 108
python_dcbx_app_get_interface. 109
python_dcbx_app_get_protocol_list_interface. 110
class CEE 110
python_cee_get_status 110
python_cee_set_status 111
class PFC 111
python_cee_pfc_get_state 111
python_cee_pfc_set_state. 111
python_cee_pfc_get_priority_map. 112
python_cee_pfc_set_priority_map 112
python_cee_pfc_interface_get_state 112
python_cee_pfc_interface_set_state 113
python_cee_pfc_interface_get_counters 113
class ETS. 114
python_cee_ets_get_config 114
python_cee_ets_set_config 114
class APP 115
python_cee_app_get_protocol_list. 115
python_cee_app_post_protocol 115
python_cee_app_del_protocol 116

DHCP Module 117
class DHCP() 117
get_dhcp_feature(). 117
set_dhcp_feature() 117
unset_dhcp_feature() 117
get_dhcpinfo() 118
set_dhcp_client() 119
unset_dhcp_client() 119
set_dhcpv6_client() 119
unset_dhcpv6_client() 120
set_dhcp_option_hostname() 120
unset_dhcp_option_hostname() 120
set_dhcp_option_ntp_server() 121
unset_dhcp_option_ntp_server() 121
set_dhcp_option_log_server() 121
unset_dhcp_option_log_server(). 122
set_dhcp_class_id() 122
unset_dhcp_class_id() 122
class DHCP_Relay() 123
get_relay_serv() 123
set_dhcpv4_relay(). 123
unset_dhcpv4_relay() 123
set_dhcpv6_relay(). 124
unset_dhcpv6_relay() 124
get_interface_relay_address() 124
set_relay4_address() 125
unset_relay4_address() 125
set_relay6_address() 126
unset_relay6_address() 126
class DHCP_Snooping 127
python_dhcpsnp_get_status_opt82 127
python_dhcpsnp_put_status_opt82 127
python_dhcpsnp_get_binding_entry 128
python_dhcpsnp_post_binding_entry 129
python_dhcpsnp_del_binding_entry 129
python_dhcpsnp_get_vlan 130
python_dhcpsnp_put_vlan 130
python_dhcpsnp_del_vlan 131
python_dhcpsnp_get_statistics 131
python_dhcpsnp_clear_statistics. 131
python_dhcpsnp_get_trust_port. 132
python_dhcpsnp_set_trust_port 132

DNS Module. 133
class DNS 133
enable_dns_domain_lookup 133
disable_dns_domain_lookup 133
add_dns_name_server 134
del_dns_name_server 134
add_default_domain. 135
del_default_domain 135
add_name_to_ip 136
del_name_to_ip 136
add_domain_name 137
del_domain_name. 137
dns_show_domain_list_info 138
ECMP Module 139
class WeightedEcmp 139
get_weighted_ecmp_state 139
set_weighted_ecmp_state 139
class WeightedEcmp_ipv4 140
set_weighted_nexthop_ipv4 140
class WeightedEcmp_ipv4_show 140
get_weighted_nexthop_ipv4 140
class WeightedEcmp_ipv6 141
set_weighted_nexthop_ipv6 141
class WeightedEcmp_ipv6_show 141
get_weighted_nexthop_ipv6 141
class WeightedEcmp_interface 142
set_weighted_ecmp_interface 142
class WeightedEcmp_interface_show 142
get_weighted_ecmp_interface. 142
FDB Module 143
class FDB 143
python_get_fdb_info 143
python_get_fdb_count_info 144
python_get_static_fdb_cfg 145
python_add_static_mac 146
python_remove_bridge_fdb 146
python_get_fdb_plearning_state 147
python_set_fdb_plearning_state. 147
python_get_fdb_glearning_cfg 148
python_set_fdb_glearning_cfg 148
python_get_fdb_aging_time_cfg 149
python_set_fdb_aging_time_cfg. 149
HostpCpy Module 151
class HostpCpy() 151
update_startup_cfg_tftp() 151
update_image_tftp() 152
get_update_image_status() 152
switch_reboot() 153

IGMP Module 155
class IgmpSnooping 155
python_igmp_snoop_get() 155
python_igmp_snoop_set() 155
class IgmpMcVlan. 156
python_igmp_snoop_all_if_get() 156
python_igmp_snoop_if_get() 156
python_igmp_snoop_all_if_get() 157
python_igmp_snoop_vlan_set() 157
IP Module 159
class IP() 159
get_ipinfo() 159
set_ip_addr() 160
set_bridge_port() 160
unset_bridge_port() 161
set_if_flagup(). 161
unset_if_flagup() 162
LACP Module 163
class LacpSystem 163
python_lacp_get_sys_priority() 163
python_lacp_get_max_bundle() 163
python_lacp_get_all_link_details() 163
python_lacp_set_system() 164
LAG Module 165
class LAG 165
python_get_lag() 165
python_get_lag_id() 166
python_update_lag_id() 167
python_update_lag_id_details() 168
python_create_lag_id() 169
python_delete_lag_all() 170
python_delete_lag_id() 170
LLDP Module 171
class LldpSystem 171
python_lldp_get_reinit_delay() 171
python_lldp_get_msg_tx_interval() 171
python_lldp_get_tx_delay() 171
python_lldp_set_reinit_delay() 172
python_lldp_set_msg_tx_interval() 172
python_lldp_set_tx_delay() 172
class LldpNeighbor 173
python_lldp_get_neighbor() 173
python_lldp_get_all_neighbor() 173
class LldpStats 174
python_lldp_get_statistics() 174
class LldpInterface 175
python_lldp_get_interface() 175
python_lldp_get_all_interface() 175
python_lldp_set_interface() 176

MSTP Module 177
class MSTP. 177
python_mstp_set_region_name() 177
python_mstp_get_region_name() 177
python_mstp_get_revision() 177
python_mstp_set_revision() 178
class MstpInstance 178
python_mstp_add_instance() 178
python_mstp_delete_instance() 179
python_mstp_update_instance() 179
python_mstp_set_instance_priority() 180
python_mstp_check_instance_exist(). 180
class MstpInterface 181
python_mstp_get_port_path_cost() 181
python_mstp_set_port_path_cost() 181
python_mstp_get_port_priority() 182
python_mstp_set_port_priority() 182
OSPF Module 183
class OSPF() 183
python_ospf_get_stats_info() 183
python_ospf_get_traffic_info() 187
python_ospf_get_neighbor_info() 189
python_ospf_get_routes_info() 190
python_ospf_get_database_info() 191
Platform Module 193
class PortInfo. 193
get_interface() 193
get_mac() 194
set_mac(). 194
is_enabled() 195
set_enabled() 195
set_mtu(). 196
get_port_speed() 196
set_port_speed() 197
get_duplex() 197
set_duplex() 198
class PortStatistics. 198
get_stats() 198
clear_stats 199

RADIUS Module 201
class Radius 201
get_global_key 201
set_global_key 201
get_global_retransmit 202
set_global_retransmit 202
get_global_timeout 202
set_global_timeout. 203
get_host 203
get_all_hosts 204
set_host 205
delete_host 206
get_group 206
get_all_groups 207
add_group 207
add_server_to_group 208
set_group_vrf 208
set_group_source_interface 209
delete_group 209
Route Module 211
class Route 211
set_route() 211
delete_route() 212
get_route() 213
Security Mode Module 215
class SecModeApi 215
get_current_security_mode 215
get_new_security_setting 215
set_security_mode 216
System Module. 217
Top-Level System Functions 217
client_connect() 217
client_disconnect() 217
class SystemInfo() 218
get_systemInfo(). 218
get_env_fan() 218
get_env_power() 219
get_env_temperature() 219
get_system_serial_num() 220
get_system_inventory() 220
get_hostname() 221
set_hostname() 221
get_system_core() 222

TACACS+ Module 223
class Tacacs 223
get_feature_status 223
set_feature_enabled 223
set_feature_disabled 223
get_global_key 224
set_global_key 224
get_host 225
get_all_hosts 225
set_host 226
delete_host 226
get_group 227
get_all_groups 227
add_group 228
add_server_to_group 228
set_group_vrf. 229
delete_group 229
Telemetry Module 231
class TelemetryBST_Tracking() 231
set_bst_tracking() 231
get_bst_tracking() 232
class TelemetryBST_Feature() 233
set_bst_feature() 233
get_bst_feature() 234
class TelemetryBST_Cgsn_Drop_Ctr() 236
get_bst_cgsn_drop_ctr() 236
class TelemetryBST_ClearStats() 237
get_bst_clear_stats() 237
class TelemetryBST_ClearThresholds(). 238
get_bst_clear_thresholds() 238
class TelemetryBST_Report() 238
get_bst_report() 238
class TelemetryBST_Threshold() 239
set_bst_threshold() 239
get_bst_threshold() 240
class TelemetryDevice_Feature 242
set_sys_feature() 242
get_sys_feature() 242
class Telemetry_DeviceProp() 243
get_swprop() 243
VLAN Module 245
class VlanSystem() 245
python_get_vlan() 245
python_create_vlan() 246
python_delete_vlan() 246
python_update_vlan_name() 247
python_update_vlan_admin_state() 247
class VlanEthIf 248
python_get_vlan_properties(). 248
python_update_vlan_properties() 249

VRF Module 251
class VRF 251
get_vrf_entry() 251
VRRP Module 253
class VRRP() 253
get_vrrp() 253
get_vrrp_intf() 254
get_vrrp_accept_mode() 255
get_vrrp_advt_interval() 256
get_vrrp_preempt_mode() 257
get_vrrp_priority(). 258
set_vrrp_accept_mode() 258
set_vrrp_advt_interval() 259
set_vrrp_preempt_mode() 259
set_vrrp_priority() 260
set_vrrp_switch_back_delay() 260
set_vrrp_oper_primary_ipaddr() 261
set_vrrp_monitored_circuit() 261
delete_vrrp_vr() 262
set_vrrp_vr() 262

Appendix A. Error Messages 263

Appendix B. Getting help and technical assistance 265

Appendix C. Notices. 267
Trademarks 269
Important Notes 270
Recycling Information. 271
Particulate Contamination 272
Telecommunication Regulatory Statement 273
Electronic Emission Notices 274
Federal Communications Commission (FCC) Statement 274
Industry Canada Class A Emission Compliance Statement 274
Avis de Conformité à la Réglementation d'Industrie Canada 274
Australia and New Zealand Class A Statement 274
European Union - Compliance to the Electromagnetic Compatibility Directive 275
Germany Class A Statement 275
Japan VCCI Class A Statement 276
Japan Electronics and Information Technology Industries Association (JEITA) Statement 277
Korea Communications Commission (KCC) Statement. 277
Russia Electromagnetic Interference (EMI) Class A statement 277
People's Republic of China Class A electronic emission statement 277
Taiwan Class A compliance statement 277

Preface

The *Lenovo Network Python Programming Guide for Cloud NOS 10.6* describes how to configure and use the Lenovo Cloud Network Operating System 10.6 software on the following Lenovo RackSwitches:

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

Who Should Use This Guide

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

Additional References

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

- *Lenovo Network Application Guide for Lenovo Cloud Network Operating System 10.6*
- *Lenovo Network Command Reference for Lenovo Cloud Network Operating System 10.6*
- *Lenovo Network Release Notes for Lenovo Cloud network Operating System 10.6*
- *Lenovo Network REST API Programming Guide for Lenovo Cloud Network Operating System 10.6*

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

Chapter 1. Introduction to Python Scripting

The Lenovo Cloud Network Operating System (CNOS) version 10.6 Python function API is a set of libraries of API functions that are embedded into the Lenovo switch Command-Line Interface (CLI) to support script execution.

About Python

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

Python is supported by the Python Software Organization, which is open source with an active user community. Python provides comprehensive set of libraries that includes many built-in modules and the ability to write scripts and functional extensions. Organizations from NASA to gaming and data security companies use Python for development. Python version 2.7 is installed on the switch version 10.6.

About CNOS Python

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

CNOS Python comes with the following features:

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

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

Chapter 2. Running Python Scripts via the ISCLI

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

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

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

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

Running a Basic Script

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

```
Switch# display script helloWorld.py

print "Hello world!"

Switch# python helloWorld.py

Hello world!
```

Running a Basic Script with Arguments

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

```
Switch# 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
```

Entering and Exiting the Python Shell

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

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

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

```
Switch# python

>>>
```

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

```
exit ()

Switch#
```

Chapter 3. Managing Python Scripts

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

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

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

Downloading a Script from a TFTP Server

To download a TFTP script, use the following command:

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

where:

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

Uploading a Script to a TFTP Server

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

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

where:

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

Editing a Script Directly on the Switch

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

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

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

Deleting a Script

To delete a script, use the following command:

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

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

To delete all Python scripts, use the following command:

```
Switch# no script all
```

Viewing A List of Script Files

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

```
Switch# 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 switch, 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 switch administrators and can be triggered by a timer, which is aligned to the `cron` service.

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

Using the Python Scheduler

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

Creating a Scheduled Python Job

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

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

where:

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

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

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

Using Crontab Format Arguments

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

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

where:

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

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

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

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

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

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

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

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

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

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

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

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

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

Deleting a Job

To delete a job, use the following command:

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

Monitoring a Running Job

To monitor a running job, use the following command:

```
Switch# display script running
```

Stopping a Running Job

To stop a running job, use the following steps:

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 switch. Python API functions extend the standard Python library to provide configuration, management, and monitoring abilities. These are located in several Python modules.

Script Components

The CNOS Python API contains the following modules:

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

  Note: You must import this module before importing anything else.
- `aaaApi`: Functions that manage the Authentication, Authorization and Accounting (AAA) switch configuration.
- `arpApi`: Functions that manage the Address Resolution Protocol (ARP) switch configuration.
- `bgpApi`: Functions that manage the Border Gateway Protocol (BGP) switch configuration.
- `bootInfoApi`: Functions that manage the switch boot properties.
- `dcbApi`: Functions that manage the Converged Enhanced Ethernet (CEE) switch configuration.
- `dhcpApi`: Functions that manage the Dynamic Host Configuration Protocol (DHCP) switch configuration.
- `dnsApi`: Functions that manage the Domain Name System (DNS) switch configuration.
- `fdbApi`: Functions that manage the Forwarding Database (FDB) switch configuration.
- `hostpCpyApi`: Functions that update image and configuration files via TFTP.
- `hostpRadiusApi`: Functions that manage the Remote Authentication Dial-In User Service (RADIUS) switch configuration.
- `hostpTacacsApi`: Functions that manage the Terminal Access Controller Access-Control System Plus (TACACS+) switch configuration.
- `igmpApi`: Functions that manage Internet Group Management Protocol (IGMP) Snooping switch configuration.
- `ipApi`: Functions that manage the Internet Protocol (IP) switch configuration.
- `lacpApi`: Functions that manage the Link Aggregation Control Protocol (LACP) switch configuration.
- `lagApi`: Functions that manage the Link Aggregation Group (LAG) switch configuration.
- `lldpApi`: Functions that manage the Link Layer Discovery Protocol (LLDP) switch configuration.
- `mstpApi`: Functions that manage the Multiple Spanning Tree Protocol (MSTP) switch configuration.
- `ospfApi`: Functions that manage the Open Shortest Path First (OSPF) switch configuration.
- `platformApi`: Functions that manage the switch port configuration.
- `routeApi`: Functions that manage the switch static route configuration.

- `secModeApi`: Functions that manage the switch security mode configuration.
- `telemetryApi`: Functions that manage the Telemetry switch configuration.
- `vlanApi`: Functions that configure VLAN properties.
- `vrfApi`: A function that manage Virtual Routing and Forwarding (VRF).
- `vrrpApi`: Functions that manage Virtual Router Redundancy Protocol (VRRP).
- `weightedEcmpApi`: Functions that manage the Equal Cost Multiple Paths (ECMP) switch configuration.

The following is a sample Python shell session:

```
Switch# python

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

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

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

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

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

Switch#
```

Viewing Online Help

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

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

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

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

Help on method python_lldp_get_all_neighbor in module lldpApi:

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

Python API Function Arguments

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

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

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

All Python API functions use keyword arguments.

Script Examples

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

ARP Configuration Example

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

```
import sys

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

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

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

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

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

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

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

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

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

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

IP Configuration Example

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

```
#Import modules
import systemApi, ipApi

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

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

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

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

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

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

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

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

LAG Configuration Example

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

```
#Import modules
import systemApi, lagApi

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

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

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

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

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

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

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

LLDP Configuration Example

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

```
#Import modules
import systemApi, lldpApi

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

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

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

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

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

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

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

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

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

VLAN Configuration Example

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

```
#Import modules
import systemApi, vlanApi

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

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

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

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

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

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

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

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

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

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

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

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

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

Chapter 6. The CNOS Python API

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

- "AAA Module" on page 53
- "ARP Module" on page 61
- "BGP Module" on page 67
- "Boot Information Module" on page 99
- "CEE Module" on page 103
- "DHCP Module" on page 117
- "DNS Module" on page 133
- "ECMP Module" on page 139
- "FDB Module" on page 143
- "HostpCpy Module" on page 151
- "IGMP Module" on page 155
- "IP Module" on page 159
- "LACP Module" on page 163
- "LAG Module" on page 165
- "LLDP Module" on page 171
- "MSTP Module" on page 177
- "OSPF Module" on page 183
- "Platform Module" on page 193
- "RADIUS Module" on page 201
- "Route Module" on page 211
- "Security Mode Module" on page 215
- "System Module" on page 217
- "TACACS+ Module" on page 223
- "Telemetry Module" on page 231
- "VLAN Module" on page 245
- "VRF Module" on page 251
- "VRRP Module" on page 253

AAA Module

The class in this module manages the Authentication, Authorization and Accounting (AAA) configuration on the switch. To use this module, in the Python file or in the Python interpreter, enter:

```
import aaaApi
```

class AAA

The functions in this class get and set AAA configurations.

get_accounting_def

Displays the current accounting configuration.

Syntax

```
get_accounting_def()
```

Returns

The current accounting configuration (string):

- `group`, followed by a list of up to eight AAA groups (optionally followed by `local`)
- `local`

set_accounting_def

Configures accounting on the switch.

Syntax

`set_accounting_def(`*<input_str>*`,[`*<groups>*`])`

where:

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

Returns

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

get_authorization_cmds_def

Displays the current User EXEC command mode authorization configuration.

Syntax

```
get_authorization_cmds_def()
```

Returns

The current authorization configuration (string):

- `group`, followed by a list of up to eight AAA groups (optionally followed by `local`)
- `local`

set_authorization_cmds_def

Enables or disables User EXEC command mode authorization.

Syntax

`set_authorization_cmds_def(`<*input_str*>`,[`<*groups*>`])`

where:

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

Returns

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

get_authorization_conf_cmds_def

Displays the current configuration command mode authorization configuration.

Syntax

```
get_authorization_conf_cmds_def()
```

Returns

The current authorization configuration (string):

- `group`, followed by a list of up to eight AAA groups (optionally followed by `local`)
- `local`

set_authorization_conf_cmds_def

Enables or disables configuration command mode authorization.

Syntax

`set_authorization_conf_cmds_def(`<*input_str*>`,[`<*groups*>`])`

where:

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

Returns

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

get_authentication_login_con

Displays the current console user login authentication configuration.

Syntax

```
get_authentication_login_con()
```

Returns

The current console user login authentication configuration (string):

- `group`, followed by a list of up to eight AAA groups (optionally followed by `local` or `none`)
- `local`
- `none`

set_authentication_login_con

Enables or disables console user login authentication.

Syntax

`set_authentication_login_con(`*<input_str>*`,[`*<groups>*`])`

where:

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

Returns

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

get_authentication_login_def

Displays the current remote user login authentication configuration.

Syntax

```
get_authentication_login_def()
```

Returns

The current remote user login authentication configuration (string):

- `group`, followed by a list of up to eight AAA groups (optionally followed by `local` or `none`)
- `local`
- `none`

set_authentication_login_def

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

Syntax

`set_authentication_login_def(`*<input_str>*`,[`*<groups>*`])`

where:

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

Returns

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

get_authentication_login_err_enable

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

Syntax

```
get_authentication_login_err_enable()
```

Returns

The status of the error messages as string:

- `enable` if error messages are displayed
- `disable` if error message are not displayed

set_authentication_login_err_enable

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

Syntax

```
set_authentication_login_err_enable()
```

Returns

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

unset_authentication_login_err_enable

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

Syntax

```
unset_authentication_login_err_enable()
```

Returns

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

get_maxfail_attempts

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

Syntax

```
get_maxfail_attempts()
```

Returns

The maximum number of unsuccessful authentication attempts (integer: *1 - 25*).

set_maxfail_attempts

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

Syntax

`set_maxfail_attempts(`*<maxfail_attempts>*`)`

where:

Variable	**Description**
maxfail_attempts	The maximum number of unsuccessful authentication attempts before a user is locked out (integer). Values: *1 - 25*

Returns

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

get_user_default_role

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

Syntax

```
get_user_default_role()
```

Returns

The status of the default user role configuration (string):

- `enable`
- `disable`

set_user_default_role

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

Syntax

```
set_user_default_role()
```

Returns

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

unset_user_default_role

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

Syntax

```
unset_user_default_role()
```

Returns

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

get_groups

Displays information about the configured AAA groups.

Syntax

```
get_groups()
```

Returns

A dictionary with information about the configured AAA groups:

Element	**Description**
group_name	The name of the AAA group (string).
type	The type of the AAA group (string). Values: `TACACS+`, `RADIUS`, or `LDAP`

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 (String). 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 (String).
`ip_addr`	The IP address (String).
`mac_addr`	The MAC address in xxxx.xxxx,xxxx format (String).

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 (String). **Note:** The interface must exist.
ip_addr	The IP address (String).

Returns

The static ARP entry, with the following parameters:

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

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 (String).
mac_addr	The MAC address in xxxx.xxxx,xxxx format (String).
if_name	The Ethernet interface name (String). **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 (String).
if_name	The Ethernet interface name (String). **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 (String). 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 (String).
`ageout_time`	The global ARP entry age out time, in seconds; an integer from 60-28800. Default value: `1500`.

set_arp_sys_pro_interface()

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 (String). 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`).

get_arp_refresh

Checks if ARP Refresh is enabled on the switch.

Syntax

```
get_arp_refresh()
```

Returns

A dictionary containing the variable `state` (string):

- `disabled`
- `enabled`

set_arp_refresh

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

Syntax

`set_arp_refresh(<`*enable*`>)`

where:

Variable	**Description**
enable	The status of ARP refresh (string). Values: `enabled` or `disabled`

Returns

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

BGP Module

The classes in this module manage Border Gateway Protocol (BGP) configurations. 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 (String).
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 (String); 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.
valid	Whether the path is valid; `Yes`. **Note:** This value only appears if the path is valid.

Element	Description
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.
state	Neighbor state.

Element	**Description**
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.
routeadv	Number of router advertisements.
update_if	Update interface.
update_source	Update source address.
established	Established count.

Element	**Description**
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.
nextHopAddress	Next-hop address.
nextHopLocalV6	Next-hop address (link local).
nextHopGlobalV6	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`.

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`.
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`.

Element	Description
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.

python_show_ip_bgp_neighbors_cfg

Displays BGP neighbor configuration information.

Syntax

`python_show_ip_bgp_neighbors_cfg([`<*ip_address*>`],[`<*vrf_name*>`])`

where:

Variable	Description
ip_address	(Optional) The IP address of the BGP neighbor (string).
vrf_name	(Optional) The VRF instance for the BGP neighbor (string).

Returns

A dictionary showing BGP neighbor information:

Element	Description
neighbor	The IP address of the BGP neighbor (string).
vrf_name	The VRF instance of the BGP neighbor (string).
remote as	The current remote AS number (integer). Values: *1 - 4294967295*
local as	The current local AS number (integer). Values: *1 - 4294967295*
address family	The BGP neighbor address family (integer). Values: `ipv4` or `ipv6`
advertisement interval	The configured minimum time interval, in seconds, between consecutive BGP updates (integer). Values: *1 - 65535*
bfd	The status of BFD (string). Values: `enabled`, `disabled`, or `multihop enabled`

Element	Description
connection retry time	The connection retry time, in seconds (integer). Values: *1 - 65535*
description	The BGP neighbor description (string).
disallow infinite holdtime	Whether the configuration of infinite hold-time is disallowed (string). Values: `yes` or `no`
do not capability negotiate	Whether capability negotiations are disabled (string). Values: `yes` or `no`
advertise dynamic capability	Whether dynamic capability advertisements are enabled (string). Values: `yes` or `no`
EBGP multihop	The number of EBGP multi-hops (integer). Values: *1 - 255*
remote private as	Whether the removal of private AS numbers from outbound routes updates is enabled (string). Values: `yes` or `no`
maximum peers	The maximum number of peers configured for the prefix of the BGP neighbor (string). Values: *1 - 96*
password	The encrypted password for the BGP neighbor (string).
shutdown	Whether the BGP neighbor is shut down (string). Values: `yes` or `no`
peer holdtime	The time interval, in seconds, the switch awaits before transitioning the BGP neighbor to IDLE state, if the switch doesn't receive an update or keep-alive message from the neighbor (integer). Values: *0 - 3600*
peer keepalive	The time interval, in seconds, the switch awaits before sending another keep-alive message to the BGP neighbor (integer). Values: *0 - 3600*
connection-mode passive	Whether the initiations of TCP sessions with the BGP neighbor are disabled (string). Displays whether the BGP neighbor is shut down. Values: `yes` or `no`
ttl security hops	The minimum number of TTL router hops an IP packet must have to not be discarded (integer). Values: *1 - 254*
update-source	The source of the BGP session and updates - string containing ethernet port, VLAN, and loopback interfaces information.

Element	**Description**
weight	The default weight of routes incoming from the BGP neighbor (integer). Values: *1 - 65535*
allow as in	Whether AS paths with the local AS number are accepted by the switch (string). Values: `yes` or `no`
default originate	Whether a default route to the BGP neighbor is configured (string). Values: `yes` or `no`
default originate map	The name of the route map for the default route (string).
prefix-list in	The prefix list for routes incoming from the BGP neighbor (string).
prefix-list out	The prefix list for routes outgoing to the BGP neighbor (string).
maximum-prefix	The maximum number of prefixes that can be received from the BGP neighbor (integer). Values: *1 - 15872*
maximum-prefix warning	Whether warning messages are generated only when the maximum prefix limit is exceeded (string). Values: `yes` or `no`
maximum-prefix threshold percent	The percentage of the maximum prefix limit at which the switch starts to generate a warning message (integer). Values: *1 - 100*
next-hop-self	Whether next-hop calculations for the BGP neighbor are disabled (string). Values: `yes` or `no`
filter-list in	The AS path ACL for routes incoming from the BGP neighbor (string).
filter-list out	The AS path ACL for routes outgoing to the BGP neighbor (string).
route-map in	The applied route map for routes incoming from the BGP neighbor (string).
route-map out	The applied route map for routes outgoing to the BGP neighbor (string).
route reflector client	Whether the BGP neighbor is configured as a route reflector client (string). Values: `yes` or `no`
send community	Whether community attributes are sent to the BGP neighbor (string). Values: `yes` or `no`

Element	**Description**
send community extended	Whether extended community attributes are sent to the BGP neighbor (string). Values: `yes` or `no`
soft reconfiguration inbound	Whether the switch is configured to store BGP neighbor updates (string). Values: `yes` or `no`
unsuppress-map	The name of the route map configured to selectively unsuppress suppressed routes (string).

python_put_ip_bgp_neighbors_cfg

Configures a BGP neighbor.

Syntax

`python_put_ip_bgp_neighbors_cfg(`*<ip_addr>*`,`*<dict_neigh>*`,[`*<vrf_name>*`])`

where:

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

The *dict_neigh* dictionary contains the following configuration details:

Element	**Description**
remote as	The current remote AS number (integer). Values: *1 - 4294967295*
local as	The current local AS number (integer). Values: *1 - 4294967295*
advertisement interval	The configured minimum time interval, in seconds, between consecutive BGP updates (integer). Values: *1 - 65535*
bfd	The status of BFD (string). Values: `enabled`, `disabled`, or `multihop enabled`
connection retry time	The connection retry time, in seconds (integer). Values: *1 - 65535*
description	The BGP neighbor description (string).

Element	**Description**
disallow infinite holdtime	Whether the configuration of infinite hold-time is disallowed (string). Values: `yes` or `no`
do not capability negotiate	Whether capability negotiations are disabled (string). Values: `yes` or `no`
advertise dynamic capability	Whether dynamic capability advertisements are enabled (string). Values: `yes` or `no`
EBGP multihop	The number of EBGP multi-hops (integer). Values: *1 - 255*
remote private as	Whether the removal of private AS numbers from outbound routes updates is enabled (string). Values: `yes` or `no`
maximum peers	The maximum number of peers configured for the prefix of the BGP neighbor (string). Values: *1 - 96*
password	The encrypted password for the BGP neighbor (string).
unencrypt-password	Whether the password is unencrypted (string). Values: `yes` or `no`
shutdown	Whether the BGP neighbor is shut down (string). Values: `yes` or `no`
peer holdtime	The time interval, in seconds, the switch awaits before transitioning the BGP neighbor to IDLE state, if the switch doesn’t receive an update or keep-alive message from the neighbor (integer). Values: *0 - 3600*
peer keepalive	The time interval, in seconds, the switch awaits before sending another keep-alive message to the BGP neighbor (integer). Values: *0 - 3600*
connection-mode passive	Whether the initiations of TCP sessions with the BGP neighbor are disabled (string). Displays whether the BGP neighbor is shut down. Values: `yes` or `no`
ttl security hops	The minimum number of TTL router hops an IP packet must have to not be discarded (integer). Values: *1 - 254*
update-source	The source of the BGP session and updates - string containing ethernet port, VLAN, and loopback interfaces information.

Element	Description
weight	The default weight of routes incoming from the BGP neighbor (integer). Values: *1 - 65535*
allow as in	Whether AS paths with the local AS number are accepted by the switch (string). Values: `yes` or `no`
default originate	Whether a default route to the BGP neighbor is configured (string). Values: `yes` or `no`
default originate map	The name of the route map for the default route (string).
prefix-list in	The prefix list for routes incoming from the BGP neighbor (string).
prefix-list out	The prefix list for routes outgoing to the BGP neighbor (string).
maximum-prefix	The maximum number of prefixes that can be received from the BGP neighbor (integer). Values: *1 - 15872*
maximum-prefix warning	Whether warning messages are generated only when the maximum prefix limit is exceeded (string). Values: `yes` or `no`
maximum-prefix threshold percent	The percentage of the maximum prefix limit at which the switch starts to generate a warning message (integer). Values: *1 - 100*
next-hop-self	Whether next-hop calculations for the BGP neighbor are disabled (string). Values: `yes` or `no`
filter-list in	The AS path ACL for routes incoming from the BGP neighbor (string).
filter-list out	The AS path ACL for routes outgoing to the BGP neighbor (string).
route-map in	The applied route map for routes incoming from the BGP neighbor (string).
route-map out	The applied route map for routes outgoing to the BGP neighbor (string).
route reflector client	Whether the BGP neighbor is configured as a route reflector client (string). Values: `yes` or `no`
send community	Whether community attributes are sent to the BGP neighbor (string). Values: `yes` or `no`

Element	**Description**
send community extended	Whether extended community attributes are sent to the BGP neighbor (string). Values: `yes` or `no`
soft reconfiguration inbound	Whether the switch is configured to store BGP neighbor updates (string). Values: `yes` or `no`
unsuppress-map	The name of the route map configured to selectively unsuppress suppressed routes (string).

Returns

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

python_bgp_set_unnumbered

Globally enables BGP unnumbered on the switch.

Syntax

`python_bgp_set_unnumbered(`<*as_number*>`)`

where:

Variable	**Description**
as_number	The BGP AS number (integer). Values: *1 - 4294967295*

Returns

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

python_bgp_unset_unnumbered

Globally disables BGP unnumbered on the switch.

Syntax

`python_bgp_unset_unnumbered(`<*as_number*>`)`

where:

Variable	**Description**
as_number	The BGP AS number (integer). Values: *1 - 4294967295*

Returns

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

class IP

The functions in this class set and unset BGP unnumbered interface configurations.

set_if_bgp_unnumbered

Enables BGP unnumbered on a switch interface.

Syntax

`set_if_bgp_unnumbered(<`*if_name*`>)`

where:

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

Returns

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

unset_if_bgp_unnumbered

Disables BGP unnumbered on a switch interface.

Syntax

`unset_if_bgp_unnumbered(<`*if_name*`>)`

where:

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

Returns

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

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 (String). 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 (String); one of `active`, `standby`.

Returns

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

CEE Module

The class in this module manages the Converged Enhanced Ethernet (CEE) configuration on the switch. To use this module, in the Python file or in the Python interpreter, enter:

```
import dcbApi
```

class DCB

The functions in this class get and set Data Center Bridging (DCB) configurations.

python_dcbx_get_interface_state

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

Syntax

`python_dcbx_get_interface_state(`*<if_name>*`)`

where:

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

Returns

A dictionary showing DCBX interface information:

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

python_dcbx_set_state

Configures DCBX on a switch interface.

Syntax

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

where:

Variable	**Description**
if_name	The name of the switch interface (string). For example: *Ethernet1/12*
enadis_string	The status of the DCBX (string). Values: `enable` or `disable`

Returns

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

python_dcbx_pfc_set_advt

Configures PFC local configuration advertisement on a switch interface.

Syntax

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

where:

Variable	**Description**
if_name	The name of the switch interface (string). For example: *Ethernet1/12*
enadis_string	The status of the PFC local configuration advertisement (string). Values: `on` or `off`

Returns

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

python_dcbx_ets_set_advt

Configures ETS local configuration advertisement on a switch interface.

Syntax

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

where:

Variable	Description
if_name	The name of the switch interface (string). For example: *Ethernet1/12*
enadis_string	The status of the ETS local configuration advertisement (string). Values: `on` or `off`

Returns

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

python_dcbx_app_set_advt

Configures application protocol local configuration advertisement on a switch interface.

Syntax

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

where:

Variable	Description
if_name	The name of the switch interface (string). For example: *Ethernet1/12*
enadis_string	The status of the application protocol local configuration advertisement (string). Values: `on` or `off`

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

python_dcbx_get_ctrl

Displays the DCBX control state machine for a switch interface.

Syntax

`python_dcbx_get_ctrl(`*<if_name>*`)`

where:

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

Returns

A dictionary showing DCBX control state machine interface information:

Element	Description
dcbx version	The version of DCBX (string). Values: `DCBX IEEE 802.1Qaz (v2.5)` or `DCBX CEE (v1.01)`
if_name	The name of the switch interface (string). For example: *Ethernet1/12*

python_dcbx_get_dcbxstate

Displays the status of DCBX for a switch interface.

Syntax

`python_dcbx_get_dcbxstate(`*<if_name>*`)`

where:

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

Returns

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

python_dcbx_pfc_get_interface

Displays the PFC configuration for a switch interface.

Syntax

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

where:

Variable	Description
if_name	The name of the switch interface (string). For example: *Ethernet1/12*
cfgtype	The type of PFC configuration (string). Values: `admin`, `operation`, or `remote`

Returns

A dictionary showing PFC interface configuration:

Element	Description
state	The status of PFC for the interface (string). Values: `on` or `off`
willing	Whether the switch is "willing" to learn PFC configurations from a DCBX peer (string). Values: `on` or `off`
advt	The status of PFC local configuration advertisement (string). Values: `on` or `off`
syncd	The status of PFC information synchronization (string). Values: `on` or `off`
priority_map	The priorities enabled on the interface (string).

python_dcbx_ets_get_interface

Displays the ETS configuration for a switch interface.

Syntax

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

where:

Variable	Description
if_name	The name of the switch interface (string). For example: *Ethernet1/12*
cfgtype	The type of PFC configuration (string). Values: `admin`, `operation`, or `remote`

Returns

A dictionary showing PFC interface configuration:

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

python_dcbx_app_get_interface

Displays the application protocol configuration for a switch interface.

Syntax

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

where:

Variable	**Description**
if_name	The name of the switch interface (string). For example: *Ethernet1/12*
cfgtype	The type of application protocol configuration (string). Values: `admin`, `operation`, or `remote`

Returns

A dictionary showing application protocol interface configuration:

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

python_dcbx_app_get_protocol_list_interface

Displays the application protocol list for a switch interface.

Syntax

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

where:

Variable	**Description**
if_name	The name of the switch interface (string). For example: *Ethernet1/12*
cfgtype	The type of application protocol configuration (string). Values: `admin`, `operation`, or `remote`

Returns

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

Element	**Description**
priority	The priority enabled for the protocol (integer). Values: *0 - 7*
protocol	The type of the protocol (string). Values: `ethertype`, `udp`, or `tcp`
protostr	The name of the protocol (string).

class CEE

The functions in this class set and get CEE configurations.

python_cee_get_status

Displays the CEE configuration of the switch.

Syntax

```
python_cee_get_status()
```

Returns

A dictionary showing the CEE configuration:

Element	**Description**
status	The status of CEE on the switch (string). Values: `on` or `off`

python_cee_set_status

Globally enables or disables CEE on the switch.

Syntax

`python_cee_set_status(`<*cee_cfg*>`)`

where:

Variable	Description
cee_cfg	The status of CEE on the switch (string). Values: `on` or `off`

Returns

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

class PFC

The functions in this class get and set PFC configurations.

python_cee_pfc_get_state

Displays the status of PFC on the switch.

Syntax

`python_cee_pfc_get_state()`

Returns

A string showing the status of PFC on the switch:

- `on`
- `off`

python_cee_pfc_set_state

Globally enables or disables PFC on the switch.

Syntax

`python_cee_pfc_set_state(`<*pfc_state*>`)`

where:

Variable	Description
pfc_state	The status of PFC (string). Values: `on` or `off`

Returns

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

python_cee_pfc_get_priority_map

Displays the list of configured PFC priorities.

Syntax

```
python_cee_pfc_get_priority_map()
```

Returns

A string showing the enabled list of PFC priorities.

python_cee_pfc_set_priority_map

Configures the PFC priority flow mapping.

Note: This function overwrites existing PFC priority configurations.

Syntax

`python_cee_pfc_set_priority_map(`<*new_priority_map*>`)`

where:

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

Returns

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

python_cee_pfc_interface_get_state

Displays the status of PFC for a switch interface.

Syntax

`python_cee_pfc_interface_get_state(`<*if_name*>`)`

where:

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

Returns

A string showing the status of PFC on the interface:

- `on`
- `off`

python_cee_pfc_interface_set_state

Enables or disables PFC on a switch interface.

Syntax

`python_cee_pfc_interface_set_state(`*<if_name>*`,`*<pfc_state>*`)`

where:

Variable	Description
if_name	The name of the switch interface (string). For example: *Ethernet1/12*
pfc_state	The status of PFC on the specified interface (string). Values: `on` or `off`

Returns

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

python_cee_pfc_interface_get_counters

Displays the PFC statistics for a switch interface.

Syntax

`python_cee_pfc_interface_get_counters(`*<if_name>*`)`

where:

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

Returns

A dictionary showing PFC interface statistics:

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

class ETS

The functions in this class get and set ETS configurations.

python_cee_ets_get_config

Displays the ETS configuration on the switch.

Syntax

```
python_cee_ets_get_config()
```

Returns

A dictionary showing the ETS configuration:

Element	**Description**
bandwidth	The bandwidth percentage allocated to a priority group (integer). Values: *0 - 100*
pgid	The ID of the priority group (integer). Values: *0 - 7*, or *15*
priority_pgid_mapping	The priorities mapped to the priority group. Type - list of priorities, which are integers between 0 and 7

python_cee_ets_set_config

Configures ETS.

Syntax

python_cee_ets_set_config(<*tcg_list*>)

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

Variable	**Description**
bandwidth	The bandwidth percentage allocated to a priority group (integer). Values: *0 - 100*
pgid	The ID of the priority group (integer). Values: *0 - 7*, or *15*
priority_pgid_mapping	The priorities mapped to the priority group. Type - list of priorities, which are integers between 0 and 7

Returns

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

class APP

The functions in this class get and set application protocol configurations.

python_cee_app_get_protocol_list

Displays the application protocol configuration.

Syntax

```
python_cee_app_get_protocol_list()
```

Returns

A dictionary showing the application protocol configuration:

Element	**Description**
priority	The priority enabled for the protocol (integer). Values: *0 - 7*
protocol	The type of the protocol (string). Values: `ethertype`, `udp`, or `tcp`
protoid	The name of the protocol (string).
config_name	The name of the application protocol configuration (string).

python_cee_app_post_protocol

Creates an application protocol configuration.

Syntax

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

where:

Element	**Description**
priority	The priority enabled for the protocol (integer). Values: *0 - 7*
protoid	The name of the protocol (string).
config_name	The name of the application protocol configuration (string).
protocol	The type of the protocol (string). Values: `ethertype`, `udp`, `tcp`, or `none`

Returns

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

python_cee_app_del_protocol

Deletes an application protocol configuration.

Syntax

`python_cee_app_del_protocol(`*<config_name>*`)`

where:

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

Returns

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

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 (String); 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 (String).

Returns

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

Element	Description
`if_name`	Interface name (String).
`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 (String).

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 (String).

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 (String).

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 (String).

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 (String).

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 (String).

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 (String).

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 (String).

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 (String).

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 (String).

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 (String).
class_id	The vendor class ID (String); 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 (String).

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 (String); one of `yes`, `no`. Default value: `no`.
`ena_v6_relay`	Whether DHCPv6 relay is enabled (String); 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 (String).

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 (String).
`dhcpv4_relay`	A dictionary containing DHCPv4 relay addresses.
`v4_relay_address`	DHCPv4 relay server address (String); a valid IPv4 address.
`dhcpv6_relay`	A dictionary containing DHCPv6 relay addresses.

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

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 (String).
ip_addr	The IP address (String); 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 (String).
ip_addr	The IP address (String); 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 (String).
ipv6_addr	The IP address (String); a valid IPv6 address.
out_if_name	(Optional) Relay outoing interface (String).

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 (String).
ipv6_addr	The IP address (String); a valid IPv6 address.
out_if_name	(Optional) Relay outoing interface (String).

Returns

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

class DHCP_Snooping

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

python_dhcpsnp_get_status_opt82

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

Syntax

```
python_dhcpsnp_get_status_opt82()
```

Returns

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

Element	**Description**
dhcpsnp_feature	The status of DHCP Snooping (string). Values: `enable` or `disable`
option_82	The status of DHCP Option 82 (string). Values: `enable` or `disable`

python_dhcpsnp_put_status_opt82

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

Syntax

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

where:

Variable	**Description**
dhcpsnp_feature	The status of DHCP Snooping (string). Values: `enable` or `disable`
option_82	The status of DHCP Option 82 (string). Values: `enable` or `disable`

Returns

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

python_dhcpsnp_get_binding_entry

Displays the DHCP Snooping binding table entries.

Syntax

```
python_dhcpsnp_get_binding_entry()
```

Returns

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

Element	**Description**
mac	The MAC address of the binding entry (string). Format: *XX:XX:XX:XX:XX:XX*
ip_addr	The IP address of the binding entry (string).
lease_time	The lease time, in seconds, of the binding entry (integer). Values: *1 - 4294967295*
type	The type of the binding entry (string). Values: `static` or `dynamic`
vlan	The VLAN ID of the binding entry (integer). Values: *1 - 4093*
if_name	The name of the switch interface associated with the binding entry (string). For example: *Ethernet1/12*

python_dhcpsnp_post_binding_entry

Adds entries to the DHCP Snooping binding table.

Syntax

`python_dhcpsnp_post_binding_entry(`*dict_dhcpsnp_entry*`)`

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

Variable	The MAC address of the binding entry (string). Format: *XX:XX:XX:XX:XX:XX*
mac	The IP address of the binding entry (string).
ip_addr	The lease time, in seconds, of the binding entry (integer). Values: *1 - 4294967295*
lease_time	The type of the binding entry (string). Values: `static` or `dynamic`
vlan	The VLAN ID of the binding entry (integer). Values: *1 - 4093*
if_name	The name of the switch interface associated with the binding entry (string). For example: *Ethernet1/12*

Returns

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

python_dhcpsnp_del_binding_entry

Deletes DHCP Snooping binding table entries.

Syntax

`python_dhcpsnp_del_binding_entry([`<*mac_vlan_if*>`])`

where:

Variable	**Description**
mac_vlan_if	(Optional) The binding entry identified by either its MAC address, VLAN ID, or interface name. Type - as follows: ● for MAC address - string with format *XX.XX.XX.XX.XX.XX* ● for VLAN ID - integer between 1 and 4093 ● for interface name - string (for example, *Ethernet1/12*)

Returns

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

python_dhcpsnp_get_vlan

Displays the VLANs for which DHCP Snooping is configured.

Syntax

```
python_dhcpsnp_get_vlan()
```

Returns

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

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

python_dhcpsnp_put_vlan

Configures DHCP Snooping on the specified VLAN.

Syntax

`python_dhcpsnp_put_vlan(`<*vlan_enabled*>`)`

where:

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

Returns

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

python_dhcpsnp_del_vlan

Disables DHCP Snooping on the specified VLAN.

Syntax

`python_dhcpsnp_del_vlan(<vlan_id>)`

where:

Variable	**Description**
vlan_id	The VLAN ID (integer). Values: *1 - 4093*

Returns

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

python_dhcpsnp_get_statistics

Displays DHCP Snooping statistics.

Syntax

`python_dhcpsnp_get_statistics()`

Returns

A dictionary showing DHCP Snooping statistics:

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

python_dhcpsnp_clear_statistics

Deletes all DHCP Snooping statistics.

Syntax

`python_dhcpsnp_clear_statistics()`

Returns

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

python_dhcpsnp_get_trust_port

Displays DHCP Snooping trusted interfaces.

Syntax

```
python_dhcpsnp_get_trust_port()
```

Returns

A dictionary with DHCP Snooping trusted interface information:

Element	**Description**
if_name	The name of the DHCP Snooping trusted interface (string). For example: *Ethernet1/12*
trusted	Whether the interface is trusted (string). Values: `yes` or `no`

python_dhcpsnp_set_trust_port

Configures DHCP Snooping trusted interfaces.

Syntax

python_dhcpsnp_set_trust_port(<*if_name*>,<*trusted*>)

where:

Variable	**Description**
if_name	The name of the switch interface (string). For example: *Ethernet1/12*
trusted	Whether the interface is trusted (string). Values: `yes` or `no`

Returns

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

DNS Module

The classes in this module manages the Domain Name Servers (DNS) configuration on the switch. To use this module, in the Python file or in the Python interpreter, enter:

```
import dnsApi
```

class DNS

The functions in this class get and set DNS configurations.

enable_dns_domain_lookup

Enables DNS on the switch.

Syntax

```
enable_dns_domain_lookup()
```

Returns

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

disable_dns_domain_lookup

Disables DNS on the switch.

Syntax

```
disable_dns_domain_lookup()
```

Returns

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

add_dns_name_server

Configures a single or multiple DNS servers on the switch.

Syntax

`add_dns_name_server(`*<vrf>*`,`*<nameserver1>*`,[`*<nameserver2>*`],`
`[`*<nameserver3>*`])`

where:

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

Returns

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

del_dns_name_server

Deletes a single or multiple already configured DNS servers.

Syntax

`del_dns_name_server(`*<vrf>*`,`*<nameserver1>*`,[`*<nameserver2>*`],`
`[`*<nameserver3>*`])`

where:

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

Returns

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

add_default_domain

Configures a default domain name.

Syntax

`add_default_domain(`*<domain_name>*`,[`*<vrf>*`])`

where:

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

Returns

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

del_default_domain

Deletes an already configured default domain name.

Syntax

`del_default_domain(`*<domain_name>*`,[`*<vrf>*`])`

where:

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

Returns

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

add_name_to_ip

Assigns a hostname to an IP address.

Syntax

`add_name_to_ip(`*<vrf>*`,`*<hostname>*`,`*<ip_addr1>*`,[`*<ip_addr2>*`])`

where:

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

Returns

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

del_name_to_ip

Deletes a hostname/IP address association.

Syntax

`del_name_to_ip(`*<vrf>*`,`*<hostname>*`,`*<ip_addr1>*`,[`*<ip_addr2>*`])`

where:

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

Returns

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

add_domain_name

Configures a domain name.

Syntax

`add_domain_name(`*<domain_name>*`,[`*<vrf>*`])`

where:

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

Returns

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

del_domain_name

Deletes an already configured domain name.

Syntax

`del_domain_name(`*<domain_name>*`,[`*<vrf>*`])`

where:

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

Returns

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

dns_show_domain_list_info

Displays DNS domain information.

Syntax

`dns_show_domain_list_info([<`*`vrf`*`>])`

where:

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

Returns

A dictionary showing DNS domain information.

Element	**Description**
domain_lookup	The status of DNS on the switch (string). Values: `enabled` or `disabled`
dynamic_domain	Dynamic domain information (string).
dynamic_nameserver	Dynamic DNS information (string).
domain_list	Domain information. Type - list of dictionaries containing string elements
nameserver_list	DNS server information. Type - list of dictionaries containing string elements
nametoip_list	Hostname/IP address associations. Type - list of dictionaries containing string elements (hostname to IP address mappings)

ECMP Module

The classes in this module manage the Equal-Cost Multi-Path (ECMP) configuration on the switch. To use this module, in the Python file or in the Python interpreter, enter:

```
import weightedEcmpApi
```

class WeightedEcmp

The functions in this class get and set weighted ECMP configurations.

get_weighted_ecmp_state

Checks if weighted ECMP is enabled on the switch.

Syntax

```
get_weighted_ecmp_state()
```

Returns

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

Element	**Description**
weighted_ecmp_state	The status of weighted ECMP (string). Values: `enable` or `disable`

set_weighted_ecmp_state

Enables or disables weighted ECMP on the switch.

Syntax

`set_weighted_ecmp_state(`*<state>*`)`

where:

Variable	**Description**
state	The status weighted ECMP (string). Values: `enable` or `disable`

Returns

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

class WeightedEcmp_ipv4

The function in this class sets ECMP weight for IPv4 next-hops.

set_weighted_nexthop_ipv4

Configures the weight of the specified IPv4 next-hop.

Syntax

`set_weighted_nexthop_ipv4(<addr>,<weight>)`

where:

Variable	**Description**
addr	The IPv4 address of the next-hop (string).
weight	The ECMP weight of the specified next-hop (string).

Returns

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

class WeightedEcmp_ipv4_show

The function in this class gets the ECMP weight of a specified IPv4 next-hop.

get_weighted_nexthop_ipv4

Displays the configured ECMP weight of a specified IPv4 next-hop.

Syntax

`get_weighted_nexthop_ipv4(<addr>)`

where:

Variable	**Description**
addr	The IPv4 address of the next-hop (string).

Returns

A dictionary showing ECMP weight information:

Element	**Description**
ipv4_nexthop_address	The IPv4 address of the next-hop (string).
ipv4_nexthop_weight	The ECMP weight of the specified next-hop (integer). Values: *1 - 4*

class WeightedEcmp_ipv6

The function in this class sets ECMP weight for IPv6 next-hops.

set_weighted_nexthop_ipv6

Configures the weight of the specified IPv6 next-hop.

Syntax

`set_weighted_nexthop_ipv6(`*<addr>*`,`*<weight>*`)`

where:

Variable	**Description**
addr	The IPv6 address of the next-hop (string).
weight	The ECMP weight of the specified next-hop (integer). Values: *1 - 4*

Returns

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

class WeightedEcmp_ipv6_show

The function in this class gets the ECMP weight of a specified IPv6 next-hop.

get_weighted_nexthop_ipv6

Displays the configured ECMP weight of a specified IPv6 next-hop.

Syntax

`get_weighted_nexthop_ipv6(`*<addr>*`)`

where:

Variable	**Description**
addr	The IPv6 address of the next-hop (string).

Returns

A dictionary showing ECMP weight information:

Element	**Description**
ipv6_nexthop_address	The IPv6 address of the next-hop (string).
ipv6_nexthop_weight	The ECMP weight of the specified next-hop (integer). Values: *1 - 4*

class WeightedEcmp_interface

The function in this class sets the ECMP weight for a specified switch interface.

set_weighted_ecmp_interface

Configures the ECMP weight of the specified switch interface.

Syntax

`set_weighted_ecmp_interface(`*<if_name>*`,`*<weight>*`)`

where:

Variable	**Description**
if_name	The name of the switch interface (string). For example: *Ethernet1/12*
weight	The ECMP weight of the specified interface (integer). Values: *1 - 4*

Returns

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

class WeightedEcmp_interface_show

The function in this class gets the ECMP weight of a specified switch interface.

get_weighted_ecmp_interface

Displays ECMP weight information for the specified switch interface.

Syntax

`get_weighted_ecmp_interface(`*<if_name>*`)`

where:

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

Returns

A dictionary showing ECMP weight information:

Element	**Description**
interface_name	The name of the switch interface (string). For example: *Ethernet1/12*
interface_weight	The ECMP weight of the specified interface (integer). Values: *1 - 4*

FDB Module

The classes in this module manages the Forwarding Database (FDB) configuration on the switch. To use this module, in the Python file or in the Python interpreter, enter:

```
import fdbApi
```

class FDB

The functions in this class get and set FDB configurations.

python_get_fdb_info

Displays all MAC addresses that match the search criteria.

Syntax

`python_get_fdb_info(`*<dict_fdb_filter>*`)`

where *dict_fdb_filter* contains the following optional variables:

Variable	**Description**
fdb_type	(Optional) The type of FDB (string). Values: `static`, `multicast`, or `dynamic`
mac_address	(Optional) The MAC address to filter on (string).
interfaces	(Optional) The list of switch interfaces to filter on. Type - list containing interface names, which are string (for example, *Ethernet1/12*)
vlan_id	(Optional) The VLANs to filter on (integer). Values: *1 - 4094*

Returns

Multiple possible return values:

- an error description string
- a boolean with the value `false`
- a dictionary showing MAC address information that matched the search filter:

Element	**Description**
address_table	The list of MAC addresses matching the search filter. Type - list with MAC addresses as string
is_static	Whether the MAC address is statically configured or dynamically learned (boolean). Values: `true` for static MAC address or `false` for dynamic MAC address
mac_address	The MAC address (string).

Element	Description
if_name	The name of the switch interface (string). For example: *Ethernet1/12*
vlan_id	The VLAN of the MAC address (integer). Values: *1 - 4094*

python_get_fdb_count_info

Displays the number of MAC table entries matching the search criteria.

Syntax

`python_get_fdb_count_info(`*<dict_fdb_filter>*`)`

where *dict_fdb_filter* contains the following optional variables:

Variable	Description
fdb_type	(Optional) The type of FDB (string). Values: `static`, `multicast`, or `dynamic`
mac_address	(Optional) The MAC address to filter on (string).
interfaces	(Optional) The list of switch interfaces to filter on. Type - list containing interface names, which are string (for example, *Ethernet1/12*)
vlan_id	(Optional) The VLANs to filter on (integer). Values: *1 - 4094*

Returns

Multiple possible return values:

- an error description string
- a boolean with the value `false`
- a dictionary showing MAC address information that matched the search filter:

Element	Description
dynamic_add_cnt	The number of dynamically learned MAC addresses matching the filter (integer).
static_add_cnt	The number of statically configured MAC addresses matching the filter (integer).
multicast_add_cnt	The number of multicast MAC addresses matching the filter (integer).
total_in_use_cnt	The total numbers of MAC address matching the filter. It's the sum of the number of dynamically learned, statically configured, and multicast MAC addresses (integer).

python_get_static_fdb_cfg

Displays all statically configured MAC addresses.

Syntax

`python_get_static_fdb_cfg(`<*dict_fdb_filter*>`)`

where *dict_fdb_filter* containg the following optional variables:

Variable	**Description**
mac_address	(Optional) The MAC address to filter on (string).
interfaces	(Optional) The list of switch interfaces to filter on. Type - list containing interface names, which are string (for example, *Ethernet1/12*)
vlan_id	The VLANs to filter on (integer). Values: *1 - 4094*

Returns

Multiple possible return values:

- an error description string
- a boolean with the value `false`
- a dictionary showing MAC address information that matched the search filter:

Element	**Description**
address_table	The list of MAC addresses matching the search filter. Type - list with MAC addresses as string
is_static	Whether the MAC address is statically configured or dynamically learned (boolean). Values: `true` for static MAC address or `false` for dynamic MAC address
mac_address	The MAC address (string).
if_name	The name of the switch interface (string). For example: *Ethernet1/12*
vlan_id	The VLAN of the MAC address (integer). Values: *1 - 4094*

python_add_static_mac

Configure a static MAC table entry or add interfaces to already configured static MAC table entries.

Syntax

`python_add_static_mac(`*<return_dict_added_static_fdb>*`)`

where *return_dict_added_static_fdb* contains the following mandatory variables:

Variable	**Description**
mac_address	The MAC address to configure (string).
interfaces	The list of switch interfaces for the MAC address. Type - list containing interface names, which are string (for example, *Ethernet1/12*)
vlan_id	The VLAN for the MAC address (integer). Values: *1 - 4094*

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `true` for success or `false` for failure

python_remove_bridge_fdb

Deletes existing MAC table entries or interfaces from already existing multicast MAC table entries.

Syntax

`python_remove_bridge_fdb(`*<dict_removed_fdb_filter>*`)`

where *dict_removed_fdb_filter* contains the following variables:

Variable	**Description**
fdb_type	The type of MAC address to delete (string). Values: `static` or `dynamic`
mac_address	(Optional) The MAC address to delete (string).
interfaces	(Optional) The list of switch interfaces for the multicast MAC address. Type - list containing interface names, which are string (for example, *Ethernet1/12*)
vlan_id	(Optional) The VLAN for the MAC address (integer). Values: *1 - 4094*

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `true` for success or `false` for failure

python_get_fdb_plearning_state

Displays the status of MAC learning for a specific switch interface.

Syntax

`python_get_fdb_plearning_state(`*<if_name>*`)`

where:

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

Returns

A dictionary showing the status of MAC learning on the specified interface:

Element	**Description**
learning_state	The status of MAC learning (string). Values: `enabled` or `disabled`

python_set_fdb_plearning_state

Configures MAC learning on a specific interface.

Syntax

`python_set_fdb_plearning_state(`*<if_name>*`,`*<learning_state>*`)`

where:

Variable	**Description**
if_name	The name of the switch interface (string). For example: *Ethernet1/12*
learning_state	The status of MAC learning (string). Values: `enabled` or `disabled`

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `true` for success or `false` for failure

python_get_fdb_glearning_cfg

Displays the status of global MAC learning on the switch.

Syntax

```
python_get_fdb_glearning_cfg()
```

Returns

A dictionary showing the status of global MAC learning on the switch:

Element	**Description**
global_learning_state	The status of global MAC learning (string). Values: `enabled` or `disabled`

python_set_fdb_glearning_cfg

Configures global MAC learning on the switch.

Syntax

`python_set_fdb_glearning_cfg(`<*glearning_state*>`)`

where:

Variable	**Description**
glearning_state	The status of global MAC learning (string). Values: `enabled` or `disabled`

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `true` for success or `false` for failure

python_get_fdb_aging_time_cfg

Displays MAC table entry aging time configuration.

Syntax

```
python_get_fdb_aging_time_cfg()
```

Returns

A dictionary showing the MAC aging time configuration:

Element	**Description**
aging_time	The MAC aging time, in seconds (integer). Values: *0 - 1000000*

python_set_fdb_aging_time_cfg

Configure MAC table entry aging time on the switch.

Syntax

`python_set_fdb_aging_time_cfg(`*<aging_time>*`)`

where:

Variable	**Description**
aging_time	The MAC aging time, in seconds (integer). Values: *0 - 1000000*

Returns

Multiple possible return values:

- error description string
- the status of the function execution as boolean: `true` for success or `false` for failure

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 (String); a valid IP address.
cfgfile	The configuration source file; a string up to 256 characters long.
vrf_name	(Optional) The VRF name; a string up to 64 characters long. Default value: `none`.

Returns

A dictionary that indicates the startup configuration update status:

Element	**Description**
`status`	Configuration update status (String).

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 (String); a valid IP address.
imgfile	The image file name; a string up to 256 characters long.
imgtype	System image type (String); 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 (String).

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 (String).

switch_reboot()

Halt the system and perform a cold restart.

Syntax

```
switch_reboot()
```

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 (String); 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 (String); 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 (String); one of `yes`, `no`. Default value: `yes`.
`fast_leave`	Whether IGMP fast leave is enabled (String); 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 (String); 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 (String); one of `yes`, `no`. Default value: `yes`.

Element	**Description**
`fast_leave`	Whether IGMP fast leave is enabled (String); 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 (String); 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 (String); 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 (String); one of `yes`, `no`. Default value: `yes`.
fast_leave	Whether IGMP fast leave is enabled (String); 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 (String); 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 (String); 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 (String); 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_bridge_port()

Make the specified interface a bridge port

Syntax

`set_bridge_port(`*<if_name>*`)`

where:

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

Returns

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

unset_bridge_port()

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

Syntax

`unset_bridge_port(`<*if_name*>`)`

where:

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

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 (String). **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 (String)
`lag_id`	LAG identifier; a positive integer from 1-65535.
`interfaces`	A dictionary containing physical interface members of the LAG: ● `if_name` ● `lag_mode` ● `lacp_prio` ● `lacp_timeout`
`if_name`	Ethernet interface name (String). **Note:** The interface must exist.
`lag_mode`	LAG mode (String); 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 (String); 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 (String)
`lag_id`	LAG identifier; a positive integer from 1-65535.
`interfaces`	A dictionary containing physical interface members of the LAG: ● `if_name` ● `lag_mode` ● `lacp_prio` ● `lacp_timeout` Up to 32 interfaces can be added.
`if_name`	Ethernet interface name (String). **Note:** The interface must exist.
`lag_mode`	LAG mode (String); 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 (String); one of `short`, `long`. Default value: `long`.
`suspend_ individual`	Whether the LACP state is suspended or individual (String); 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 (String).
`lag_id`	LAG identifier; a positive integer from 1-65535.
`interfaces`	A list of interfaces associated with this LAG, containing: ● `if_name` ● `lag_mode` ● `lacp_prio` ● `lacp_timeout` Up to 32 interfaces can be added.
`if_name`	Ethernet interface name (String). **Note:** The interface must exist.
`lag_mode`	LAG mode (String); 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 (String); one of `short`, `long`. Default value: `long`.
`suspend_ individual`	Whether the LACP state is suspended or individual (String); 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 (String). **Note:** The interface must exist.
`lag_mode`	LAG mode (String); 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 (String); 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 (String). **Note:** The interface must exist.
`lag_mode`	LAG mode (String); 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 (String); 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`).

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 (String).

Returns

A dictionary containing information about the specified port.

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

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 (String).
`capability`	Remote switch capability (String). 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 (String).
`system description`	Remote system description (String).

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 (String).

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 (String).

Returns

A dictionary of LLDP status information for the specified interface:

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

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 (String)
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 (String). **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 (String). **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 (String). **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`).

OSPF Module

The classes in this module contain functions that get and provide Open Shortest Path First (OSPF) information. To use this module, in the Python file or in the Python interpreter, enter:

```
import ospfApi
```

class OSPF()

The functions in this class get and set OSPF configurations.

python_ospf_get_stats_info()

Get OSPF global statistics.

Syntax

`python_ospf_get_stats_info(`*<vrf_name>*`)`

where:

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

Returns

A dictionary containing OSPF global statistics:

Element	Description
`vrf_name`	Default VRF name. Default value: `default`.
`ospf_id`	OSPF identifier. Default value: `0`.
`clr_timer_str`	Time since last OSPF process clear in HH:MM:SS format.
`router_id_changes`	Router-id changes counter; a positive integer.
`dr_election_counter`	DR elections counter; a positive integer.
`older_lsas_counter`	Older received LSAs counter; a positive integer.
`nbr_state_change_counter`	Neighbor state changes counter; a positive integer.

Element	Description
`nbr_bad_ lsreqs_ counter`	Neighbor bad LS received requests counter; a positive integer.
`nbr_ interval_ expired_ counter`	Neighbor dead-interval expirations counter; a positive integer.
`nbr_seq_ number_ mismatch`	Neighbor sequence number mismatches counter; a positive integer.
`spf_full`	Full SPF Computations counter; a positive integer.
`spf_ summary`	Summary SPF Computations counter; a positive integer.
`spf_ external`	External SPF Computations counter; a positive integer.
`recv_buf`	Received packet buffer; a positive integer.
`send_buf`	Sent packet buffer; a positive integer.
`lsa_buf`	LSA buffer; a positive integer.
`packet_ unuse`	Unused packets number; a positive integer.
`packet_max`	Maximum packets number; a positive integer.
`lsa_unuse`	Unused LSAs number; a positive integer.
`lsa_max`	Maximum LSAs number; a positive integer.
`router_lsa _type`	Router LSA type name; a positive integer.
`routerLsa_ generated`	Number of generated router LSAs; a positive integer.
`routerLsa_ refreshed`	Number of refreshed router LSAs; a positive integer.
`routerLsa_ flushed`	Number of flushed router LSAs; a positive integer.
`routerLsa_ agedOut`	Number of aged out router LSAs; a positive integer.
`networkLsa _generated`	Number of generated network LSAs; a positive integer.
`networkLsa _refreshed`	Number of refreshed network LSAs; a positive integer.

Element	Description
`networkLsa_flushed`	Number of flushed network LSAs; a positive integer.
`networkLsa_agedOut`	Number of aged out network LSAs; a positive integer.
`summaryLsa_generated`	Number of generated summary LSAs; a positive integer.
`summaryLsa_refreshed`	Number of refreshed summary LSAs; a positive integer.
`summaryLsa_flushed`	Number of flushed summary LSAs; a positive integer.
`summaryLsa_agedOut`	Number of aged out summary LSAs; a positive integer.
`asbrSummaryLsa_generated`	Number of generated ASBR summary LSAs; a positive integer.
`asbrSummaryLsa_refreshed`	Number of refreshed ASBR summary LSAs; a positive integer.
`asbrSummaryLsa_flushed`	Number of flushed ASBR summary LSAs; a positive integer.
`asbrSummaryLsa_agedOut`	Number of aged out ASBR summary LSAs; a positive integer.
`asExternalLsa_generated`	Number of generated AS-External LSAs; a positive integer.
`asExternalLsa_refreshed`	Number of refreshed AS-External LSAs; a positive integer.
`asExternalLsa_flushed`	Number of flushed AS-External LSAs; a positive integer.
`asExternalLsa_agedOut`	Number of aged out AS-External LSAs; a positive integer.
`asNssaLsa_generated`	Number of generated AS-NSSA LSAs; a positive integer.
`asNssaLsa_refreshed`	Number of refreshed AS-NSSA LSAs; a positive integer.

Element	Description
asNssaLsa_ flushed	Number of flushed AS-NSSA LSAs; a positive integer.
asNssaLsa_ agedOut	Number of aged out AS-NSSA LSAs; a positive integer.
type8Lsa_ generated	Number of generated type-8 LSAs; a positive integer.
type8Lsa_ refreshed	Number of refreshed type-8 LSAs; a positive integer.
type8Lsa_ flushed	Number of flushed type-8 LSAs; a positive integer.
type8Lsa_ agedOut	Number of aged out type-8 LSAs; a positive integer.
linkOpaque Lsa_ generated	Number of generated Link Opaque LSAs; a positive integer.
linkOpaque Lsa_ refreshed	Number of refreshed Link Opaque LSAs; a positive integer.
linkOpaque Lsa_ flushed	Number of flushed Link Opaque LSAs; a positive integer.
linkOpaque Lsa_ agedOut	Number of aged out Link Opaque LSAs; a positive integer.
areaOpaque _lsa_type	Area Opaque LSA type name; a positive integer.
areaOpaque Lsa_ generated	Number of generated Area Opaque LSAs; a positive integer.
areaOpaque Lsa_ refreshed	Number of refreshed Area Opaque LSAs; a positive integer.
areaOpaque Lsa_ flushed	Number of flushed Area Opaque LSAs; a positive integer.
areaOpaque Lsa_ agedOut	Number of aged out Area Opaque LSAs; a positive integer.
asOpaque_ lsa_type	AS Opaque LSA type name; a positive integer.

Element	Description
asOpaqueLsa_ generated	Number of generated AS External Opaque LSAs; a positive integer.
asOpaqueLsa_ refreshed	Number of refreshed AS External Opaque LSAs; a positive integer.
asOpaqueLsa_ flushed	Number of flushed AS External Opaque LSAs; a positive integer.
asOpaqueLsa_agedOut	Number of aged out AS External Opaque LSAs; a positive integer.

python_ospf_get_traffic_info()

Get OSPF traffic statistics.

Syntax

`python_ospf_get_traffic_info(<vrf_name>)`

where:

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

Returns

A dictionary containing OSPF traffic statistics:

Element	Description
vrf_name	Default VRF name. Default value: `default`.
ospf_id	OSPF identifier. Default value: `0`.
timer_str	Time since last OSPF process clear in HH:MM:SS format.
total_pkt_ in	Number of total packets in; a positive integer.
total_pkt_ out	Number of total packets out; a positive integer.
hello_in	Number of hello packets in; a positive integer.
hello_out	Number of hello packets out; a positive integer.
db_desc_in	Number of DB descriptor packets in; a positive integer.

Element	Description
db_desc_out	Number of DB descriptor packets out; a positive integer.
ls_req_in	Number of LS Request packets in; a positive integer.
ls_req_out	Number of LS Request packets out; a positive integer.
ls_upd_in	Number of LS Update packets in; a positive integer.
ls_upd_out	Number of LS Update packets out; a positive integer.
ls_ack_in	Number of LS ACK packets in; a positive integer.
ls_ack_out	Number of LS ACK packets out; a positive integer.
error_drops_in	Number of errors related to drops in; a positive integer.
error_drops_out	Number of errors related to drops out; a positive integer.
error_hellosin	Number of errors related to hellos in; a positive integer.
error_dbsin	Number of errors related to DB Descriptors; a positive integer.
error_lsreqin	Number of errors related to LS Requests; a positive integer.
error_lsuin	Number of errors related to LS Updates; a positive integer.
error_lsackin	Number of errors related to LS ACKs; a positive integer.
error_unknown_in	Number of errors related to unknown in; a positive integer.
error_unknown_out	Number of errors related to unknown out; a positive integer.
error_badcrc	Number of errors related to Bad CRC; a positive integer.
error_wrong_area	Number of errors related to Wrong Area; a positive integer.
error_bad_version	Number of errors related to Bad Version; a positive integer.
error_bad_auth	Number of errors related to Bad Authentication; a positive integer.
error_passive	Number of errors related to Passive; a positive integer.

Element	Description
`error_ nonbr`	Number of errors related to No Neighbor; a positive integer.
`error_ invalid_ src`	Number of errors related to Invalid Source; a positive integer.
`error_ invalid_ dst`	Number of errors related to Invalid Destination; a positive integer.
`error_ pktlength`	Number of errors related to Packet Length; a positive integer.

python_ospf_get_neighbor_info()

Get OSPF neighbors statistics.

Syntax

`python_ospf_get_neighbor_info(`*<vrf_name>*`)`

where:

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

Returns

A dictionary containing OSPF neighbor statistics:

Element	Description
`vrf_name`	Default VRF name. Default value: `default`.
`nbr_router _id`	Neighbor router ID identifier; a valid IPv4 or IPv6 address.
`priority`	The neighbor priority; an integer from 0-255.
`dead_timer`	The time left for dead interval expiry in HH:MM:SS format.
`nbr_addr`	Neighbor IP address; a valid IPv4 or IPv6 address.
`ifp_name`	Ethernet interface name.

python_ospf_get_routes_info()

Get OSPF routes statistics.

Syntax

`python_ospf_get_routes_info(`*<vrf_name>*`)`

where:

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

Returns

A dictionary containing OSPF routes statistics:

Element	**Description**
`Network`	Network name; a string in the format "AA:BB:CC:DD/MM".
`pathcode`	Path type; one of: ● `connected` ● `Discard` ● `OSPF` ● `OSPF inter area` ● `OSPF NSSA external type 1` ● `OSPF NSSA external type 2` ● `OSPF external type 1` ● `OSPF external type 2`
`pathCount`	Number of ecmp paths; a positive integer.
`route_path_cost`	Route-path cost; a positive integer.
`route_type2path_cost`	Route-type 2 path cost; a positive integer.
`next_hop_info`	Next-hop information; a list of dictionaries. Depending on the configuration, each dictionary may contain the following values: ● *interface:* Neighbor IP address; a valid IPv4 or IPv6 address. ● *area_id:* Neighbor area-id; a valid IPv4 or IPv6 address. ● *neighbor_addr:* Neighbor IP address; a valid IPv4 or IPv6 address.

python_ospf_get_database_info()

Get OSPF database statistics.

Syntax

`python_ospf_get_database_info(`*<vrf_name>*`)`

where:

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

Returns

A dictionary containing OSPF traffic statistics:

Element	**Description**
`link_state_id`	VRF name; a valid IPv4 or IPv6 address.
`adv_router`	Advertising router ID; a valid IPv4 or IPv6 address.
`lsa_type`	LSA type; one of: ● `Router-LSA` ● `Network-LSA` ● `Summary-LSA` ● `ASBR-summary-LSA` ● `AS-external-LSA` ● `AS-NSSA-LSA`
`lsa_age`	LSA age; a positive integer.
`ls_seqnum_str`	LS sequence number in hexadecimal format.
`checksum`	LSA checksum in hexadecimal format.
`link count`	Links number; a positive integer.
`area_id`	The area-id of the LSDB; a valid IPv4 or IPv6 address.
`route`	Network route (String).
`tag`	External/NSSA LSAs tag; a positive integer.
`metric_type`	Name of the type of metric; one of: `E1, E2, N1, N2`.

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 (String). 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 (String); one of `auto`, `full`, `half`.
`if_name`	The interface name (String).
`mtu`	The maximum transmission unit, in bytes; a positive integer from 64-9216. Default value: `1500`.
`admin_state`	The admin status (String); one of `up`, `down`.
`mac_addr`	The MAC address in xxxx.xxxx,xxxx format (String).
`speed`	The communication speed of the interface (String); 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 (String). **Note:** The interface must exist.

Returns

The MAC address:

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

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 (String). **Note:** The interface must exist.
mac_addr	The MAC address in xxxx.xxxx,xxxx format (String).

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 (String). **Note:** The interface must exist.

Returns

Enabled status:

Element	Description
`is_enabled`	Port enabled status (String); 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 (String). **Note:** The interface must exist.
flag	Snooping status (String); 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 (String). **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 (String). **Note:** The interface must exist.

Returns

The port speed:

Element	Description
`speed`	The communication speed of the interface (String); 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 (String). **Note:** The interface must exist.
speed	The communication speed of the interface (String); 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 (String). **Note:** The interface must exist.

Returns

The duplex of the interface:

Element	Description
`duplex`	The communication method of the interface (String); 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 (String). **Note:** The interface must exist.
duplex	The communication method of the interface (String); 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 name of the switch interface (string). For example: *Ethernet1/12*

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).

clear_stats

Resets statistics for the specified switch interface.

Note: This command is available only for non-aggregated switch interfaces.

Syntax

`clear_stats(<`*if_name*`>)`

where:

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

Returns

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

RADIUS Module

The classes in this module manages the Remote Authentication Dial-In User Service (RADIUS) configuration on the switch. To use this module, in the Python file or in the Python interpreter, enter:

```
import hostpRadiusApi
```

class Radius

The functions in this class get and set RADIUS configurations.

get_global_key

Checks if a RADIUS global authentication key is configured.

Syntax

```
get_global_key()
```

Returns

A string with possible values:

- `configured`
- `not configured`

set_global_key

Configures the RADIUS global authentication key.

Syntax

`set_global_key(`*<key>*`,[`*<key_from>*`])`

where:

Variable	**Description**
key	The RADIUS authentication key. Type - string
key_form	(Optional) The encryption method of the authentication key (integer). Values: `0` (clear text) or `7` (encrypted)

Returns

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

get_global_retransmit

Displays the number of retries the switch will make to establish a connection with a RADIUS server after the initial attempt failed.

Syntax

```
get_global_retransmit()
```

Returns

Integer: *0 - 5*

set_global_retransmit

Configures the number of retries the switch will make to establish a connection with a RADIUS server after the initial attempt failed.

Syntax

`set_global_restransmit(`*<retransmit>*`)`

where:

Variable	**Description**
retransmit	The number of retries (integer). Values: *0 - 5*

Returns

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

get_global_timeout

Displays the amount of time, in seconds, before a RADIUS server connection attempt is considered to have failed.

Syntax

```
get_global_timeout()
```

Returns

Integer: *1 - 60*

set_global_timeout

Configures the amount of time, in seconds, before a RADIUS server connection attempt is considered to have failed.

Syntax

`set_global_timeout(`*<timeout>*`)`

where:

Variable	**Description**
timeout	The time interval, in seconds, after which the RADIUS server will timeout (integer). Values: *1 - 60*

Returns

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

get_host

Displays information about an already configured RADIUS server.

Syntax

`get_host(`*<ip_addr>*`)`

where:

Variable	**Description**
ip_addr	The hostname or IP address of the RADIUS server (string).

Returns

A dictionary showing RADIUS server information:

Element	**Description**
ip_addr	The hostname or IP address of the RADIUS server (string).
auth-port	The port used for authentication (integer). Values: *1 - 65535*
acct-port	The port used for accounting (integer). Values: *1 - 65535*
retransmit	The number of retries the switch will make to establish a connection with a RADIUS server after the initial attempt failed (integer). Values: *0 - 5*

Element	**Description**
timeout	The amount of time, in seconds, before a RADIUS server connection attempt is considered to have failed (integer). Values: *1 - 60*
key	The status of the RADIUS server authentication key (string). Values: `configured` or `not configured`

get_all_hosts

Displays information about all configured RADIUS servers.

Syntax

```
get_all_hosts()
```

Returns

A list of dictionaries, each showing RADIUS server information:

Element	**Description**
ip_addr	The hostname or IP address of the RADIUS server (string).
auth-port	The port used for authentication (integer). Values: *1 - 65535*
acct-port	The port used for accounting (integer). Values: *1 - 65535*
retransmit	The number of retries the switch will make to establish a connection with a RADIUS server after the initial attempt failed (integer). Values: *0 - 5*
timeout	The amount of time, in seconds, before a RADIUS server connection attempt is considered to have failed (integer). Values: *1 - 60*
key	The status of the RADIUS server authentication key (integer). Values: `configured` or `not configured`

set_host

Configures a RADIUS server.

Syntax

`set_host(`*<ip_addr>*`,[`*<auth_port>*`],[`*<acct_port>*`],[`*<retransmit>*`],[`*<timeout>*`],[`*<key>*`],[`*<key_form>*`])`

where:

Variable	**Description**
ip_addr	The hostname or IP address of the RADIUS server (string).
auth_port	(Optional) The port used for authentication (integer). Values: *1 - 65535*
acct_port	(Optional) The port used for accounting (integer). Values: *1 - 65535*
retransmit	(Optional) The number of retries the switch will make to establish a connection with a RADIUS server after the initial attempt failed (integer). Values: *0 - 5*
timeout	(Optional) The amount of time, in seconds, before a RADIUS server connection attempt is considered to have failed (integer). Values: *1 - 60*
key	(Optional) The status of the RADIUS server authentication key (string). Values: `configured` or `not configured`
key_form	(Optional) The method of encryption for the authentication key (integer). Values: `0` (clear text) or `7` (encrypted)

Returns

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

delete_host

Deletes an already configured RADIUS server.

Syntax

`delete_host(<`*ip_addr*`>)`

where:

Variable	Description
ip_addr	The hostname or IP address of the RADIUS server (string).

Returns

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

get_group

Displays information about an already configured RADIUS server group.

Syntax

`get_group(<`*group_name*`>)`

where:

Variable	Description
group_name	The name of the RADIUS server group (string).

Returns

A dictionary showing information about the specified RADIUS server group:

Element	Description
group_name	The name of the RADIUS server group (string).
vrf_name	The VRF instance for the RADIUS server group (string).
hosts	Information about the RADIUS servers members of the specified group. Return values are the same as for get_all_hosts.
source_interface	The switch interface to connect to the RADIUS server (string). Values: interface name (for example, *Ethernet1/12*) or `not configured`

get_all_groups

Displays information about all configured RADIUS server groups.

Syntax

```
get_all_groups()
```

Returns

A list of dictionaries, each showing information about a specific RADIUS server group:

Element	**Description**
group_name	The name of the RADIUS server group (string).
vrf_name	The VRF instance for the RADIUS server group (string).
hosts	Information about the RADIUS servers members of the specified group. Return values are the same as for get_all_hosts.
source_interface	The switch interface to connect to the RADIUS server (string). Values: interface name (for example, *Ethernet1/12*) or `not configured`

add_group

Configures a RADIUS server group.

Syntax

`add_group(`*<group_name>*`)`

where:

Variable	**Description**
group_name	The name of the RADIUS server group (string).

Returns

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

add_server_to_group

Adds a RADIUS server to a RADIUS server group.

Syntax

`add_server_to_group(`*<group_name>*`,`*<server_ip>*`)`

where:

Variable	Description
group_name	The name of the RADIUS server group (string).
server_ip	The IP address of the RADIUS server (string).

Returns

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

set_group_vrf

Configures the VRF instance for the RADIUS server group.

Syntax

`set_group_vrf(`*<group_name>*`,`*<vrf_name>*`)`

where:

Variable	Description
group_name	The name of the RADIUS server group (string).
vrf_name	The name of the VRF instance (string).

Returns

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

set_group_source_interface

Configures the source switch interface to connect to the RADIUS server group.

Syntax

`set_group_source_interface(`*<group_name>*`,`*<source_interface>*`)`

where:

Variable	**Description**
group_name	The name of the RADIUS server group (string).
source_interface	The name of the switch interface (string). For example: *Ethernet1/12*

Returns

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

delete_group

Deletes the specified RADIUS server group.

Syntax

`delete_group(`*<group_name>*`)`

where:

Variable	**Description**
group_name	The name of the RADIUS server group (string).

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

Returns

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

delete_route()

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

Returns

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

get_route()

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 (String). **Note:** The named VRF must exist.
ip_dest	(Optional) The destination IP address (String); 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 (String).
`vrf_name`	The VRF name (String).
`if_name`	Interface name used for communication (String).

Security Mode Module

The classes in this module manages the Security Mode configuration on the switch. To use this module, in the Python file or in the Python interpreter, enter:

```
import secModeApi
```

class SecModeApi

The functions in this class get and set Security Mode configurations.

get_current_security_mode

Displays the current Security Mode.

Syntax

```
get_current_security_mode()
```

Returns

A string showing the current security mode, with possible values:

- `secure_mode`
- `legacy_mode`

get_new_security_setting

Displays what security mode is configured to run on the switch after reloading.

Syntax

```
get_new_security_setting()
```

Returns

A string showing the current security mode, with possible values:

- `secure_mode`
- `legacy_mode`

set_security_mode

Configures what security mode to run on the switch after reloading.

Syntax

`set_security_mode(`*<mode>*`)`

where:

Variable	**Description**
mode	The security mode (string). Values: `secure_mode` or `legacy_mode`

Returns

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

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 (String).
`Manufacturer`	Power supply manufacturer (String).
`Model`	Power supply model (String).
`State`	Power supply state (String).

get_env_temperature()

Returns power supply information for the switch.

Syntax

```
get_env_temperature()
```

Returns

A dictionary with switch temperature properties:

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).

Element	**Description**
`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 (String).

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.

Element	**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 (String).
`date`	Date (String).

TACACS+ Module

The classes in this module manages the Terminal Access Controller Access-Control System Plus (TACACS+) configuration on the switch. To use this module, in the Python file or in the Python interpreter, enter:

```
import hostpTacacsApi
```

class Tacacs

The functions in this class get and set TACACS+ configurations.

get_feature_status

Checks if TACACS+ is enabled on the switch.

Syntax

```
get_feature_status()
```

Returns

The status of the TACACS+ feature as a string with the following possible values:

- `enable`
- `disable`

set_feature_enabled

Enables TACACS+ on the switch.

Syntax

```
set_feature_enabled()
```

Returns

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

set_feature_disabled

Disables TACACS+ on the switch.

Syntax

```
set_feature_disabled()
```

Returns

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

get_global_key

Check if a TACACS+ global encryption key is enabled.

Syntax

```
get_global_key()
```

Returns

The status of the TACACS+ global encryption key as a string with the following possible values:

- `configured`
- `not configured`

set_global_key

Configures a TACACS+ global encryption key.

Syntax

`set_global_key(`*<key>*`,[`*<key_from>*`])`

where:

Variable	**Description**
key	The TACACS+ encryption key (string).
key_from	The type of TACACS+ encryption key (integer). Values: `0` (clear text password) or `7` (encrypted password) **Note:** The default encryption type is clear text password.

Returns

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

get_host

Displays information about a configured TACACS+ server.

Syntax

`get_host(<`*ip_addr*`>)`

where:

Variable	Description
ip_addr	The address of the TACACS+ server (string).

Returns

A dictionary showing information about the specified TACACS+ server.

Element	Description
ip_addr	The address of the TACACS+ server (string). Values: IPv4 address, IPv6 address or hostname
port	The port used to connect to TACACS+ server (integer). Values: *1 - 65535*
key	The status of the server encryption key configuration (string). Values: `configured` or `not configured`

get_all_hosts

Displays information about all configured TACACS+ servers.

Syntax

```
get_all_hosts()
```

Returns

A list of dictionaries, each showing information about a specific TACACS+ server:

Element	Description
ip_addr	The address of the TACACS+ servers (string). Values: IPv4 address, IPv6 address or hostname
port	The port used to connect to TACACS+ server (integer). Values: *1 - 65535*
key	The status of the server encryption key configuration (string). Values: `configured` or `not configured`

set_host

Configures a TACACS+ server.

Syntax

`set_host(`*<ip_addr>*`,[`*<port>*`],[`*<key>*`],[`*<key_form>*`])`

where:

Variable	**Description**
ip_addr	The TACACS+ server address (string). Values: IPv4 address, IPv6 address, or hostname
port	(Optional) The port used to connect to TACACS+ server (integer). Values: *1 - 65535*
key	(Optional) The server encryption key used to communicate with the TACACS+ server (string).
key_form	(Optional) The type of the server encryption key (string). Values: `0` (clear text password) or `7` (encrypted password) **Note:** The default encryption type is clear text password.

Returns

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

delete_host

Deletes a configured TACACS+ server.

Syntax

`delete_host(`*<ip_addr>*`)`

where:

Variable	**Description**
ip_addr	The TACACS+ server address (string). Values: IPv4 address, IPv6 address, or hostname

Returns

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

get_group

Displays information about a configured TACACS+ server group.

Syntax

`get_group(<group_name>)`

where:

Variable	Description
group_name	The name of the TACACS+ server group (string).

Returns

A dictionary showing information about the specified TACACS+ server group:

Element	Description
group_name	The name of the TACACS+ server group (string).
vrf_name	The name of the Virtual Routing and Forwarding (VRF) instance (string).
hosts	Displays information about the configured TACACS+ servers that are part of the specified group. The details are the same as the Return values for "get_all_hosts" on page 225.

get_all_groups

Displays information about all configured TACACS+ server groups.

Syntax

```
get_all_groups()
```

Returns

A list of dictionaries, each showing information about a specific TACACS+ server group:

Element	Description
group_name	The name of the TACACS+ server group (string).
vrf_name	The name of the Virtual Routing and Forwarding (VRF) instance (string).
hosts	Information about the configured TACACS+ servers that are members of the specified group. The details are the same as the Return values for get_all_hosts.

add_group

Configures a TACACS+ server group.

Syntax

`add_group(`*<group_name>*`)`

where:

Variable	**Description**
group_name	The TACACS+ server group name (string).

Returns

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

add_server_to_group

Adds a configured TACACS+ server to a TACACS+ server group.

Syntax

`add_server_to_group(`*<group_name>*`,`*<server_ip>*`)`

where:

Variable	**Description**
group_name	The name of the TACACS+ server group (string).
server_ip	The address of the TACACS+ server (string).

Returns

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

set_group_vrf

Configures the VRF instance used by a TACACS+ group.

Syntax

`set_group_vrf(`*<group_name>*`,`*<vrf_name>*`)`

where:

Variable	**Description**
group_name	The name of the TACACS+ server group (string).
vrf_name	The name of the VRF instance to be used by the specified TACACS+ server group (string).

Returns

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

delete_group

Deletes a TACACS+ server group.

Syntax

`delete_group(`*<group_name>*`)`

where:

Variable	**Description**
group_name	The name of the TACACS+ server group (string).

Returns

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

Telemetry Module

The classes in this module contain functions that get and provide telemetry information. To use this module, in the Python file or in the Python interpreter, enter:

```
import telemetryApi
```

class TelemetryBST_Tracking()

The functions in this class get and set buffer statistics tracking parameters.

set_bst_tracking()

Set buffer statistics tracking parameters on the switch.

Syntax

`set_bst_tracking(`*<dict_bst_tracking>*`)`

where *<dict_bst_tracking>* is a dictionary containing the following elements:

Element	**Description**
`track-peak-stats`	Set to 1 to enable peak statistics tracking, 0 to disable this feature
`track-ingress-port-priority-group`	Set to 1 to enable ingress port priority group tracking, 0 to disable this feature
`track-ingress-port-service-pool`	Set to 1 to enable ingress port service pool tracking, 0 to disable this feature
`track-ingress-service-pool`	Set to 1 to enable ingress service pool tracking, 0 to disable this feature
`track-egress-port-service-pool`	Set to 1 to enable egress port service pool tracking, 0 to disable this feature
`track-egress-service-pool`	Set to 1 to enable egress service pool tracking, 0 to disable this feature
`track-egress-rqe-queue`	Set to 1 to enable egress RQE queue tracking, 0 to disable this feature
`track-device`	Set to 1 to enable tracking of this device, 0 to disable this feature

Returns

A dictionary containing the following elements:

Element	Description
`Return Status`	Return status of the API call: ● `TRUE`: Successfully set the Buffer statistics. ● `FALSE`: Setting Buffer statistics failed
`Error Message`	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

get_bst_tracking()

Get buffer statistics tracking parameters.

Syntax

```
get_bst_tracking()
```

Returns

A dictionary containing the following elements:

Element	Description
`track-peak-stats`	Set to 1 to peak statistics tracking, 0 to disable this feature
`track-ingress-port-priority-group`	Set to 1 to enable ingress port priority group tracking, 0 to disable this feature
`track-ingress-port-service-pool`	Set to 1 to enable ingress port service pool tracking, 0 to disable this feature
`track-ingress-service-pool`	Set to 1 to enable ingress service pool tracking, 0 to disable this feature
`track-egress-port-service-pool`	Set to 1 to enable egress port service pool tracking, 0 to disable this feature
`track-egress-service-pool`	Set to 1 to enable egress service pool tracking, 0 to disable this feature
`track-egress-rqe-queue`	Set to 1 to enable egress RQE queue tracking, 0 to disable this feature
`track-device`	Set to 1 to enable tracking of this device, 0 to disable this feature

class TelemetryBST_Feature()

The functions in this class get and set buffer statistics feature parameters.

set_bst_feature()

Set buffer statistics and tracking feature parameters on the switch.

Syntax

`set_bst_feature(`*<dict_bst_feature>*`)`

where *<dict_bst_feature>* is a dictionary containing the following elements:

Element	**Description**
`bst-enable`	Set to 1 to enable BST, 0 to disable it. Enabling BST allows the switch to track buffer utilization statistics.
`send-async-reports`	Set to 1 to enable the transmission of periodic asynchronous reports, 0 to disable this feature.
`collection-interval`	The collection interval, in seconds. This defines how frequently periodic reports will be sent to the configured controller.
`trigger-rate-limit`	The trigger rate limit, which defines the maximum number of threshold-driven triggered reports that the agent is allowed to send to the controller per `trigger-rate-limit-interval`; an integer from 1-5. Default value: 1.
`trigger-rate-limit-interval`	The trigger rate limit interval, in seconds; an integer from 10-60. Default value: 1.
`send-snapshot-on-trigger`	Set to 1 to enable sending a complete snapshot of all buffer statistics counters when a trigger happens, 0 to disable this feature.
`async-full-report`	Set to 1 to enable the async full report feature, 0 to disable it. When this feature is enabled, the agent sends full reports containing data related to all counters. When the feature is disabled, the agent sends incremental reports containing only the counters that have changed since the last report.

Returns

A dictionary containing the following elements:

Element	Description
`Return Status`	Return status of the API call: ● `TRUE`: Successfully set the Buffer statistics feature. ● `FALSE`: Setting Buffer statistics feature failed
`Error Message`	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

get_bst_feature()

Get buffer statistics and tracking feature parameters.

Syntax

```
get_bst_feature()
```

Returns

A dictionary containing the following elements:

Element	Description
`bst-enable`	Set to 1 to enable BST, 0 to disable it. Enabling BST allows the switch to track buffer utilization statistics.
`send-async-reports`	Set to 1 to enable the transmission of periodic asynchronous reports, 0 to disable this feature.
`collection-interval`	The collection interval, in seconds. This defines how frequently periodic reports will be sent to the configured controller; an integer from 0-600. Default value: 60.
`trigger-rate-limit`	The trigger rate limit, which defines the maximum number of threshold-driven triggered reports that the agent is allowed to send to the controller per `trigger-rate-limit-interval`; an integer from 1-5. Default value: 1
`trigger-rate-limit-interval`	The trigger rate limit interval, in seconds; an integer from 10-60. Default value: 1.

Element	Description
`send-snapshot-on-trigger`	Set to 1 to enable sending a complete snapshot of all buffer statistics counters when a trigger happens, 0 to disable this feature.
`async-full-report`	Set to 1 to enable the async full report feature, 0 to disable it. When this feature is enabled, the agent sends full reports containing data related to all counters. When the feature is disabled, the agent sends incremental reports containing only the counters that have changed since the last report.

class TelemetryBST_Cgsn_Drop_Ctr()

The functions in this class manage buffer statistics tracking congestion drop counters.

get_bst_cgsn_drop_ctr()

Get buffer statistics congestion drop counters on the switch based on the request parameters.

Syntax

`set_bst_tracking(`*<dict_bst_cdropctr>*`)`

where *<dict_bst_cdropctr>* is a dictionary containing the following elements:

Element	**Description**
`request-type`	One of the following: ● `top-drops`: Show ports with maximum congestion on the switch and their drop-counters ● `top-port-queue-drops`: Show top port-queue level drop-counters on the switch ● port-queue-drops: Show per port-queue level drop-counters on the switch ● `port-drops`: Show per-port total drop counters on the switch
`request-params`	Request parameters; one of the following: ● `count`: Number of ports required in the report. The ports are sorted with the port suffering maximum congestion at the top; an integer. ● `queue-type`: Filters the report on the queue type; one of the following strings: – `ucast`: Unicast queues – `mcast`: Multicast queues – `all`: All supported queues ● `interface-list`: Comma-separated list of ports for the congestion drop counter report; an array. A value of all requests all the ports. ● `queue-list`: An array of queue numbers to be considered for the drop report. ● `collection-interval`: (Optional) The period in which the counters are collected from ASIC; An integer from 10-3600. Default value: 0.

Returns

A dictionary containing the following elements:

Element	**Description**
`Time-stamp`	Time of the report generation.
`report-type`	One of the following: ● `top-drops`: Show ports with maximum congestion on the switch and their drop-counters ● `top-port-queue-drops`: Show top port-queue level drop-counters on the switch ● port-queue-drops: Show per port-queue level drop-counters on the switch ● `port-drops`: Show per-port total drop counters on the switch
`congestion-ctr`	Congestion counters contents; a list of dictionaries. Depending on the configuration, each dictionary may contain the following values: ● `interface`: Interface name ● `ctr`: Counter value; a string. ● `queue-type`; one of ucast, mcast ● `queue-drop-ctr`; one of: – `queue number`: an integer from 1-8. – `counter value`: the 64 bit counter value; a string.

class TelemetryBST_ClearStats()

The functions in this class clear buffer statistics and tracking counters.

get_bst_clear_stats()

Clear buffer statistics and tracking counters.

Syntax

```
get_bst_clear_stats()
```

class TelemetryBST_ClearThresholds()

The functions in this class reset the buffer statistics and tracking thresholds to the default values.

get_bst_clear_thresholds()

Reset the buffer statistics and tracking thresholds to the default values.

Syntax

```
get_bst_clear_thresholds()
```

class TelemetryBST_Report()

The functions in this class retrieve the buffer statistics and tracking report.

get_bst_report()

Get buffer statistics congestion drop counters on the switch based on the request parameters.

Syntax

`get_bst_report(`*<dict_bst_report>*`)`

where *<dict_bst_report>* is a dictionary containing the following elements:where:

Element	**Description**
`include-ingress-port-priority-group`	When set, the Agent includes the ingress per-port per-priority group buffer statistics into the report; 1 to enable including realm to the report, 0 to disable.
`include-ingress-port-service-pool`	When set, the Agent includes the ingress per-port per-service pool buffer statistics into the report; 1 to enable including realm to the report, 0 to disable.
`include-ingress-service-pool`	When set, the Agent includes the ingress per-port per-service pool buffer statistics into the report; 1 to enable including realm to the report, 0 to disable.
`include-egress-port-service-pool`	When set, the Agent includes the egress per-port per-service pool buffer statistics into the report; 1 to enable including realm to the report, 0 to disable.
`include-egress-service-pool`	When set, the Agent includes the egress per-service pool buffer statistics into the report; 1 to enable including realm to the report, 0 to disable.
`include-egress-rqe-queue`	When set, the Agent includes the ingress per-RQE queue buffer statistics into the report; 1 to enable including realm to the report, 0 to disable.

Element	Description
`include-device`	When set, the Agent includes the ingress device level buffer statistics into the report; 1 to enable including realm to the report, 0 to disable.
`include-cpu-queue`	When set, the agent includes CPU queue statistics in the report.
`include-egress-uc-queue`	When set, the agent includes egress multicast queue statistics in the report.
`include-egress-mc-queue`	When set, the agent includes egress unicast queue statistics in the report.

class TelemetryBST_Threshold()

The functions in this class manage and configure buffer statistics and tracking thresholds.

set_bst_threshold()

Configure the buffer statistics and tracking thresholds.

Syntax

`set_bst_threshold(`*<dict_bst_threshold_cfg>*`)`

where *<dict_bst_threshold_cfg>* is a dictionary containing the following elements:

Realm	Index # 1	Index # 2	Statistics
`ingress-port-priority-group`	*interface* (such as `Ethernet1/7`)	`priority-group`	`um-share-buffer-count` `um-head room-buffer-count`
`ingress-port-service-pool`	*interface* (such as `Ethernet1/7`)	`service-pool`	`um-share-buffer-count`
`ingress-service-pool`	`service-pool`		`um-share-buffer-count`
`egress-port-service-pool`	*interface* (such as `Ethernet1/7`)	`service-pool`	`uc-share-buffer-count`, `um-share-buffer-count`, `mc-share-buffer-count`,
`egress-service-pool`	`service-pool`		`um-share-buffercount`, `mc-share-buffer-count`
`egress-rqe-queue`	`queue`		`rqe-buffer-count`
`include-device`			`data`

Realm	Index # 1	Index # 2	Statistics
`egress-uc-queue`	`queue`		`uc-threshold`
`egress-mc-queue`	`queue`		`mc-threshold`
`egress-cpu-queue`	`queue`		`cpu-threshold`

Returns

A dictionary containing the following elements::

Element	Description
`Return Status`	Return status of the API call: ● `TRUE`: Successfully set the Buffer statistics. ● `FALSE`: Setting Buffer statistics failed
`Error Message`	String value containing the reason for the API failure: ● none: When Return Status is `TRUE`. ● error string: When Return Status is `FALSE`.

get_bst_threshold()

Get buffer statistics and tracking threshold values.

Syntax

`get_bst_threshold(`*<dict_get_bst_threshold>*`)`

where *<dict_bst_report>* is a dictionary containing the following elements:

Element	Description
`include-ingress-port-priority-group`	When set, the Agent includes the ingress per-port per-priority group buffer statistics into the report; 1 to enable including realm to the report, 0 to disable.
`include-ingress-port-service-pool`	When set, the Agent includes the ingress per-port per-service pool buffer statistics into the report; 1 to enable including realm to the report, 0 to disable.
`include-ingress-service-pool`	When set, the Agent includes the ingress per-port per-service pool buffer statistics into the report; 1 to enable including realm to the report, 0 to disable.
`include-egress-port-service-pool`	When set, the Agent includes the egress per-port per-service pool buffer statistics into the report; 1 to enable including realm to the report, 0 to disable.

Element	Description
`include-egress-service-pool`	When set, the Agent includes the egress per-service pool buffer statistics into the report; 1 to enable including realm to the report, 0 to disable.
`include-egress-rqe-queue`	When set, the Agent includes the ingress per-RQE queue buffer statistics into the report; 1 to enable including realm to the report, 0 to disable.
`include-device`	When set, the Agent includes the ingress device level buffer statistics into the report; 1 to enable including realm to the report, 0 to disable.
`include-cpu-queue`	When set, the agent includes CPU queue statistics in the report.
`include-egress-uc-queue`	When set, the agent includes egress multicast queue statistics in the report.
`include-egress-mc-queue`	When set, the agent includes egress unicast queue statistics in the report.

Returns

A dictionary containing the following elements:

Realm	Index # 1	Index # 2	Statistics
`ingress-port-priority-group`	*interface* (such as `Ethernet1/7`)	`priority-group`	`um-share-threshold` `um-headroom-threshold`
`ingress-port-service-pool`	*interface* (such as `Ethernet1/7`)	`service-pool`	`um-share-threshold`
`ingress-service-pool`	`service-pool`		`um-share-threshold`
`egress-port-service-pool`	*interface* (such as `Ethernet1/7`)	`service-pool`	`uc-share-threshold`, `um-share-threshold`, `mc-share-threshold`,
`egress-service-pool`	`service-pool`		`um-share-threshold`, `mc-share-threshold`
`egress-rqe-queue`	`queue`		`rqe-threshold`
`include-device`			`threshold`
`egress-uc-queue`	`queue`		`uc-threshold`

Realm	Index # 1	Index # 2	Statistics
`egress-mc-queue`	`queue`		`mc-threshold`
`egress-cpu-queue`	`queue`		`cpu-threshold`

class TelemetryDevice_Feature

The functions in this class set and retrieve the device system parameters by enabling heartbeat and heartbeat interval.

set_sys_feature()

Configure the system features.

Syntax

`set_sys_feature(`*<dict_sys_feature>*`)`

where *<dict_sys_feature>* is a dictionary containing the following elements:

Element	Description
`heartbeat-enable`	When enabled, the Agent asynchronously sends the registration and heartbeat message to the collector; 1 to enable heartbeat, 0 to disable.
`msg-interval`	Determines the interval with which the registration and heartbeat messages are sent to the collector; units of seconds, range 1-600.

Returns

A dictionary containing the following elements::

Element	Description
`Return Status`	Return status of the API call: ● TRUE: Successfully set the Buffer statistics. ● FALSE: Setting Buffer statistics failed
`Error Message`	String value containing the reason for the API failure: ● none: When Return Status is TRUE. ● error string: When Return Status is FALSE.

get_sys_feature()

Get the system features parameters set on the switch.

Syntax

```
get_sys_feature()
```

Returns

A dictionary containing the following elements:

Element	Description
`heartbeat-enable`	When enabled, the Agent asynchronously sends the registration and heartbeat message to the collector; 1 to enable heartbeat, 0 to disable.
`msg-interval`	Determines the interval with which the registration and heartbeat messages are sent to the collector; units of seconds, range 1-600.

class Telemetry_DeviceProp()

The functions in this class contain methods to retrieve the device switch properties.

get_swprop()

Retrieve the device properties.

Syntax

```
get_swprop()
```

Returns

A dictionary containing the following elements:

Element	Description
`number-of asics`	Number of ASICS in the switch (integer).
`asic-info`	A dictionary consisting of the following values: ● `asic-id`: ASIC identifier on the switch (string) ● `chip-id`: part number of the silicon (string) ● `num-ports`: number of ports available on the switch, managed by this ASIC (integer)
`supported features`	A list of the features supported by the Agent (string).
`network-os`	The network operating system currently used on the switch.
`uid`	Unique identifier for the switch used by the SDN Controller to map the switch to the nodes existing in their discovery database.
`agent-ip`	IP address of the switch where the Agent is running (string).
`agent-port`	TCP port number of the switch at which the Agent is listening (string).
`agent-sw-version`	Software version number for the Agent (string).

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 (String).
`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 (String); one of `access`, `trunk`.
`if_name`	Ethernet interface name (String)
`admin_state`	The admin status of the VLAN (String); 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 (String). To create a VLAN with the default name, this field must be null.
vlan_id	The VLAN identifier; an integer from 2-3999.
admin_state	The admin status (String); 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 (String)

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 (String); 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 (*if_name* argument) is `None`.

Syntax

```
python_get_vlan_properties(if_name)
```

where:

Variable	Description
if_name	The interface name (String)

Returns

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

Element	Description
`if_name`	Ethernet interface name (String)
`bridgeport_mode`	Bridge port mode (String); 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.
`egress_type`	Whether the switch tags egress traffic when in hybrid bridge port mode. String with possible values: ● *tagged* ● *untagged*
`egress_type_ vlans`	A list of VLANs on which the switch tags egress traffic. Integer between *1* and *3999*.

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 (String)
bridgeport_mode	Bridge port mode (String); 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.
`egress_type`	Whether the switch tags egress traffic when in hybrid bridge port mode. String with possible values: ● *tagged* ● *untagged*
`egress_type_ vlans`	A list of VLANs on which the switch tags egress traffic. Integer between *1* and *3999*.

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	(Optional) The name of the virtual router (String). 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 (String).
`interfaces`	The interface (String).

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 (String); default value is `None`. **Note:** The interface must exist.

Returns

A list of VRRP VR information:

Element	**Description**
`if_name`	The Ethernet interface name (String).
`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 (String); one of `up` (default), `down`.
`oper_state`	The operation state of the VR (String); one of `master`, `backup`, `init`.
`track_if`	The interface to track by this VR (String). Default value: `none`. **Note:** If an interface is specified, it must exist.
`accept_mode`	Enables or disables the accept mode for this session (String); 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 (String); 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 (String). **Note:** The interface must exist.

Returns

A list of VRRP properties:

Element	Description
`if_name`	The Ethernet interface name (String).
`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 (String); one of `up` (default), `down`.
`oper_state`	The operation state of the VR (String); one of `master`, `backup`, `init`.
`track_if`	The interface to track by this VR (String). Default value: `none`. **Note:** If an interface is specified, it must exist.
`accept_mode`	Enables or disables the accept mode for this session (String); 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 (String); 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 (String). **Note:** The interface must exist.

Returns

The `accept_mode` for the session (String); 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 (String). **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 (String). **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 (String). **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 (String). **Note:** The interface must exist.
accept_mode	Whether to enable Accept mode for this VRRP session (String); 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 (String). **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 (String). **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 (String). **Note:** The interface must exist.
prio	The priority of the VR on the switch; an integer from 1-254. Default value: 100.

Returns

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

set_vrrp_switch_back_delay()

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 (String). **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 (String). **Note:** The interface must exist.
ipaddr	The primary IP address (String).

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 (String). **Note:** The interface must exist.
track_if	The interface to track by this VR. **Note:** If an interface is specified, it must exist.

Returns

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

delete_vrrp_vr()

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 (String). **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 (String). **Note:** The interface must exist.

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.

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 - Compliance to the Electromagnetic Compatibility Directive

This product is in conformity with the protection requirements of EU Council Directive 2004/108/EC (until April 19, 2016) and EU Council Directive 2014/30/EU (from April 20, 2016) on the approximation of the laws of the Member States relating to electromagnetic compatibility. Lenovo cannot accept responsibility for any failure to satisfy the protection requirements resulting from a non-recommended modification of the product, including the installation of option cards from other manufacturers.

This product has been tested and found to comply with the limits for Class A equipment according to European Standards harmonized in the Directives in compliance. The limits for Class A equipment were derived for commercial and industrial environments to provide reasonable protection against interference with licensed communication equipment.

Lenovo, Einsteinova 21, 851 01 Bratislava, Slovakia

Warning: This is a Class A product. In a domestic environment this product may cause radio interference in which case the user may be required to take adequate measures.

Germany Class A Statement

Deutschsprachiger EU Hinweis:

Hinweis für Geräte der Klasse A EU-Richtlinie zur Elektromagnetischen Verträglichkeit

Dieses Produkt entspricht den Schutzanforderungen der EU-Richtlinie 2014/30/EU (früher 2004/108/EC) zur Angleichung der Rechtsvorschriften über die elektromagnetische Verträglichkeit in den EU-Mitgliedsstaaten und hält die Grenzwerte der Klasse A der Norm gemäß Richtlinie.

Um dieses sicherzustellen, sind die Geräte wie in den Handbüchern beschrieben zu installieren und zu betreiben. Des Weiteren dürfen auch nur von der Lenovo empfohlene Kabel angeschlossen werden. Lenovo übernimmt keine Verantwortung für die Einhaltung der Schutzanforderungen, wenn das Produkt ohne Zustimmung der Lenovo verändert bzw. wenn Erweiterungskomponenten von Fremdherstellern ohne Empfehlung der Lenovo gesteckt/eingebaut werden.

Deutschland:

Einhaltung des Gesetzes über die elektromagnetische Verträglichkeit von Betriebsmitteln

Dieses Produkt entspricht dem „Gesetz über die elektromagnetische Verträglichkeit von Betriebsmitteln“ EMVG (früher „Gesetz über die elektromagnetische Verträglichkeit von Geräten“). Dies ist die Umsetzung der EU-Richtlinie 2014/30/EU (früher 2004/108/EC) in der Bundesrepublik Deutschland.

Zulassungsbescheinigung laut dem Deutschen Gesetz über die elektromagnetische Verträglichkeit von Betriebsmitteln, EMVG vom 20. Juli 2007 (früher Gesetz über die elektromagnetische Verträglichkeit von Geräten), bzw. der EMV EU Richtlinie 2014/30/EU (früher 2004/108/EC), für Geräte der Klasse A.

Dieses Gerät ist berechtigt, in Übereinstimmung mit dem Deutschen EMVG das EG-Konformitätszeichen - CE - zu führen. Verantwortlich für die Konformitätserklärung nach Paragraf 5 des EMVG ist die Lenovo (Deutschland) GmbH, Meitnerstr. 9, D-70563 Stuttgart.

Informationen in Hinsicht EMVG Paragraf 4 Abs. (1) 4:

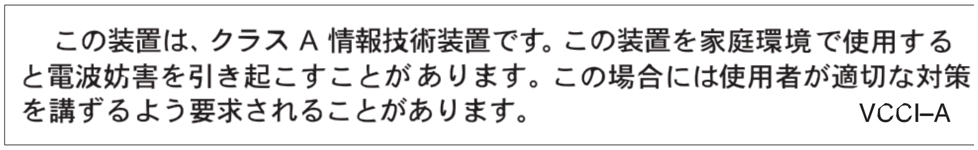
Das Gerät erfüllt die Schutzanforderungen nach EN 55024 und EN 55022 Klasse A.

Nach der EN 55022: „Dies ist eine Einrichtung der Klasse A. Diese Einrichtung kann im Wohnbereich Funkstörungen verursachen; in diesem Fall kann vom Betreiber verlangt werden, angemessene Maßnahmen durchzuführen und dafür aufzukommen.“

Nach dem EMVG: „Geräte dürfen an Orten, für die sie nicht ausreichend entstört sind, nur mit besonderer Genehmigung des Bundesministers für Post und Telekommunikation oder des Bundesamtes für Post und Telekommunikation betrieben werden. Die Genehmigung wird erteilt, wenn keine elektromagnetischen Störungen zu erwarten sind.“ (Auszug aus dem EMVG, Paragraph 3, Abs. 4). Dieses Genehmigungsverfahrenist nach Paragraph 9 EMVG in Verbindung mit der entsprechenden Kostenverordnung (Amtsblatt 14/93) kostenpflichtig.

Anmerkung: Um die Einhaltung des EMVG sicherzustellen sind die Geräte, wie in den Handbüchern angegeben, zu installieren und zu betreiben.

Japan VCCI Class A Statement



この装置は、クラス A 情報技術装置です。この装置を家庭環境で使用すると電波妨害を引き起こすことがあります。この場合には使用者が適切な対策を講ずるよう要求されることがあります。 VCCI–A

This is a Class A product based on the standard of the Voluntary Control Council for Interference (VCCI). If this equipment is used in a domestic environment, radio interference may occur, in which case the user may be required to take corrective actions.

Japan Electronics and Information Technology Industries Association (JEITA) Statement

高調波ガイドライン適合品

Japan Electronics and Information Technology Industries Association (JEITA) Confirmed Harmonics Guidelines (products less than or equal to 20 A per phase)

高調波ガイドライン準用品

Japan Electronics and Information Technology Industries Association (JEITA) Confirmed Harmonics Guidelines with Modifications (products greater than 20 A per phase).

Korea Communications Commission (KCC) Statement

이 기기는 업무용(A급)으로 전자파적합기기로
서 판매자 또는 사용자는 이 점을 주의하시기
바라며, 가정외의 지역에서 사용하는 것을 목
적으로 합니다.

This is electromagnetic wave compatibility equipment for business (Type A). Sellers and users need to pay attention to it. This is for any areas other than home.

Russia Electromagnetic Interference (EMI) Class A statement

ВНИМАНИЕ! Настоящее изделие относится к классу А.
В жилых помещениях оно может создавать радиопомехи, для
снижения которых необходимы дополнительные меры

People's Republic of China Class A electronic emission statement

中华人民共和国“A类”警告声明

声 明

此为A级产品，在生活环境中，该产品可能会造成无线电干扰。在这种情况下，可能需要用户对其干扰采取切实可行的措施。

Taiwan Class A compliance statement

警告使用者:
這是甲類的資訊產品，在
居住的環境中使用時，可
能會造成射頻干擾，在這
種情況下，使用者會被要
求採取某些適當的對策。