Lenovo Network

# Python Programming Guide

For Lenovo Cloud Network Operating System 10.8

Lenovo™

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

First Edition (July 2018)

# Contents


Who Should Use This Guide . . . . . 18
Additional References . . . . . 19
Terminology . . . . . 20
Typographic Conventions . . . . . 21

**Chapter 1. Introduction to Python Scripting . . . . . 23**
About Python . . . . . 24
About CNOS Python . . . . . 25

**Chapter 2. Running Python Scripts via the ISCLI . . . . . 27**
Running a Basic Script . . . . . 27
Running a Basic Script with Arguments . . . . . 27
Entering and Exiting the Python Shell . . . . . 28

**Chapter 3. Managing Python Scripts . . . . . 29**
Downloading a Script from a TFTP Server . . . . . 30
Uploading a Script to a TFTP Server . . . . . 31
Editing a Script Directly on the Switch . . . . . 32
Deleting a Script . . . . . 33
Viewing A List of Script Files . . . . . 34
Viewing Script File Content . . . . . 35
Viewing Configured Scheduler Jobs . . . . . 36

**Chapter 4. Using the CNOS Python Scheduler . . . . . 37**
Using the Python Scheduler . . . . . 38
Creating a Scheduled Python Job . . . . . 38
Using Crontab Format Arguments . . . . . 39
Deleting a Job . . . . . 40
Monitoring a Running Job . . . . . 40
Stopping a Running Job . . . . . 40
Viewing Python Logs . . . . . 41
Creating Syslog Messages . . . . . 42

**Chapter 5. Writing Python Scripts . . . . . 43**
Script Components . . . . . 44
Viewing Online Help . . . . . 46
Python API Function Arguments . . . . . 47
Script Examples . . . . . 48
ARP Configuration Example . . . . . 48
IP Configuration Example . . . . . 49
LAG Configuration Example . . . . . 50
LLDP Configuration Example . . . . . 51
VLAN Configuration Example . . . . . 52

**Chapter 6. The CNOS Python API . . . . . 53**
AAA Module . . . . . 55
class AAA . . . . . 55
get_accounting_def . . . . . 55
set_accounting_def . . . . . 55
get_authorization_cmds_def . . . . . 56
set_authorization_cmds_def . . . . . 56
get_authorization_conf_cmds_def . . . . . 56
set_authorization_conf_cmds_def . . . . . 57
get_authentication_login_con . . . . . 57
set_authentication_login_con . . . . . 58
get_authentication_login_def . . . . . 58
set_authentication_login_def . . . . . 59
get_authentication_login_err_enable . . . . . 59
set_authentication_login_err_enable . . . . . 59
unset_authentication_login_err_enable . . . . . 60
get_maxfail_attempts . . . . . 60
set_maxfail_attempts . . . . . 60
get_user_default_role . . . . . 61
set_user_default_role . . . . . 61
unset_user_default_role . . . . . 61
get_groups . . . . . 62
ARP Module . . . . . 63
class ARP . . . . . 63
get_all_static_arp_entry() . . . . . 63
get_one_static_arp_entry() . . . . . 64
set_ip_arp() . . . . . 64
delete_ip_arp() . . . . . 65
get_arp_sys_pro() . . . . . 65
set_arp_sys_pro() . . . . . 65
get_arp_sys_interfaces() . . . . . 66
set_arp_sys_pro_interface() . . . . . 66
get_arp_refresh . . . . . 67
set_arp_refresh . . . . . 67

BGP Module . . . . . .69
class BGP() . . . . . .69
python_bgp_get_global_statistics() . . . . . .69
python_bgp_clear_global_statistics() . . . . . .70
python_show_bgp_peer_adj_routes() . . . . . .70
python_bgp_get_status() . . . . . .72
python_bgp_get_router_id() . . . . . .72
python_bgp_get_as_number() . . . . . .72
python_bgp_get_hold_down_timer() . . . . . .73
python_bgp_get_keep_alive_timer() . . . . . .73
python_bgp_get_enforce_first_as() . . . . . .73
python_bgp_get_fast_external_failover() . . . . . .74
python_bgp_get_log_neighbor_changes() . . . . . .74
python_bgp_get_as_local_cnt() . . . . . .74
python_bgp_get_maxas_limit() . . . . . .75
python_bgp_get_synchronization() . . . . . .75
python_bgp_get_bestpath_cfg() . . . . . .75
python_bgp_get_confed_id() . . . . . .77
python_bgp_get_confederation_peers() . . . . . .77
python_bgp_get_graceful_helper_status() . . . . . .78
python_bgp_get_graceful_stalepath_time() . . . . . .78
python_bgp_get_cluster_id() . . . . . .79
python_show_ip_bgp() . . . . . .79
python_show_ip_bgp_network() . . . . . .80
python_show_l2vpn_bgp_network() . . . . . .83
python_show_ip_bgp_summary() . . . . . .85
python_show_ip_bgp_neighbors() . . . . . .86
python_bgp_get_af_distance_config() . . . . . .89
python_bgp_get_af_global_config() . . . . . .89
python_bgp_get_af_maximum_paths_config() . . . . . .90
python_bgp_get_af_nexthop_trigger_delay_config() . . . . . .91
python_bgp_get_af_aggregate_config() . . . . . .91
python_bgp_get_af_dampening_config() . . . . . .92
python_bgp_get_af_network_config() . . . . . .93
python_bgp_get_af_redistribute_config() . . . . . .94
python_bgp_put_af_redistribute_config() . . . . . .95
python_show_ip_bgp_neighbor_stats() . . . . . .96
python_show_ip_bgp_neighbors_cfg . . . . . .96
python_put_ip_bgp_neighbors_cfg . . . . . 100
python_bgp_set_unnumbered . . . . . 102
python_bgp_unset_unnumbered . . . . . 103
python_bgp_get_dscp() . . . . . 103
python_bgp_put_dscp() . . . . . 104
Boot Information Module . . . . . 105
class BootInfo() . . . . . 105
get_boot() . . . . . 105
get_boot_ztp() . . . . . 106
set_boot_ztp() . . . . . 106
get_boot_image() . . . . . 106
set_boot_image() . . . . . 107

CEE Module . . . . . 109
class DCB . . . . . 109
python_dcbx_get_interface_state . . . . . 109
python_dcbx_set_state . . . . . 110
python_dcbx_pfc_set_advt . . . . . 110
python_dcbx_ets_set_advt . . . . . 111
python_dcbx_app_set_advt. . . . . 111
python_dcbx_get_ctrl . . . . . 112
python_dcbx_get_dcbxstate . . . . . 112
python_dcbx_pfc_get_interface . . . . . 113
python_dcbx_ets_get_interface . . . . . 114
python_dcbx_app_get_interface. . . . . 115
python_dcbx_app_get_protocol_list_interface. . . . . 116
class CEE . . . . . 116
python_cee_get_status . . . . . 116
python_cee_set_status . . . . . 117
class PFC . . . . . 117
python_cee_pfc_get_state . . . . . 117
python_cee_pfc_set_state. . . . . 117
python_cee_pfc_get_priority_map. . . . . 118
python_cee_pfc_set_priority_map . . . . . 118
python_cee_pfc_interface_get_state . . . . . 118
python_cee_pfc_interface_set_state . . . . . 119
python_cee_pfc_interface_get_counters . . . . . 119
class ETS. . . . . 120
python_cee_ets_get_config . . . . . 120
python_cee_ets_set_config . . . . . 120
class APP . . . . . 121
python_cee_app_get_protocol_list. . . . . 121
python_cee_app_post_protocol . . . . . 121
python_cee_app_del_protocol . . . . . 122

DHCP Module . . . . . 123
class DHCP_Client() . . . . . 123
get_dhcp_feature() . . . . . 123
set_dhcp_feature() . . . . . 123
unset_dhcp_feature() . . . . . 123
get_dhcpinfo() . . . . . 124
set_dhcp_client() . . . . . 125
unset_dhcp_client() . . . . . 125
set_dhcpv6_client() . . . . . 125
unset_dhcpv6_client() . . . . . 126
set_dhcp_option_hostname() . . . . . 126
unset_dhcp_option_hostname() . . . . . 126
set_dhcp_option_ntp_server() . . . . . 127
unset_dhcp_option_ntp_server() . . . . . 127
set_dhcp_option_log_server() . . . . . 127
unset_dhcp_option_log_server() . . . . . 128
set_dhcp_class_id() . . . . . 128
unset_dhcp_class_id() . . . . . 128
class DHCP_Relay() . . . . . 129
get_relay_serv() . . . . . 129
set_dhcpv4_relay() . . . . . 129
unset_dhcpv4_relay() . . . . . 129
set_dhcpv6_relay() . . . . . 130
unset_dhcpv6_relay() . . . . . 130
get_interface_relay_address() . . . . . 130
set_relay4_address() . . . . . 131
unset_relay4_address() . . . . . 131
set_relay6_address() . . . . . 132
unset_relay6_address() . . . . . 132
class DHCP_Snooping . . . . . 133
python_dhcpsnp_get_status_opt82 . . . . . 133
python_dhcpsnp_put_status_opt82 . . . . . 133
python_dhcpsnp_get_binding_entry . . . . . 134
python_dhcpsnp_post_binding_entry . . . . . 135
python_dhcpsnp_del_binding_entry . . . . . 135
python_dhcpsnp_get_vlan . . . . . 136
python_dhcpsnp_put_vlan . . . . . 136
python_dhcpsnp_del_vlan . . . . . 137
python_dhcpsnp_get_statistics . . . . . 137
python_dhcpsnp_clear_statistics . . . . . 137
python_dhcpsnp_get_trust_port . . . . . 138
python_dhcpsnp_set_trust_port . . . . . 138

DNS Module . . . 139
class DNS . . . 139
enable_dns_domain_lookup . . . 139
disable_dns_domain_lookup . . . 139
add_dns_name_server . . . 140
del_dns_name_server . . . 140
add_default_domain. . . . 141
del_default_domain . . . 141
add_name_to_ip . . . 142
del_name_to_ip . . . 142
add_domain_name . . . 143
del_domain_name. . . . 143
dns_show_domain_list_info . . . 144
Dot1QEncaps Module . . . 145
class Dot1qEncapsulation() . . . 145
set_dot1q_tag_interface() . . . 145
unset_dot1q_tag_interface() . . . 145
get_dot1q_tag_interface() . . . 146
ECMP Module . . . 147
class WeightedEcmp . . . 147
get_weighted_ecmp_state . . . 147
set_weighted_ecmp_state . . . 147
class WeightedEcmp_ipv4 . . . 148
set_weighted_ecmp_nexthop_ipv4 . . . 148
class WeightedEcmp_ipv4_show . . . 148
get_weighted_ecmp_ipv4 . . . 148
class WeightedEcmp_ipv6 . . . 149
set_weighted_ecmp_nexthop_ipv6 . . . 149
class WeightedEcmp_ipv6_show . . . 149
get_weighted_ecmp_ipv6 . . . 149
class WeightedEcmp_interface . . . 150
set_weighted_ecmp_interface . . . 150
class WeightedEcmp_interface_show . . . 150
get_weighted_ecmp_interface. . . . 150
FDB Module . . . 151
class FDB . . . 151
python_get_fdb_info . . . 151
python_get_fdb_count_info . . . 152
python_get_static_fdb_cfg . . . 153
python_add_static_mac . . . 154
python_remove_bridge_fdb . . . 154
python_get_fdb_plearning_state . . . 155
python_set_fdb_plearning_state. . . . 155
python_get_fdb_glearning_cfg . . . 156
python_set_fdb_glearning_cfg . . . 156
python_get_fdb_aging_time_cfg . . . 157
python_set_fdb_aging_time_cfg. . . . 157

HostpCpy Module . . . 159
class HostpCpy() . . . 159
update_startup_cfg_tftp() . . . 159
update_image_tftp() . . . 160
get_update_image_status() . . . 160
switch_reboot() . . . 161
IGMP Module . . . 163
class IgmpSnooping . . . 163
python_igmp_snoop_get() . . . 163
python_igmp_snoop_set() . . . 163
class IgmpMcVlan . . . 164
python_igmp_snoop_all_if_get() . . . 164
python_igmp_snoop_if_get() . . . 164
python_igmp_snoop_all_if_get() . . . 165
python_igmp_snoop_vlan_set() . . . 165
IP Module . . . 167
class IP() . . . 167
get_ipinfo() . . . 167
set_ip_addr() . . . 168
set_bridge_port() . . . 168
unset_bridge_port() . . . 169
set_if_flagup() . . . 169
unset_if_flagup() . . . 170
set_if_bgp_unnumbered . . . 170
unset_if_bgp_unnumbered . . . 171
LACP Module . . . 173
class LacpSystem . . . 173
python_lacp_get_sys_priority() . . . 173
python_lacp_get_max_bundle() . . . 173
python_lacp_get_all_link_details() . . . 173
python_lacp_set_system() . . . 174
LAG Module . . . 175
class LAG . . . 175
python_get_lag() . . . 175
python_get_lag_id() . . . 176
python_update_lag_id() . . . 177
python_update_lag_id_details() . . . 178
python_create_lag_id() . . . 179
python_delete_lag_all() . . . 180
python_delete_lag_id() . . . 180

LLDP Module . . . 181
class LldpSystem . . . 181
python_lldp_get_reinit_delay() . . . 181
python_lldp_get_msg_tx_interval() . . . 181
python_lldp_get_tx_delay() . . . 181
python_lldp_set_reinit_delay() . . . 182
python_lldp_set_msg_tx_interval() . . . 182
python_lldp_set_tx_delay() . . . 182
class LldpNeighbor . . . 183
python_lldp_get_neighbor() . . . 183
python_lldp_get_all_neighbor() . . . 183
class LldpStats . . . 184
python_lldp_get_statistics() . . . 184
class LldpInterface . . . 184
python_lldp_get_interface() . . . 184
python_lldp_get_all_interface() . . . 185
python_lldp_set_interface() . . . 185
MSTP Module . . . 187
class MSTP . . . 187
python_mstp_set_region_name() . . . 187
python_mstp_get_region_name() . . . 187
python_mstp_get_revision() . . . 187
python_mstp_set_revision() . . . 188
class MstpInstance . . . 188
python_mstp_add_instance() . . . 188
python_mstp_delete_instance() . . . 189
python_mstp_update_instance() . . . 189
python_mstp_set_instance_priority() . . . 190
python_mstp_check_instance_exist() . . . 190
class MstpInterface . . . 191
python_mstp_get_port_path_cost() . . . 191
python_mstp_set_port_path_cost() . . . 191
python_mstp_get_port_priority() . . . 192
python_mstp_set_port_priority() . . . 192
Nexthop Health Check . . . 193
class NexthopHealthCheck() . . . 193
set_nexthop_healthcheck_interval() . . . 193
unset_nexthop_healthcheck() . . . 193
NTP Authentication Module . . . 195
class NtpSystem() . . . 195
python_ntp_show_keys_info() . . . 195
python_ntp_set_auth_keys() . . . 195
NWV Authentication Module . . . 197
class NwvCfg() . . . 197
get_nwv() . . . 197
set_nwv_mode() . . . 197

OSPF Module . . . . . 199
class OSPF() . . . . . 199
python_ospf_get_stats_info() . . . . . 199
python_ospf_get_traffic_info() . . . . . 202
python_ospf_get_neighbor_info() . . . . . 204
python_ospf_get_routes_info() . . . . . 205
python_ospf_get_database_info() . . . . . 206
python_ospf_get_border_routers_info() . . . . . 207
python_ospf_get_summary_address_info() . . . . . 208
python_ospf_get_interface_info() . . . . . 208
python_ospf_get_vlinks_info() . . . . . 211
python_set_ospf_if_ospf_status() . . . . . 212
python_set_ospf_if_hello_interval . . . . . 212
python_set_ospf_if_dead_interval() . . . . . 213
python_set_ospf_if_retransmit_interval() . . . . . 213
python_set_ospf_if_transmit_delay() . . . . . 214
python_set_ospf_if_priority() . . . . . 214
python_set_ospf_if_output_cost() . . . . . 215
python_set_ospf_if_network_type() . . . . . 215
python_set_ospf_if_mtu() . . . . . 216
python_set_ospf_if_mtu_ignore() . . . . . 216
python_set_ospf_if_bfd() . . . . . 217
python_set_ospf_if_authentication_type() . . . . . 217
python_set_ospf_if_authentication_key() . . . . . 218
python_set_ospf_if_message_digest_key() . . . . . 218
python_set_ospf_if_passive() . . . . . 219
python_set_ospf_if_area_id() . . . . . 219
python_set_ospf_if_db_filter_out() . . . . . 220
python_set_ospf_if_unset_config() . . . . . 220
python_set_ospf_if_unset_message_digest_key_config() . . . . . 221
python_set_ospf_virtual_link() . . . . . 221
python_set_ospf_virtual_link_disable() . . . . . 222
python_set_ospf_virtual_hello_interval() . . . . . 222
python_set_ospf_virtual_dead_interval() . . . . . 223
python_set_ospf_virtual_retransmit_interval() . . . . . 223
python_set_ospf_virtual_transmit_delay() . . . . . 224
python_set_ospf_virtual_bfd() . . . . . 224
python_set_ospf_virtual_authentication_type() . . . . . 225
python_set_ospf_virtual_authentication_key() . . . . . 225
python_set_ospf_virtual_ message_digest _key() . . . . . 226
python_set_ospf_virtual_unset_config() . . . . . 226
python_set_ospf_virtual_unset_message_digest_key_config() . . . . . 227
python_ospf_get_proc_info() . . . . . 227
python_ospf_get_multiarea_info() . . . . . 229
python_ospf_get_ribcounters_info() . . . . . 230
python_ospf_get_redist_config() . . . . . 231
python_ospf_put_redist_config() . . . . . 233
python_ospf_get_nssa_config() . . . . . 234
python_ospf_put_nssa_config() . . . . . 235
python_ospf_put_ area_def_cost_config() . . . . . 236
python_ospf_put_area_auth_config() . . . . . 236
python_ospf_put_summary_addr_config() . . . . . 237
python_ospf_put_area_range_config() . . . . . 238

python_ospf_put_overflow_db_config() . . . 239
python_ospf_put_refbw_config() . . . 240
python_ospf_put_stub_config() . . . 240
python_ospf_put_clear_config() . . . 241
python_ospf_put_procinfo_config() . . . 242
Platform Module . . . 243
class PortInfo . . . 243
get_interface() . . . 243
get_mac() . . . 244
set_mac() . . . 244
is_enabled() . . . 245
set_enabled() . . . 245
set_mtu() . . . 246
get_port_speed() . . . 246
set_port_speed() . . . 247
get_duplex() . . . 247
set_duplex() . . . 248
class PortStatistics . . . 248
get_stats() . . . 248
clear_stats . . . 249
Private VLAN Module . . . 251
class Pvlan() . . . 251
manageGlobalPvlan() . . . 251
managePvlan() . . . 251
managePvlanAssoc() . . . 252
setPortMode() . . . 252
managePortMapping() . . . 253
managePortAssociation() . . . 254
showPvlanInfo() . . . 254
showPvlanInterfaceInfo() . . . 255
RADIUS Module . . . 257
class Radius . . . 257
get_global_key . . . 257
set_global_key . . . 257
get_global_retransmit . . . 258
set_global_retransmit . . . 258
get_global_timeout . . . 258
set_global_timeout . . . 259
get_host . . . 259
get_all_hosts . . . 260
set_host . . . 261
delete_host . . . 262
get_group . . . 262
get_all_groups . . . 263
add_group . . . 263
add_server_to_group . . . 264
set_group_vrf . . . 264
set_group_source_interface . . . 265
delete_group . . . 265

Route Module . . . 267
class Route . . . 267
set_route() . . . 267
delete_route() . . . 268
get_route() . . . 269
Routemap Module . . . 271
class Routemap . . . 271
get_all_routemap_entry() . . . 271
Security Mode Module . . . 273
class SecModeApi . . . 273
get_current_security_mode . . . 273
get_new_security_setting . . . 273
set_security_mode . . . 274
System Module . . . 275
Top-Level System Functions . . . 275
client_connect() . . . 275
client_disconnect() . . . 275
class SystemInfo() . . . 276
get_systemInfo() . . . 276
get_env_fan() . . . 276
get_env_power() . . . 277
get_env_temperature() . . . 277
get_system_serial_num() . . . 278
get_system_inventory() . . . 278
get_hostname() . . . 279
set_hostname() . . . 279
get_system_core() . . . 279
TACACS+ Module . . . 281
class Tacacs . . . 281
get_feature_status . . . 281
set_feature_enabled . . . 281
set_feature_disabled . . . 281
get_global_key . . . 282
set_global_key . . . 282
get_host . . . 283
get_all_hosts . . . 283
set_host . . . 283
delete_host . . . 284
get_group . . . 285
get_all_groups . . . 285
add_group . . . 286
add_server_to_group . . . 286
set_group_vrf . . . 287
delete_group . . . 287

Telemetry Module . . . 289
class TelemetryBST_CancelRequest() . . . 289
set_bst_cancel_request () . . . 289
class TelemetryBST_Tracking() . . . 290
set_bst_tracking() . . . 290
get_bst_tracking() . . . 291
class TelemetryBST_Feature() . . . 292
set_bst_feature() . . . 292
get_bst_feature() . . . 293
class TelemetryBST_ClearCgsnDrop . . . 294
get_bst_clear_thresholds() . . . 294
class TelemetryBST_Cgsn_Drop_Ctr() . . . 295
get_bst_cgsn_drop_ctr() . . . 295
class TelemetryBST_ClearStats() . . . 296
get_bst_clear_stats() . . . 296
class TelemetryBST_ClearThresholds() . . . 297
get_bst_clear_thresholds() . . . 297
class TelemetryBST_Limits() . . . 297
get_bst_limits() . . . 297
class TelemetryBST_Report() . . . 298
get_bst_report() . . . 298
class TelemetryBST_Threshold() . . . 300
set_bst_threshold() . . . 300
get_bst_threshold() . . . 303
class TelemetryDevice_Feature . . . 305
set_sys_feature() . . . 305
get_sys_feature() . . . 305
class Telemetry_DeviceProp() . . . 306
get_swprop() . . . 306
VLAN Module . . . 309
class VlanSystem() . . . 309
python_get_vlan() . . . 309
python_create_vlan() . . . 310
python_delete_vlan() . . . 310
python_update_vlan_name() . . . 311
python_update_vlan_admin_state() . . . 311
class VlanEthIf . . . 312
python_get_vlan_properties() . . . 312
python_update_vlan_properties() . . . 313
VRF Module . . . 315
class VRF . . . 315
get_vrf_entry() . . . 315

VRRP Module . . . 317
class VRRP() . . . 317
get_vrrp() . . . 317
get_vrrp_intf() . . . 318
get_vrrp_accept_mode() . . . 319
get_vrrp_advt_interval() . . . 320
get_vrrp_preempt_mode() . . . 321
get_vrrp_priority() . . . 322
set_vrrp_accept_mode() . . . 322
set_vrrp_advt_interval() . . . 323
set_vrrp_preempt_mode() . . . 323
set_vrrp_priority() . . . 324
set_vrrp_switch_back_delay() . . . 324
set_vrrp_oper_primary_ipaddr() . . . 325
set_vrrp_monitored_circuit() . . . 325
delete_vrrp_vr() . . . 326
set_vrrp_vr() . . . 326
VXLAN Module . . . 327
python_vxlan_interface_ena() . . . 327
python_vxlan_interface_dis() . . . 327
python_vxlan_interface_get() . . . 327
python_vxlan_get() . . . 328
python_vxlan_vni_cnt_get() . . . 328
python_vxlan_vp_get() . . . 329
python_vxlan_vp_cnt_get() . . . 329
python_vxlan_vp_cnt_del() . . . 330
python_vxlan_vn_cnt_del() . . . 330
python_vxlan_tunnel_get() . . . 330
python_vxlan_mac_address_get() . . . 331
python_vxlan_vni_get() . . . 331
python_vxlan_interface_vni_map_set() . . . 332
class VxlanCfg() . . . 332
set_tunnel_ip() . . . 332
del_tunnel_ip() . . . 333
set_remote_vtep() . . . 333

**Appendix A. Error Messages . . . 335**

**Appendix B. Getting help and technical assistance . . . 337**

**Appendix C. Notices . . . 339**
Trademarks . . . 341
Important Notes . . . 342
Recycling Information . . . 343
Particulate Contamination . . . 344
Telecommunication Regulatory Statement . . . 345
Electronic Emission Notices . . . 346
Federal Communications Commission (FCC) Statement . . . 346
Industry Canada Class A Emission Compliance Statement . . . 346
Avis de Conformité à la Réglementation d'Industrie Canada . . . 346
Australia and New Zealand Class A Statement . . . 346
European Union - Compliance to the Electromagnetic Compatibility Directive

347
Germany Class A Statement . . . . . 347
Japan VCCI Class A Statement . . . . . 348
Japan Electronics and Information Technology Industries Association (JEITA) Statement . . . . . 349
Korea Communications Commission (KCC) Statement. . . . . 349
Russia Electromagnetic Interference (EMI) Class A statement . . . . . 349
People's Republic of China Class A electronic emission statement . . . . 349
Taiwan Class A compliance statement . . . . . 349

# Preface

The *Lenovo Network Python Programming Guide for Cloud NOS 10.8* describes how to configure and use the Lenovo Cloud Network Operating System 10.8 software on the following Lenovo RackSwitches:

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

# Who Should Use This Guide

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

# Additional References

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

- *Lenovo Network Application Guide for Lenovo Cloud Network Operating System 10.8*
- *Lenovo Network Command Reference for Lenovo Cloud Network Operating System 10.8*
- *Lenovo Network Release Notes for Lenovo Cloud network Operating System 10.8*
- *Lenovo Network REST API Programming Guide for Lenovo Cloud Network Operating System 10.8*

# 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:<br>`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.<br><br>It also depicts on-screen computer output and prompts. | View the `readme.txt` file.<br><br>`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.<br><br>This also shows book titles, special terms, or words to be emphasized. | To establish a Telnet session, enter:<br>`Switch#` **`telnet`** *<IP address>*<br><br>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 10.8 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 10.8.

# About CNOS Python

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

CNOS Python comes with the following features:

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

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

# Chapter 2. Running Python Scripts via the ISCLI

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

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

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

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

## Running a Basic Script

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

```
Switch# show script helloWorld.py

print "Hello world!"

Switch# python helloWorld.py

Hello world!
```

## Running a Basic Script with Arguments

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

```
Switch# show script scriptWithArgs.py

import sys

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

Switch# python scriptWithArgs.py 2 secondArgument 3rdArgument

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

# Entering and Exiting the Python Shell

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

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

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

```
Switch# python

>>>
```

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

```
exit ()

Switch#
```

# Chapter 3. Managing Python Scripts

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

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

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

# Downloading a Script from a TFTP Server

To download a TFTP script, use the following command:

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

where:

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

# Uploading a Script to a TFTP Server

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

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

where:

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

## Editing a Script Directly on the Switch

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

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

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

## Deleting a Script

To delete a script, use the following command:

```
Switch# no script 〈script filename〉
```

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

To delete all Python scripts, use the following command:

```
Switch# no script all
```

## Viewing A List of Script Files

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

```
Switch# show script
```

## Viewing Script File Content

To view a script file, use the following command:

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

## Viewing Configured Scheduler Jobs

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

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

Running jobs will be displayed in the running configuration display.

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

# Chapter 4. Using the CNOS Python Scheduler

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

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

# 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:<br>• Jan • Apr • Jul • Oct<br>• Feb • May • Aug • Nov<br>• Mar • Jun • Sep • 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:<br>• Mon • Wed • Fri • Sun<br>• Tue • Thu • Sat |

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

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

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

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

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

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

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

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

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

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

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

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

## Deleting a Job

To delete a job, use the following command:

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

## Monitoring a Running Job

To monitor a running job, use the following command:

```
Switch# show script running
```

## Stopping a Running Job

To stop a running job, use the following steps:

1. Check the list of running scripts:

```
Switch# show script running

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

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

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

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

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

## Viewing Python Logs

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

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

To view an individual log file, enter:

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

# Creating Syslog Messages

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

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

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

For example:

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

# Chapter 5. Writing Python Scripts

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

# Script Components

The CNOS Python API contains the following modules:

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

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

- `aaaApi`: Functions that manage the Authentication, Authorization and Accounting (AAA) switch configuration.
- `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.
- `dot1qEncapsApi`: Functions that manage Dot1Q encapsulation.
- `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.
- `nhopHealthcheckApi`: Functions that manage Nexthop health check.
- `ntpApi`: Functions that manage Network Type Protocol (NTP) authentication.
- `nwvApi`: Functions that manage Network Virtualization (NWV) authentication.

- `ospfApi`: Functions that manage the Open Shortest Path First (OSPF) switch configuration.
- `platformApi`: Functions that manage the switch port configuration.
- `pvlanApi`: Functions that manage Private VLAN properties.
- `routeApi`: Functions that manage the switch static route configuration.
- `routemapApi`: Functions that manage routemaps.
- `secModeApi`: Functions that manage the switch security mode configuration.
- `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).
- `vxlanApi`: Functions that manage Virtual Extensible LAN (VXLAN) properties.
- `weightedEcmpApi`: Functions that manage the Equal Cost Multiple Paths (ECMP) switch configuration.

The following is a sample Python shell session:

```
Switch# python

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

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

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

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

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

Switch#
```

## Viewing Online Help

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

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

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

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

Help on method python_lldp_get_all_neighbor in module lldpApi:

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

## Python API Function Arguments

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

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

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

All Python API functions use keyword arguments.

# Script Examples

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

## ARP Configuration Example

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

```
import sys

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

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

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

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

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

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

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

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

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

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

## IP Configuration Example

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

```
#Import modules
import systemApi, ipApi

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

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

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

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

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

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

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

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

## LAG Configuration Example

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

```
#Import modules
import systemApi, lagApi

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

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

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

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

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

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

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

## LLDP Configuration Example

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

```
#Import modules
import systemApi, lldpApi

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

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

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

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

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

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

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

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

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

## VLAN Configuration Example

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

```
#Import modules
import systemApi, vlanApi

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

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

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

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

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

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

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

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

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

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

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

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

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

# Chapter 6. The CNOS Python API

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


- "AAA Module" on page 55
- "ARP Module" on page 63
- "BGP Module" on page 69
- "Boot Information Module" on page 105
- "CEE Module" on page 109
- "DHCP Module" on page 123
- "DNS Module" on page 139
- "Dot1QEncaps Module" on page 145
- "ECMP Module" on page 147
- "FDB Module" on page 151
- "HostpCpy Module" on page 159
- "IGMP Module" on page 163
- "IP Module" on page 167
- "LACP Module" on page 173
- "LAG Module" on page 175
- "LLDP Module" on page 181
- "MSTP Module" on page 187
- "Nexthop Health Check" on page 193
- "NTP Authentication Module" on page 195
- "NWV Authentication Module" on page 197
- "OSPF Module" on page 199
- "Platform Module" on page 243
- "Private VLAN Module" on page 251
- "RADIUS Module" on page 257
- "Route Module" on page 267
- "Routemap Module" on page 271
- "Security Mode Module" on page 273
- "System Module" on page 275
- "TACACS+ Module" on page 281
- "Telemetry Module" on page 289
- "VLAN Module" on page 309
- "VRF Module" on page 315

- "VRRP Module" on page 317
- "VXLAN Module" on page 327

# 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 hostpAAAApi
```

## 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`).

### *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); one of `group`, `local`. |
| *groups* | (Optional) The list of AAA groups (string). Up to eight AAA groups can be configured, separated by space. List of groups can be followed by `local`.<br>**Note:** To configure the list of AAA groups, the variable *input_str* must be set to `group`. |

#### Returns

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

## *get_authorization_cmds_def*

Displays the current User EXEC command mode authorization configuration.

### Syntax

```
get_authorization_cmds_def()
```

### Returns

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

## *set_authorization_cmds_def*

Enables or disables User EXEC command mode authorization.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *input_str* | The AAA method type (string); one of `group`, `local`. |
| *groups* | (Optional) The list of AAA groups (string). Up to eight AAA groups can be configured, separated by space. List of groups can be followed by `local`.<br>**Note:** To configure the list of AAA groups, the variable *input_str* must be set to `group`. |

### Returns

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

## *get_authorization_conf_cmds_def*

Displays the current configuration command mode authorization configuration.

### Syntax

```
get_authorization_conf_cmds_def()
```

### Returns

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

## *set_authorization_conf_cmds_def*

Enables or disables configuration command mode authorization.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *input_str* | The AAA method type (string); one of `group`, `local`. |
| *groups* | (Optional) The list of AAA groups (string). Up to eight AAA groups can be configured, separated by space. List of groups can be followed by `local`.<br>**Note:** To configure the list of AAA groups, the variable *input_str* must be set to `group`. |

### Returns

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

## *get_authentication_login_con*

Displays the current console user login authentication configuration.

### Syntax

```
get_authentication_login_con()
```

### Returns

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

## *set_authentication_login_con*

Enables or disables console user login authentication.

### Syntax

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

where:

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

### Returns

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

## *get_authentication_login_def*

Displays the current remote user login authentication configuration.

### Syntax

```
get_authentication_login_def()
```

### Returns

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

## *set_authentication_login_def*

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

### Syntax

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

where:

| Variable | Description |
|---|---|
| *input_str* | The AAA method type (string); one of `group`, `local`, or `none`. |
| *groups* | (Optional) The list of AAA groups (string). Up to eight AAA groups can be configured, separated by space. List of groups can be followed by `local` or `none`.<br>**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; an integer from 1-25. |

### Returns

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

## *get_user_default_role*

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

### Syntax

```
get_user_default_role()
```

### Returns

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

- `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); one of `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()*

Gets 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`.<br>**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 (string), in the following format: xxxx.xxxx,xxxx. |

## *get_one_static_arp_entry()*

Gets one static ARP entry for the specified interface.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The Ethernet interface name (string).<br>**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 (string), in the following format: xxxx.xxxx,xxxx. |

## *set_ip_arp()*

Creates a static proxy ARP entry.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *ip_addr* | The IP address (string). |
| *mac_addr* | The MAC address (string), in the following format: xxxx.xxxx,xxxx. |
| *if_name* | The Ethernet interface name (string).<br>**Note:** The interface must exist. |

### Returns

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

## *delete_ip_arp()*

Deletes an ARP entry.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *ip_addr* | The IP address (string). |
| *if_name* | The Ethernet interface name (string).<br>**Note:** The interface must exist. |

### Returns

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

## *get_arp_sys_pro()*

Gets the global ARP properties of the system.

### Syntax

`get_arp_sys_pro()`

### Returns

The global ARP entry age out time:

| 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()*

Sets 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()*

Gets 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`.<br>**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()*

Sets 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`.<br>**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);<br>one of `enabled`, `disabled`. |

### Returns

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

# BGP Module

The class in this module manages Border Gateway Protocol (BGP) configurations. To use this module, in the Python file or in the Python interpreter, enter:

```
import bgpApi
```

## class BGP()

The functions in this class get and set BGP configurations.

### *python_bgp_get_global_statistics()*

Gets 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 message number; a positive integer. |
| *bytes_in* | Bytes received; a positive integer. |
| *bytes_out* | Bytes sent; a positive integer. |
| *open_in* | Open message input count; a positive integer. |
| *open_out* | Open message output count; a positive integer. |
| *update_in* | Update message input count; a positive integer. |
| *update_out* | Update message ouput count; a positive integer. |
| *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()*

Gets 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()*

Shows 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:<br>• `1` - adjacent routes<br>• `0` - advertised adjacent routes |
| *neighbor_ip* | Neighbor IP address; a valid IPv4 or IPv6 address. |
| *vrf_name* | (Optional) Address family name; one of `ipv4`, `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:<br>• `i` - IGP<br>• `e` - EGP<br>• `?` - 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:<br>• `s` - suppressed<br>• `d` - damped<br>• `h` - history<br>• `*` - valid<br>• `>` - best<br>• `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()*

Shows 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()*

Gets 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()*

Gets the BGP AS number.

### Syntax

`python_bgp_get_as_number()`

### Returns

The BGP AS number; an integer from 0-4294967295. Default value: 0.

## *python_bgp_get_hold_down_timer()*

Gets the BGP hold down interval.

### Syntax

`python_bgp_get_hold_down_timer(`*<vrf_name>*`)`

where:

| 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()*

Gets 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()*

Shows 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()*

Shows 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()*

Shows 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()*

Gets 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()*

Gets 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()*

Shows 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()*

Gets 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()*

Gets 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()*

Gets 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()*

Shows 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()*

Gets 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()*

Gets 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()*

Gets 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`, or `L2VPN`. 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:<br>• `s` - suppressed<br>• `d` - damped<br>• `h` - history<br>• `*` - valid<br>• `>` - best<br>• `i` – internal |

| Element | Description |
|---|---|
| *network* | Route destination IP address; a valid IPv4 or IPv6 address. For L2VPN, the network field displays:<br>• `EVPN type-1 prefix: [1]:[ESI]:[EthTag]`<br>• `EVPN type-2 prefix: [2]:[ESI]:[EthTag]:[MAClen]:[MAC]` |
| *nextHopGlobal* | Route nexthop IPv6 address. Not used for IPv4. |
| *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:<br>• `i` - IGP<br>• `e` - EGP<br>• `?` - incomplete |

## *python_show_ip_bgp_network()*

Gets 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:<br>• IPv4: An integer from 0-32.<br>• 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`.<br>**Note:** This value only appears if it has been set. |
| *suppressed (damp)* | Suppressed due to dampening; `Yes`.<br>**Note:** This value only appears if it has been set. |
| *history entry* | History entry; `Yes`.<br>**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.<br>**Note:** This value only appears if it is `No`. |
| *from peer* | Whether the from peer address can be accessed, `No`.<br>**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`.<br>**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.<br>**Note:** This value only appears if it is set. |
| *label* | Label; only appears if set. |
| *valid* | Whether the path is valid; `Yes`.<br>**Note:** This value only appears if the path is valid. |

| Element | Description |
|---|---|
| *stale* | Whether the state is stale; `Yes`.<br>**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`.<br>**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 advertised to peer groups. |
| *not advertised* | Not advertised to any peer.<br>**Note:** This value only appears if `true`. |

## *python_show_l2vpn_bgp_network()*

Shows the BGP routing table.

### Syntax

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

where:

| Element | Description |
|---|---|
| *keyword* | (Mandatory) The route type (string); one of `mac`, `esi`. |
| *value* | (Mandatory) A mac or an Ethernet segment ID; a string with an ESI value, or a MAC value in the following format: `EEEE.EEEE.EEEE`. |
| *vni* | (Mandatory) The Virtual Network Identifier; string representing an integer from 1-16777214. |
| *rd* | (Optional) The route distinguisher; a string in the following format: `IP_address:nn`. |

### Returns

A dictionary containing the route list of BGP for a certain address family:

| Element | Description |
|---|---|
| *table entry for* | Route IP address/mask; a valid IP address and net mask. |
| *paths* | The number of paths to destination. |
| *best* | Which path number is best; the maximum number of paths for this destination. |
| *no advertise* | Whether the route is advertised to any peers; one of `yes`, `no`. |
| *no export* | Whether the route is advertised to any EBGP peers; one of `yes`, `no`. |
| *local as* | The local AS number; an integer from 1-4294967295. |
| *suppress* | Whether the advertisements are suppressed by an aggregate; one of `yes`, `no`. |
| *adv non peer-group* | The non peer-group name. |
| *adv peer-group* | The peer-group name. |
| *no peer adv* | Not advertised to any peer. |
| *as path str* | Route AS path attribute; an AS path VTY string. |
| *aggregator as* | Aggregator AS number. |
| *aggregator as4* | Aggregator 4-byte AS number. |

| Element | Description |
|---|---|
| *aggregator address* | Aggregator address. |
| *rec from RR-client* | Received from route reflector client; one of `yes`, `no`. |
| *suppressed (damp)* | Suppressed due to dampening; one of `yes`, `no`. |
| *history entry* | History entry; one of `yes`, `no`. |
| *nexthop address* | Route next-hop; a valid IP address. |
| *peer* | Peer address. |
| *inaccessible* | Whether the RIB is can be accessed; one of `yes`, `no`. |
| *igpmetric* | The IGP metric value. |
| *from peer* | The peer address. |
| *orig id* | The originator ID. |
| *next-hop local ip* | The next-hop IP address; a valid IP address. |
| *metric removed* | Whether one of the metric values is removed; one of `yes`, `no`. |
| *local pref* | Local preference value. |
| *weight* | Route weight attribute; an integer from 0-65535. |
| *label* | The label. |
| *valid* | Whether the paths is valid; one of `yes`, `no`. |
| *stale* | Whether the state is stale; one of `yes`, `no`. |
| *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; one of `yes`, `no`. |
| *best* | Whether this is the best path; one of `yes`, `no`. |
| *community* | The community string. |
| *extended community* | The extended community string. |
| *originator* | The originator ID. |
| *cluster id* | The cluster ID. |
| *reuse info* | Reuse information. |
| *last update* | Last update time. |

## *python_show_ip_bgp_summary()*

Gets IP, IPv6, L2VPN BGP summary information.

### Syntax

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

where:

| Element | Description |
|---|---|
| *af_name* | (Optional) Address family name; one of `ipv4`, `ipv6` or `l2vpn`. 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`, `epvn`. Default value: `unicast`. |

### Returns

A dictionary containing BGP route information:

| Element | Description |
|---|---|
| *router id* | Router ID; a valid IPv4, IPv6 or L2VPN address. |
| *table version* | The table version. |
| *path count* | The number of paths. |
| *conf max ebgp paths* | The maximum number of configured EBGP paths. |
| *max ebgp paths* | The maximum number of EBGP paths. |
| *conf max ibgp paths* | The maximum number of configured IBGP paths |
| *max ibgp paths* | The maximum number of IBGP paths. |
| *local AS count* | The local AS count. |
| *peer* | Peer address; a valid IPv4, IPv6 or L2VPN 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 input count. |

| Element | Description |
|---|---|
| *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_neighbors()*

Gets BGP neighbor details.

### Syntax

`python_show_ip_bgp_neighbors(`*<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`. |

| Element | Description |
|---|---|
| *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. |
| *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. |

| Element | Description |
|---|---|
| *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. |
| *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()*

Gets 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()*

Gets 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`.<br>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;<br>one of `enable`, `disable`. |

## *python_bgp_get_af_maximum_paths_config()*

Gets BGP multipath ECMP number configuration information.

### Syntax

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

where:

| 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`.<br>Default value: `default`. |

### Returns

A dictionary containing BGP global configuration information:

| Element | Description |
|---|---|
| *vrf_name* | VRF name; one of the VRF name, `default`, `all`.<br>Default value: `default`. |
| *ibgp_max_number* | IBGP multipath maximum ECMP number;<br>an integer from 1-32. |
| *ebgp_max_number* | EBGP multipath maximum ECMP number;<br>an integer from 1-32. |

## *python_bgp_get_af_nexthop_trigger_delay_config()*

Gets the BGP nexthop trigger-delay configuration.

### Syntax

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

where:

| 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;<br>an integer from 1-4294967295. |
| *non-critical* | Nexthop changes affecting metric;<br>an integer from 1-4294967295. |

## *python_bgp_get_af_aggregate_config()*

Gets the BGP aggregate configuration.

### Syntax

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

where:

| 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`.<br>Default value: `default`. |

### Returns

A dictionary containing BGP aggregate information:

| Element | Description |
|---|---|
| *vrf_name* | VRF name; one of the VRF name, `default`.<br>Default value: `default`. |
| *prefix* | Aggregate prefix; an IP address in one of the following forms:<br>• A.B.C.D/M<br>• X:X::X:X/M. |
| *type* | Aggregate type; one of the following:<br>• `as_set` - Generate AS set path information.<br>• `summary_only` - Filter more specific routes from updates.<br>• `as_set_summary_only` - Both `as-set` and `summary-only`. |

## *python_bgp_get_af_dampening_config()*

Gets the BGP dampening configuration.

### Syntax

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

`where:`

| 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`.<br>Default value: `default`. |

### Returns

A dictionary containing BGP dampening information:

| Element | Description |
|---|---|
| *vrf_name* | VRF name; one of the VRF name, `default`, `all`.<br>Default value: `default`. |

| Element | Description |
|---|---|
| *prefix* | Aggregate prefix; an IP address in one of the following forms:<br>• A.B.C.D/M<br>• X:X::X:X/M. |
| *type* | Aggregate type; one of the following:<br>• `as_set` - Generate AS set path information.<br>• `summary_only` - Filter more specific routes from updates.<br>• `as_set_summary_only` - Both `as-set` and `summary-only`. |

## *python_bgp_get_af_network_config()*

Gets the BGP network configuration.

### Syntax

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

where:

| 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:<br>• A.B.C.D/M<br>• 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()*

Gets 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` or `l2vpn`. |
| *saf_name* | Subsequent Address Family Identifier name; one of `unicast`, `evpn`. |
| *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_bgp_put_af_redistribute_config()*

Configures the BGP redistribute configuration.

### Syntax

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

where:

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

### Returns

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

## *python_show_ip_bgp_neighbor_stats()*

Gets BGP neighbor details.

### Syntax

`python_show_ip_bgp_neighbor_stats(`*<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; an integer from 1 - 4294967295. |
| *local as* | The current local AS number; an integer from 1 - 4294967295. |
| *address family* | The BGP neighbor address family (integer); one of `ipv4`, `ipv6`. |
| *advertisement interval* | The configured minimum time interval, in seconds, between consecutive BGP updates; an integer from 1-65535. |
| *bfd* | The status of BFD (string); one of `enabled`, `disabled`, or `multihop enabled`. |
| *connection retry time* | The connection retry time, in seconds; an integer from 1-65535. |
| *description* | The BGP neighbor description (string). |
| *disallow infinite holdtime* | Whether the configuration of infinite hold-time is disallowed (string); one of `yes`, `no`. |
| *do not capability negotiate* | Whether capability negotiations are disabled (string); one of `yes`, `no`. |
| *advertise dynamic capability* | Whether dynamic capability advertisements are enabled (string); one of `yes`, `no`. |
| *EBGP multihop* | The number of EBGP multi-hops; an integer from 1-255. |
| *remote private as* | Whether the removal of private AS numbers from outbound routes updates is enabled (string); one of `yes`, `no`. |
| *maximum peers* | The maximum number of peers configured for the prefix of the BGP neighbor; a string from 1-96. |
| *password* | The encrypted password for the BGP neighbor (string). |
| *shutdown* | Whether the BGP neighbor is shut down (string); one of `yes`, `no`. |
| *peer holdtime* | The time interval, in seconds, the switch awaits before transitioning the BGP neighbor to IDLE state, if the switch doesn't receive an update or keep-alive message from the neighbor; an integer from 0-3600. |

| Element | Description |
|---|---|
| *peer keepalive* | The time interval, in seconds, the switch awaits before sending another keep-alive message to the BGP neighbor; an integer from 0-3600. |
| *connection-mode passive* | Whether the initiations of TCP sessions with the BGP neighbor are disabled (string).<br>Displays whether the BGP neighbor is shut down; one of `yes`, `no`. |
| *ttl security hops* | The minimum number of TTL router hops an IP packet must have to not be discarded; an integer from 1-254. |
| *update-source* | The source of the BGP session and updates - string containing ethernet port, VLAN, and loopback interfaces information. |
| *weight* | The default weight of routes incoming from the BGP neighbor; an integer from 1-65535. |
| *allow as in* | Whether AS paths with the local AS number are accepted by the switch (string); one of `yes`, `no`. |
| *default originate* | Whether a default route to the BGP neighbor is configured (string); one of `yes`, `no`. |
| *default originate map* | The name of the route map for the default route (string). |
| *prefix-list in* | The prefix list for routes incoming from the BGP neighbor (string). |
| *prefix-list out* | The prefix list for routes outgoing to the BGP neighbor (string). |
| *maximum-prefix* | The maximum number of prefixes that can be received from the BGP neighbor; an integer from 1-15872. |
| *maximum-prefix warning* | Whether warning messages are generated only when the maximum prefix limit is exceeded (string); one of `yes`, `no`. |
| *maximum-prefix threshold percent* | The percentage of the maximum prefix limit at which the switch starts to generate a warning message; an integer from 1-100. |
| *next-hop-self* | Whether next-hop calculations for the BGP neighbor are disabled (string); one of `yes`, `no`. |
| *filter-list in* | The AS path ACL for routes incoming from the BGP neighbor (string). |
| *filter-list out* | The AS path ACL for routes outgoing to the BGP neighbor (string). |
| *route-map in* | The applied route map for routes incoming from the BGP neighbor (string). |

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

## *python_put_ip_bgp_neighbors_cfg*

Configures a BGP neighbor.

### Syntax

`python_put_ip_bgp_neighbors_cfg(`<*ip_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 |
|---|---|
| *address family* | The neighbor address family (string); one of `ipv4 unicast`, `ipv6 unicast`, or `l2vpn evpn`. |
| *remote as* | The current remote AS number; an integer from 1-4294967295. |
| *local as* | The current local AS number; an integer from 1-4294967295. |
| *advertisement interval* | The configured minimum time interval, in seconds, between consecutive BGP updates; an integer from 0-65535. |
| *bfd* | The status of BFD (string); one of `enabled`, `disabled`, or `multihop enabled` |
| *connection retry time* | The connection retry time, in seconds; an integer from 0-65535. |
| *description* | The BGP neighbor description (string). |
| *disallow infinite holdtime* | Whether the configuration of infinite hold-time is disallowed (string); one of `yes`, `no.` |
| *do not capability negotiate* | Whether capability negotiations are disabled (string); one of `yes`, `no`. |
| *advertise dynamic capability* | Whether dynamic capability advertisements are enabled (string); one of `yes`, `no`. |
| *EBGP multihop* | The number of EBGP multi-hops; an integer from 1-255. |

| Element | Description |
|---|---|
| *remote private as* | Whether the removal of private AS numbers from outbound routes updates is enabled (string); one of `yes`, `no`. |
| *maximum peers* | The maximum number of peers configured for the prefix of the BGP neighbor; a string from 1-96. |
| *password* | The encrypted password for the BGP neighbor (string). |
| *unencrypt-password* | Whether the password is unencrypted (string); one of `yes`, `no`. |
| *shutdown* | Whether the BGP neighbor is shut down (string); one of `yes`, `no`. |
| *peer holdtime* | The time interval, in seconds, the switch awaits before transitioning the BGP neighbor to IDLE state, if the switch doesn't receive an update or keep-alive message from the neighbor; an integer from 0-3600. |
| *peer keepalive* | The time interval, in seconds, the switch awaits before sending another keep-alive message to the BGP neighbor; an integer from 0-3600. |
| *connection-mode passive* | Whether the initiations of TCP sessions with the BGP neighbor are disabled (string).<br>Displays whether the BGP neighbor is shut down; one of `yes`, `no`. |
| *ttl security hops* | The minimum number of TTL router hops an IP packet must have to not be discarded; an integer from 1-254. |
| *update-source* | The source of the BGP session and updates - string containing ethernet port, VLAN, and loopback interfaces information. |
| *weight* | The default weight of routes incoming from the BGP neighbor; an integer from 0-65535. |
| *allow as in* | Whether AS paths with the local AS number are accepted by the switch (string); an integer from 1-10. |
| *default originate* | Whether a default route to the BGP neighbor is configured (string); one of `yes`, `no`. |
| *default originate map* | The name of the route map for the default route (string). |
| *prefix-list in* | The prefix list for routes incoming from the BGP neighbor (string). |
| *prefix-list out* | The prefix list for routes outgoing to the BGP neighbor (string). |
| *maximum-prefix* | The maximum number of prefixes that can be received from the BGP neighbor; an integer from 1-15872. |

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

### Returns

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

## *python_bgp_set_unnumbered*

Globally enables BGP unnumbered on the switch.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *as_number* | The BGP AS number; an integer from 1 - 4294967295. |
| *set_bfd* | (Optional) BFD option (string); one of `enabled`, `disabled`. Default value: `disabled.` |

### 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; an integer from 1 - 4294967295. |

### Returns

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

## *python_bgp_get_dscp()*

Gets the BGP DSCP marking value.

### Syntax

`python_bgp_get_dscp(`*<vrf_name>*`)`

where:

| Element | Description |
|---|---|
| *vrf_name* | (Optional) VRF name (string); one of `data`, `default`. Default value: `default`. |

### Returns

The DSCP marking value; an integer from 0-63.

## *python_bgp_put_dscp()*

Sets the BGP DSCP marking value.

### Syntax

`python_bgp_put_dscp(`*<dscp_value>*`,`*<vrf_name>*`)`

where:

| Element | Description |
|---|---|
| *dscp_value* | (Mandatory) The DSCP marking value; an integer from 0-63. |
| *vrf_name* | (Optional) VRF name (string); one of `data`, `default`.<br>Default value: `default`. |

### 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()*

Gets 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()*

Gets 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`.<br>Default value: `Enable`. |

## *set_boot_ztp()*

Sets 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()*

Gets 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()*

Sets 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 classes in this module manage 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).<br>For example: *Ethernet1/12*. |

#### Returns

A dictionary showing DCBX interface information:

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

### Returns

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

## *python_dcbx_pfc_set_advt*

Configures PFC local configuration advertisement on a switch interface.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The name of the switch interface (string).<br>For example: *Ethernet1/12*. |
| *enadis_string* | The status of the PFC local configuration advertisement (string); one of `on`, `off`. |

### Returns

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

## *python_dcbx_ets_set_advt*

Configures ETS local configuration advertisement on a switch interface.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The name of the switch interface (string).<br>For example: *Ethernet1/12*. |
| *enadis_string* | The status of the ETS local configuration advertisement (string); one of `on`, `off`. |

### Returns

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

## *python_dcbx_app_set_advt*

Configures application protocol local configuration advertisement on a switch interface.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The name of the switch interface (string).<br>For example: *Ethernet1/12*. |
| *enadis_string* | The status of the application protocol local configuration advertisement (string); one of `on`, `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).<br>For example: *Ethernet1/12*. |

### Returns

A dictionary showing DCBX control state machine interface information:

| Element | Description |
|---|---|
| *dcbx version* | The version of DCBX (string); one of `DCBX IEEE 802.1Qaz (v2.5)` or `DCBX CEE (v1.01)`. |
| *if_name* | The name of the switch interface (string).<br>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).<br>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).<br>For example: *Ethernet1/12*. |
| *cfgtype* | The type of PFC configuration (string);<br>one of `admin`, `operation`, or `remote`. |

### Returns

A dictionary showing PFC interface configuration:

| Element | Description |
|---|---|
| *state* | The status of PFC for the interface (string);<br>one of `on`, `off`. |
| *willing* | Whether the switch is "willing" to learn PFC configurations from a DCBX peer (string);<br>one of `on`, `off`. |
| *advt* | The status of PFC local configuration advertisement (string); one of `on`, `off`. |
| *syncd* | The status of PFC information synchronization (string);<br>one of `on`, `off`. |
| *priority_map* | The priorities enabled on the interface (string). |

## *python_dcbx_ets_get_interface*

Displays the ETS configuration for a switch interface.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The name of the switch interface (string).<br>For example: *Ethernet1/12*. |
| *cfgtype* | The type of PFC configuration (string);<br>one of `admin`, `operation`, or `remote`. |

### Returns

A dictionary showing PFC interface configuration:

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

## *python_dcbx_app_get_interface*

Displays the application protocol configuration for a switch interface.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The name of the switch interface (string).<br>For example: *Ethernet1/12*. |
| *cfgtype* | The type of application protocol configuration (string); one of `admin`, `operation`, or `remote`. |

### Returns

A dictionary showing application protocol interface configuration:

| Element | Description |
|---|---|
| *state* | The status of application protocol for the interface (string); one of `on`, `off`. |
| *willing* | Whether the switch is "willing" to learn application protocol configurations from a DCBX peer (string); one of `on`, `off`. |
| *advt* | The status of application protocol local configuration advertisement (string); one of `on`, `off`. |

## *python_dcbx_app_get_protocol_list_interface*

Displays the application protocol list for a switch interface.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The name of the switch interface (string).<br>For example: *Ethernet1/12*. |
| *cfgtype* | The type of application protocol configuration (string); one of `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;<br>an integer from 0-7. |
| *protocol* | The type of the protocol (string);<br>one of `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); one of `on`, `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); one of `on`, `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); one of `on`, `off`. |

#### Returns

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

## *python_cee_pfc_get_priority_map*

Displays the list of configured PFC priorities.

### Syntax

```
python_cee_pfc_get_priority_map()
```

### Returns

A string showing the enabled list of PFC priorities.

## *python_cee_pfc_set_priority_map*

Configures the PFC priority flow mapping.

**Note:** This function overwrites existing PFC priority configurations.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *new_priority_map* | The PFC priority flow map.<br>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).<br>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).<br>For example: *Ethernet1/12*. |
| *pfc_state* | The status of PFC on the specified interface (string); one of `on`, `off`. |

### Returns

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

## *python_cee_pfc_interface_get_counters*

Displays the PFC statistics for a switch interface.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The name of the switch interface (string).<br>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).<br>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; an integer from 0-100. |
| *pgid* | The ID of the priority group; an integer from 0-7, or 15. |
| *priority_pgid_mapping* | The priorities mapped to the priority group.<br>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; an integer from 0-7, or 15. |
| *pgid* | The ID of the priority group; an integer from 0-7, or 15. |
| *priority_pgid_mapping* | The priorities mapped to the priority group.<br>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;<br>an integer from 0-7. |
| *protocol* | The type of the protocol (string);<br>one of `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;<br>an integer from 0-7. |
| *protoid* | The name of the protocol (string). |
| *config_name* | The name of the application protocol configuration (string). |
| *protocol* | The type of the protocol (string);<br>one of `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_Client()

The functions in this class get and set DHCP configurations.

### *get_dhcp_feature()*

Determines whether the DHCP client is enabled or disabled.

#### Syntax

```
get_dhcp_feature()
```

#### Returns

Whether the DHCP client is enabled or disabled:

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

### *set_dhcp_feature()*

Enables the DHCP client.

#### Syntax

```
set_dhcp_feature()
```

#### Returns

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

### *unset_dhcp_feature()*

Disables the DHCP client.

#### Syntax

```
unset_dhcp_feature()
```

#### Returns

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

## *get_dhcpinfo()*

Gets the DHCP client configuration.

### Syntax

`get_dhcpinfo(`<*if_name*>`)`

where:

| 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`.<br>Default value: `no`. |
| *class_id* | The name of the vendor class-identifier; a string up to 64 characters long. |

## *set_dhcp_client()*

Enables the DHCP client on the specified interface.

### Syntax

`set_dhcp_client(<if_name>)`

where:

| Variable | Description |
|---|---|
| *if_name* | The interface port name (string). |

### Returns

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

## *unset_dhcp_client()*

Disables the DHCP client on the specified interface.

### Syntax

`unset_dhcp_client(<if_name>)`

where:

| Variable | Description |
|---|---|
| *if_name* | The interface port name (string). |

### Returns

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

## *set_dhcpv6_client()*

Enables the DHCPv6 client on the specified interface.

### Syntax

`set_dhcpv6_client(<if_name>)`

where:

| Variable | Description |
|---|---|
| *if_name* | The interface port name (string). |

### Returns

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

## *unset_dhcpv6_client()*

Disables the DHCPv6 client on the specified interface.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The interface port name (string). |

### Returns

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

## *set_dhcp_option_hostname()*

Enables the DHCP hostname option on the specified interface.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The interface port name (string). |

### Returns

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

## *unset_dhcp_option_hostname()*

Disables the DHCP hostname option on the specified interface.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The interface port name (string). |

### Returns

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

## *set_dhcp_option_ntp_server()*

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

### Syntax

`set_dhcp_option_ntp_server(<if_name>)`

where:

| Variable | Description |
|---|---|
| *if_name* | The interface port name (string). |

### Returns

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

## *unset_dhcp_option_ntp_server()*

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

### Syntax

`unset_dhcp_option_ntp_server(<if_name>)`

where:

| Variable | Description |
|---|---|
| *if_name* | The interface port name (string). |

### Returns

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

## *set_dhcp_option_log_server()*

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

### Syntax

`set_dhcp_option_log_server(<if_name>)`

where:

| Variable | Description |
|---|---|
| *if_name* | The interface port name (string). |

### Returns

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

## *unset_dhcp_option_log_server()*

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

### Syntax

`unset_dhcp_option_log_server(`<*if_name*>`)`

where:

| Variable | Description |
|---|---|
| *if_name* | The interface port name (string). |

### Returns

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

## *set_dhcp_class_id()*

Sets the specified vendor class ID on the specified interface.

### Syntax

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

where:

| 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()*

Removes 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()*

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

#### Syntax

```
get_relay_serv()
```

#### Returns

A dictionary showing whether DHCP relay is enabled or disabled:

| 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()*

Enables DHCPv4 relay service.

#### Syntax

```
set_dhcpv4_relay()
```

#### Returns

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

### *unset_dhcpv4_relay()*

Disables DHCPv4 relay service.

#### Syntax

```
unset_dhcpv4_relay()
```

#### Returns

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

## *set_dhcpv6_relay()*

Enables DHCPv6 relay service.

### Syntax

```
set_dhcpv6_relay()
```

### Returns

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

## *unset_dhcpv6_relay()*

Disables DHCPv6 relay service.

### Syntax

```
unset_dhcpv6_relay()
```

### Returns

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

## *get_interface_relay_address()*

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

### Syntax

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

where:

| 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);<br>a valid IPv4 address. |
| *dhcpv6_relay* | A dictionary containing DHCPv6 relay addresses. |

| Element | Description |
|---|---|
| *v6_relay_address* | DHCPv6 relay server address (string);<br>a valid IPv6 address. |
| *v6_relay_out_if* | DHCPv6 outgoing interface (string). |

## *set_relay4_address()*

Sets a DHCPv4 relay address for the specified interface.

### Syntax

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

where:

| 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()*

Removes 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()*

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

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The interface port name (string). |
| *ipv6_addr* | The IP address (string); a valid IPv6 address. |
| *out_if_name* | (Optional) Relay outgoing interface (string). |

### Returns

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

## *unset_relay6_address()*

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

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The interface port name (string). |
| *ipv6_addr* | The IP address (String); a valid IPv6 address. |
| *out_if_name* | (Optional) Relay outgoing interface (string). |

### Returns

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

## class DHCP_Snooping

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); one of `enable` or `disable`. |
| *option_82* | The status of DHCP Option 82 (string); one of `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); one of `enable` or `disable`. |
| *option_82* | The status of DHCP Option 82 (string); one of `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), in the following format: *XX:XX:XX:XX:XX:XX*. |
| *ip_addr* | The IP address of the binding entry (string). |
| *lease_time* | The lease time, in seconds, of the binding entry; an integer from 1-4294967295. |
| *type* | The type of the binding entry (string); one of `static`, `dynamic`. |
| *vlan* | The VLAN ID of the binding entry; an integer from 1-4093. |
| *if_name* | The name of the switch interface associated with the binding entry (string). For example: *Ethernet1/12*. |

## *python_dhcpsnp_post_binding_entry*

Adds entries to the DHCP Snooping binding table.

### Syntax

`python_dhcpsnp_post_binding_entry(`<*dict_dhcpsnp_entry*>`)`

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

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

### Returns

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

## *python_dhcpsnp_del_binding_entry*

Deletes DHCP Snooping binding table entries.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *mac_vlan_if* | (Optional) The binding entry identified by either its MAC address, VLAN ID, or interface name. Type - as follows:<br>• for MAC address (string); in the following format: *XX.XX.XX.XX.XX.XX*<br>• for VLAN ID - an integer from 1-4093<br>• for interface name - a string (for example, *Ethernet1/12*) |

### Returns

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

## *python_dhcpsnp_get_vlan*

Displays the VLANs for which DHCP Snooping is configured.

### Syntax

```
python_dhcpsnp_get_vlan()
```

### Returns

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

| 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; an integer from 1-4093. |

### Returns

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

## *python_dhcpsnp_get_statistics*

Displays DHCP Snooping statistics.

### Syntax

```
python_dhcpsnp_get_statistics()
```

### Returns

A dictionary showing DHCP Snooping statistics:

| 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).<br>For example: *Ethernet1/12*. |
| *trusted* | Whether the interface is trusted (string); one of `yes`, `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).<br>For example: *Ethernet1/12*. |
| *trusted* | Whether the interface is trusted (string); one of `yes`, `no`. |

### Returns

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

# DNS Module

The class 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 hostpDnsClntApi
```

## 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);<br>one of `enabled`, `disabled`. |
| *dynamic_domain* | Dynamic domain information (string). |
| *dynamic_nameserver* | Dynamic DNS information (string). |
| *domain_list* | Domain information.<br>Type - list of dictionaries containing string elements. |
| *nameserver_list* | DNS server information.<br>Type - list of dictionaries containing string elements. |
| *nametoip_list* | Hostname/IP address associations.<br>Type - list of dictionaries containing string elements (hostname to IP address mappings). |

# Dot1QEncaps Module

The class and functions in this module manage Dot1Q encapsulation. To use this module, in the Python file or in the Python interpreter, enter:

```
import dot1qEncapsApi
```

## class Dot1qEncapsulation()

This class provides functions that manage Dot1Q encapsulation.

### *set_dot1q_tag_interface()*

Enables Dot1Q tag on an interface.

#### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | (Mandatory) The interface name (string); |
| *tag* | (Mandatory) The Dot1Q tag value;<br>an integer from 1-4093. |

#### Returns

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

### *unset_dot1q_tag_interface()*

Disables Dot1Q tag on an interface.

#### Syntax

`unset_dot1q_tag_interface(<`*if_name*`>)`

where:

| Variable | Description |
|---|---|
| *if_name* | (Mandatory) The interface name (string); |

#### Returns

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

## *get_dot1q_tag_interface()*

Gets Dot1Q tag on an interface.

### Syntax

`get_dot1q_tag_interface(`<*if_name*>`)`

where:

| Variable | Description |
|---|---|
| *if_name* | (Mandatory) The interface name (string). |

### Returns

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

# 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);<br>one of `enable`, `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);<br>one of `enable`, `disable`. |

#### Returns

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

## class WeightedEcmp_ipv4

The function in this class sets ECMP weight for IPv4 nexthops.

### *set_weighted_ecmp_nexthop_ipv4*

Configures ECMP weight for the specified IPv4 nexthop.

#### Syntax

`set_weighted_ecmp_nexthop_ipv4(`*<addr>*`,`*<weight>*`)`

where:

| Variable | Description |
|---|---|
| *addr* | (Mandatory) The IPv4 address of the nexthop (string). |
| *weight* | (Mandatory) The ECMP weight of the specified nexthop (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 nexthop.

### *get_weighted_ecmp_ipv4*

Gets ECMP weight for an IPv4 nexthop.

#### Syntax

`get_weighted_ecmp_ipv4(`*<addr>*`)`

where:

| Variable | Description |
|---|---|
| *addr* | (Mandatory) The IPv4 address of the nexthop (string). |

#### Returns

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

## class WeightedEcmp_ipv6

The function in this class sets ECMP weight for IPv6 nexthops.

### *set_weighted_ecmp_nexthop_ipv6*

Sets ECMP weight for the specified IPv6 nexthop.

#### Syntax

`set_weighted_ecmp_nexthop_ipv6(`*<addr>*`,`*<weight>*`)`

where:

| Variable | Description |
|---|---|
| *addr* | (Mandatory) The IPv6 address of the nexthop (string). |
| *weight* | (Mandatory) The ECMP weight of the specified nexthop;<br>an integer from 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 nexthop.

### *get_weighted_ecmp_ipv6*

Get ECMP weight for a specified IPv6 nexthop.

#### Syntax

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

where:

| Variable | Description |
|---|---|
| *addr* | (Mandatory) The IPv6 address of the nexthop (string). |

#### Returns

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

## 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).<br>For example: *Ethernet1/12*. |
| *weight* | The ECMP weight of the specified interface;<br>an integer from 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).<br>For example: *Ethernet1/12*. |

#### Returns

A dictionary showing ECMP weight information:

| Element | Description |
|---|---|
| *interface_name* | The name of the switch interface (string).<br>For example: *Ethernet1/12*. |
| *interface_weight* | The ECMP weight of the specified interface;<br>an integer from 1-4. |

# FDB Module

The class 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);<br>one of `static`, `multicast`, or `dynamic`. |
| *mac_address* | (Optional) The MAC address to filter on (string). |
| *interfaces* | (Optional) The list of switch interfaces to filter on.<br>Type - list containing interface names, which are string (for example, *Ethernet1/12*). |
| *vlan_id* | (Optional) The VLANs to filter on;<br>an integer from 1-4094. |

#### Returns

Multiple possible return values:

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

| Element | Description |
|---|---|
| *address_table* | The list of MAC addresses matching the search filter.<br>Type - list with MAC addresses as string. |
| *is_static* | Whether the MAC address is statically configured or dynamically learned (boolean).<br>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).<br>For example: *Ethernet1/12*. |
| *vlan_id* | The VLAN of the MAC address;<br>an integer from 1-4094. |

## *python_get_fdb_count_info*

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

### Syntax

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

where *dict_fdb_filter* contains the following optional variables:

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

### Returns

Multiple possible return values:

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

| 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.<br>Type - list containing interface names, which are string (for example, *Ethernet1/12*). |
| *vlan_id* | The VLANs to filter on (integer);<br>an integer from 1-4094. |

### Returns

Multiple possible return values:

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

| Element | Description |
|---|---|
| *address_table* | The list of MAC addresses matching the search filter.<br>Type - list with MAC addresses as string. |
| *is_static* | Whether the MAC address is statically configured or dynamically learned (boolean).<br>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).<br>For example: *Ethernet1/12*. |
| *vlan_id* | The VLAN of the MAC address (integer);<br>an integer from 1-4094. |

## *python_add_static_mac*

Configures a static MAC table entry or adds interfaces to already configured static MAC table entries.

### Syntax

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

where *return_dict_added_static_fdb* contains the following mandatory variables:

| Variable | Description |
|---|---|
| *mac_address* | The MAC address to configure (string). |
| *interfaces* | The list of switch interfaces for the MAC address.<br>Type - list containing interface names, which are string (for example, *Ethernet1/12*). |
| *vlan_id* | The VLAN for the MAC address (integer);<br>an integer from 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);<br>one of `static`, `dynamic`. |
| *mac_address* | (Optional) The MAC address to delete (string). |
| *interfaces* | (Optional) The list of switch interfaces for the multicast MAC address.<br>Type - list containing interface names, which are string (for example, *Ethernet1/12*). |
| *vlan_id* | (Optional) The VLAN for the MAC address (integer);<br>an integer from 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).<br>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);<br>one of `enabled`, `disabled`. |

## *python_set_fdb_plearning_state*

Configures MAC learning on a specific interface.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The name of the switch interface (string).<br>For example: *Ethernet1/12*. |
| *learning_state* | The status of MAC learning (string);<br>one of `enabled`, `disabled`. |

### Returns

Multiple possible return values:

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

## *python_get_fdb_glearning_cfg*

Displays the status of global MAC learning on the switch.

### Syntax

```
python_get_fdb_glearning_cfg()
```

### Returns

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

| Element | Description |
|---|---|
| *global_learning_state* | The status of global MAC learning (string); one of `enabled`, `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); one of `enabled`, `disabled`. |

### Returns

Multiple possible return values:

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

## *python_get_fdb_aging_time_cfg*

Displays MAC table entry aging time configuration.

### Syntax

```
python_get_fdb_aging_time_cfg()
```

### Returns

A dictionary showing the MAC aging time configuration:

| Element | Description |
|---|---|
| *aging_time* | The MAC aging time, in seconds;<br>an integer from 0-1000000. |

## *python_set_fdb_aging_time_cfg*

Configures MAC table entry aging time on the switch.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *aging_time* | The MAC aging time, in seconds;<br>an integer from 0-1000000. |

### Returns

Multiple possible return values:

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

# 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()*

Updates 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()*

Updates 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()*

Gets 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()*

Halts the system and performs 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()*

Gets 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); one of `yes`, `no`. Default value: `yes`. |

### *python_igmp_snoop_set()*

Sets 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); one of `yes`, `no`. Default value: `yes`. |

#### 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()*

Gets 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()*

Gets 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()*

Gets 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()*

Sets 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:<br>• `vid`<br>• `ena_igmp_snoop` |
| *vid* | The VLAN ID; an integer from 1-3999. |
| *ena_igmp_snoop* | Whether IGMP snooping is enabled (string);<br>one of `yes`, `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()*

Gets the IP properties of a specified interface.

#### Syntax

`get_ipinfo(<`*if_name*`>)`

where:

| Variable | Description |
|---|---|
| *if_name* | (Optional) The interface name (string);<br>Default value: `None`. |

#### Returns

A dictionary containing the IP details:

| Element | Description |
|---|---|
| *if_name* | IP interface name.<br>**Note:** The interface must exist. |
| *switchport* | Whether or not the port is a switchport;<br>one of `yes`, `no`. Default value: `yes`. |
| *mtu* | The maximum transmission unit, in bytes;<br>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()*

Sets the IP address of a specified interface.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | IP interface name.<br>**Note:** The interface must exist. |
| *ip_addr* | IP address for the interface. |
| *secondary* | Secondary value for an interface (integer). |

### Returns

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

## *set_bridge_port()*

Makes the specified interface a bridge port.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | IP interface name.<br>**Note:** The interface must exist. |

### Returns

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

## *unset_bridge_port()*

Changes the specified interface from a bridge port to a routed port.

### Syntax

`unset_bridge_port(<if_name>)`

where:

| Variable | Description |
|---|---|
| *if_name* | IP interface name (string).<br>**Note:** The interface must exist. |

### Returns

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

## *set_if_flagup()*

Sets the interface flag to make it operational.

### Syntax

`set_if_flagup(<if_name>)`

where:

| Variable | Description |
|---|---|
| *if_name* | IP interface name.<br>**Note:** The interface must exist. |

### Returns

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

## *unset_if_flagup()*

Unsets the interface flag to make it non-operational.

### Syntax

`unset_if_flagup(<if_name>)`

where:

| Variable | Description |
|---|---|
| *if_name* | IP interface name (string).<br>**Note:** The interface must exist. |

### Returns

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

## *set_if_bgp_unnumbered*

Enables BGP unnumbered on a switch interface.

### Syntax

`set_if_bgp_unnumbered(<if_name>)`

where:

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

### 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()*

Gets LACP system priority.

#### Syntax

```
python_lacp_get_sys_priority()
```

#### Returns

The LACP system priority (integer).

### *python_lacp_get_max_bundle()*

Gets the LACP max bundle, which is the supported maximum number of links for each LAG.

#### Syntax

```
python_lacp_get_max_bundle()
```

#### Returns

The supported maximum number of links per LAG (integer).

### *python_lacp_get_all_link_details()*

Gets all LACP interface details.

#### Syntax

```
python_lacp_get_all_link_details()
```

#### Returns

A list of dictionaries containing LACP link details, where:

| Element | Description |
|---|---|
| *if_name* | Ethernet interface name (string).<br>**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;<br>a positive integer from 1-65535. Default value: 32768. |
| *lacp_timeout* | LACP timeout for the physical port;<br>one of `short`, `long`. Default value: `long`. |

## *python_lacp_set_system()*

Sets LACP system priority.

### Syntax

`python_lacp_set_system(`<*sys_prio*>`)`

where:

| Variable | Description |
|---|---|
| *sys_prio* | The interface system priority (integer). |

### 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()*

Gets a list of all the LAG information for the device.

#### Syntax

```
python_get_lag()
```

#### Returns

A list of LAG dictionaries containing LAG information for the device:

| 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:<br>• `if_name`<br>• `lag_mode`<br>• `lacp_prio`<br>• `lacp_timeout` |
| *if_name* | Ethernet interface name (string).<br>**Note:** The interface must exist. |
| *lag_mode* | LAG mode (string);<br>one of `lacp_active`, `lacp_passive`, `no_lacp`. |
| *lacp_prio* | LACP priority for the physical port;<br>a positive integer from 1-65535. Default value: 32768. |
| *lacp_timeout* | LACP timeout for the physical port (string);<br>one of `short`, `long`. Default value: `long`. |

## *python_get_lag_id()*

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

### Syntax

`python_get_lag(`*<lag_id>*`)`

where:

| 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:<br>• `if_name`<br>• `lag_mode`<br>• `lacp_prio`<br>• `lacp_timeout`<br>Up to 32 interfaces can be added. |
| *if_name* | Ethernet interface name (string).<br>**Note:** The interface must exist. |
| *lag_mode* | LAG mode (string);<br>one of `lacp_active`, `lacp_passive`, `no_lacp`. |
| *lacp_prio* | LACP priority for the physical port;<br>a positive integer from 1-65535. Default value: 32768. |
| *lacp_timeout* | LACP timeout for the physical port (string);<br>one of `short`, `long`. Default value: `long`. |
| *suspend_ individual* | Whether the LACP state is suspended or individual (string);<br>one of `Individual`, `Suspended`, `N/A`. |

## *python_update_lag_id()*

Updates 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:<br>• `if_name`<br>• `lag_mode`<br>• `lacp_prio`<br>• `lacp_timeout`<br>Up to 32 interfaces can be added. |
| *if_name* | Ethernet interface name (string).<br>**Note:** The interface must exist. |
| *lag_mode* | LAG mode (string);<br>one of `lacp_active`, `lacp_passive`, `no_lacp`. |
| *lacp_prio* | LACP priority for the physical port;<br>a positive integer from 1-65535. Default value: 32768. |
| *lacp_timeout* | LACP timeout for the physical port (string);<br>one of `short`, `long`. Default value: `long`. |
| *suspend_ individual* | Whether the LACP state is suspended or individual (string);<br>one of `Individual`, `Suspended`, `N/A`. |

## *python_update_lag_id_details()*

Updates LAG information for the specified LAG ID.

### Syntax

`python_update_lag_id_details(<lag>)`

where *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:<br>• `if_name`<br>• `lag_mode`<br>• `lacp_prio`<br>• `lacp_timeout`<br>Up to 32 interfaces can be added. |
| *if_name* | Ethernet interface name (string).<br>**Note:** The interface must exist. |
| *lag_mode* | LAG mode (string);<br>one of `lacp_active`, `lacp_passive`, `no_lacp`. |
| *lacp_prio* | LACP priority for the physical port;<br>a positive integer from 1-65535. Default value: 32768. |
| *lacp_timeout* | LACP timeout for the physical port (string);<br>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:<br>• `if_name`<br>• `lag_mode`<br>• `lacp_prio`<br>• `lacp_timeout`<br>Up to 32 interfaces can be added. |
| *if_name* | Ethernet interface name (string).<br>**Note:** The interface must exist. |
| *lag_mode* | LAG mode (string);<br>one of `lacp_active`, `lacp_passive`, `no_lacp`. |
| *lacp_prio* | LACP priority for the physical port;<br>a positive integer from 1-65535. Default value: 32768. |
| *lacp_timeout* | LACP timeout for the physical port (string);<br>one of `short`, `long`. Default value: `long`. |

### Returns

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

## *python_delete_lag_all()*

Deletes all LAGs on the device.

### Syntax

```
python_delete_lag_all()
```

### Returns

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

## *python_delete_lag_id()*

### Syntax

`python_delete_lag_id(`*<lag_id>*`)`

where:

| 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()*

Gets 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 (integer).

### *python_lldp_get_msg_tx_interval()*

Gets the time interval in seconds between transmissions of LLDP messages.

#### Syntax

```
python_lldp_get_msg_tx_interval()
```

#### Returns

The transmit interval value, in seconds (integerS).

### *python_lldp_get_tx_delay()*

Gets the number of seconds for LLDP transmission delay.

#### Syntax

```
python_lldp_get_tx_delay()
```

#### Returns

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

## *python_lldp_set_reinit_delay()*

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

### Syntax

`python_lldp_set_reinit_delay(`*<reinit_delay>*`)`

where:

| Variable | Description |
|---|---|
| *reinit_delay* | The time to wait, in seconds (integer). |

### Returns

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

## *python_lldp_set_msg_tx_interval()*

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

### Syntax

`python_lldp_set_msg_tx_interval(`*<tx_interval>*`)`

where:

| Variable | Description |
|---|---|
| *tx_interval* | The transmission rate, in seconds (integer). |

### Returns

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

## *python_lldp_set_tx_delay()*

Sets the delay time in seconds.

### Syntax

`python_lldp_set_tx_delay(`*<tx_delay>*`)`

where:

| Variable | Description |
|---|---|
| *tx_delay* | The transmission delay, in seconds (integer).<br>**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()*

Gets 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);<br>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()*

Gets 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);<br>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()*

Gets 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 (integer). |
| *total tlvs discarded* | The total number of LLDP TLVs discarded (integer). |
| *total frames transmitted* | The total number of LLDP frames transmitted (integer). |
| *total errrored frames* | The total number of frames received with errors (integer). |
| *total frames discarded* | The total number of discarded frames (integer). |
| *total entries aged* | The total number of aged-out entries (integer). |
| *total tlvs unrecognized* | The total number of unrecognized LLDP TLVs (integer). |

## class LldpInterface

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

### *python_lldp_get_interface()*

Gets LLDP interface admin status of a specific interface.

#### Syntax

`python_lldp_get_interface(`<*ifname*>`)`

where:

| 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()*

Gets LLDP interface admin status for all interfaces.

### Syntax

```
python_lldp_get_all_interface()
```

### Returns

A list of dictionaries containing LLDP status information for all interfaces:

| 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`.<br>Default value: `Yes`. |

## *python_lldp_set_interface()*

Sets LLDP interface admin status based on the lldp_interface_admin_status dictionary values.

### Syntax

```
python_lldp_set_interface(<lldp_interface_port_status>)
```

where:

| Variable | Description |
|---|---|
| *lldp_interface_port_status* | The LLDP interface status (dictionary) containing:<br>• `if_name`<br>• `ena_lldp_rx`<br>• `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 (integer).

## *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 (integer). |

### 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()*

Checks 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 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 (integer).

### *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).<br>**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).<br>**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 (integer):

| 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()*

Sets the MSTP interface port priority.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *ifname* | The name of the interface (string).<br>**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`).

# Nexthop Health Check

The class and function in this module manage Nexthop health check. To use this module, in the Python file or in the Python interpreter, enter:

```
import nhopHealthcheckApi
```

## class NexthopHealthCheck()

This class provides functions that manage nexthop health check.

### *set_nexthop_healthcheck_interval()*

Sets nexthop health check interval.

#### Syntax

```
set_nexthop_healthcheck_interval(<interval>)
```

where:

| Variable | Description |
|---|---|
| *interval* | (Mandatory) The value of nexthop health check interval, in seconds; an integer from 5-60. |

#### Returns

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

### *unset_nexthop_healthcheck()*

Disables nexthop health check interval.

#### Syntax

```
unset_nexthop_healthcheck()
```

#### Returns

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

# NTP Authentication Module

The class and functions in this module manage Network Type Protocol (NTP) authentication. To use this module, in the Python file or in the Python interpreter, enter:

```
import ntpApi
```

## class NtpSystem()

This class provides functions to manage NTP authentication.

### *python_ntp_show_keys_info()*

Gets authentication keys information.

#### Syntax

```
python_ntp_show_keys_info
```

#### Returns

A list with the authentication keys information.

### *python_ntp_set_auth_keys()*

Set the NTP authentication key.

#### Syntax

`python_ntp_set_auth_keys(`*<dict_ntp>*`)`

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

| Variable | Description |
|---|---|
| *key_num* | The key identifier; an integer from 1-65534. |
| *md5 or sha1* | The authentication type (string); one of `MD5`, `SHA1`. |

#### Returns

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

# NWV Authentication Module

The class and functions in this module manage Network Virtualization (NWV) authentication. To use this module, in the Python file or in the Python interpreter, enter:

```
import nwvApi
```

## class NwvCfg()

This class provides a function to set NWV configurations.

### *get_nwv()*

Gets NWV mode information.

#### Syntax

```
get_nwv()
```

#### Returns

A dictionary containing NWV information:

| Variable | Description |
|---|---|
| *encapsulation* | The VXLAN encapsulation (string). Default value: `vtep`. |
| *ha* | The High Availability mode status (string);<br>one of `disabled`, `vlag`. Default value: `disabled`. |
| *mode* | The NWV mode (string); one of `disabled`, `Static`, `Bgp-evp`, `Vxlan not ready`. Default value: `disabled`.<br>**Note:** When the mode is `Vxlan not ready`, the configuration is incorrect and you must change the vLAG settings. |

### *set_nwv_mode()*

Sets the NWV mode.

#### Syntax

```
set_nwv_mode(<mode>)
```

where:

| Variable | Description |
|---|---|
| *mode* | (Mandatory) The new NWM mode; an integer with the following values:<br>• `0 (Disabled)`<br>• `1 (Static)`<br>• `3 (Bgp-epvn)`<br>• `4 (Static-ha)`<br>• `5 (Bgp-evpn-ha)` |

## Returns

Boolean (`True` on success, otherwise `False`). Otherwise, you must change the NWV mode. For example, to change the `Bgp-epvn` mode, you must first set it to `Disabled` and then `Static`. It is not valid to change the `Bgp-epvn` mode directly to `Static`.

# OSPF Module

The class in this module contains 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()*

Gets 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`".<br>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 the following format: HH:MM:SS. |
| *router_id_changes* | Router ID changes counter; a positive integer. |
| *dr_election_counter* | DR elections counter; a positive integer. |
| *older_lsas_counter* | Older received LSAs counter; a positive integer. |
| *nbr_state_change_counter* | Neighbor state changes counter; a positive integer. |
| *nbr_bad_lsreqs_counter* | Neighbor bad LS received requests counter; a positive integer. |
| *nbr_interval_expired_counter* | Neighbor dead-interval expirations counter; a positive integer. |

| Element | Description |
|---|---|
| *nbr_seq_number_mismatch* | Neighbor sequence number mismatches counter; a positive integer. |
| *spf_full* | Full SPF Computations counter; a positive integer. |
| *spf_summary* | Summary SPF Computations counter; a positive integer. |
| *spf_external* | External SPF Computations counter; a positive integer. |
| *recv_buf* | Received packet buffer; a positive integer. |
| *send_buf* | Sent packet buffer; a positive integer. |
| *lsa_buf* | LSA buffer; a positive integer. |
| *packet_unuse* | Unused packets number; a positive integer. |
| *packet_max* | Maximum packets number; a positive integer. |
| *lsa_unuse* | Unused LSAs number; a positive integer. |
| *lsa_max* | Maximum LSAs number; a positive integer. |
| *router_lsa_type* | Router LSA type name; a positive integer. |
| *routerLsa_generated* | Number of generated router LSAs; a positive integer. |
| *routerLsa_refreshed* | Number of refreshed router 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. |
| *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. |

| Element | Description |
|---|---|
| *summaryLsa_flushed* | Number of flushed summary LSAs;<br>a positive integer. |
| *summaryLsa_agedOut* | Number of aged out summary LSAs;<br>a positive integer. |
| *asbrSummaryLsa_generated* | Number of generated ASBR summary LSAs;<br>a positive integer. |
| *asbrSummaryLsa_refreshed* | Number of refreshed ASBR summary LSAs;<br>a positive integer. |
| *asbrSummaryLsa_flushed* | Number of flushed ASBR summary LSAs;<br>a positive integer. |
| *asbrSummaryLsa_agedOut* | Number of aged out ASBR summary LSAs;<br>a positive integer. |
| *asExternalLsa_generated* | Number of generated AS-External LSAs;<br>a positive integer. |
| *asExternalLsa_refreshed* | Number of refreshed AS-External LSAs;<br>a positive integer. |
| *asExternalLsa_flushed* | Number of flushed AS-External LSAs;<br>a positive integer. |
| *asExternalLsa_agedOut* | Number of aged out AS-External LSAs;<br>a positive integer. |
| *asNssaLsa_generated* | Number of generated AS-NSSA LSAs;<br>a positive integer. |
| *asNssaLsa_refreshed* | Number of refreshed AS-NSSA LSAs;<br>a positive integer. |
| *asNssaLsa_flushed* | Number of flushed AS-NSSA LSAs;<br>a positive integer. |
| *asNssaLsa_agedOut* | Number of aged out AS-NSSA LSAs;<br>a positive integer. |
| *type8Lsa_generated* | Number of generated type-8 LSAs;<br>a positive integer. |
| *type8Lsa_refreshed* | Number of refreshed type-8 LSAs;<br>a positive integer. |
| *type8Lsa_flushed* | Number of flushed type-8 LSAs;<br>a positive integer. |
| *type8Lsa_agedOut* | Number of aged out type-8 LSAs;<br>a positive integer. |
| *linkOpaqueLsa_generated* | Number of generated Link Opaque LSAs;<br>a positive integer. |

| Element | Description |
|---|---|
| *linkOpaqueLsa_refreshed* | Number of refreshed Link Opaque LSAs; a positive integer. |
| *linkOpaqueLsa_flushed* | Number of flushed Link Opaque LSAs; a positive integer. |
| *linkOpaqueLsa_agedOut* | Number of aged out Link Opaque LSAs; a positive integer. |
| *areaOpaque_lsa_type* | Area Opaque LSA type name; a positive integer. |
| *areaOpaqueLsa_generated* | Number of generated Area Opaque LSAs; a positive integer. |
| *areaOpaqueLsa_refreshed* | Number of refreshed Area Opaque LSAs; a positive integer. |
| *areaOpaqueLsa_flushed* | Number of flushed Area Opaque LSAs; a positive integer. |
| *areaOpaqueLsa_agedOut* | Number of aged out Area Opaque LSAs; a positive integer. |
| *asOpaque_lsa_type* | AS Opaque LSA type name; a positive integer. |
| *asOpaqueLsa_generated* | Number of generated AS External Opaque LSAs; a positive integer. |
| *asOpaqueLsa_refreshed* | Number of refreshed AS External Opaque LSAs; a positive integer. |
| *asOpaqueLsa_flushed* | Number of flushed AS External Opaque LSAs; a positive integer. |
| *asOpaqueLsa_agedOut* | Number of aged out AS External Opaque LSAs; a positive integer. |

## *python_ospf_get_traffic_info()*

Gets OSPF traffic statistics.

### Syntax

`python_ospf_get_traffic_info(`*<vrf_name>*`)`

where:

| 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 the following format: HH:MM:SS. |
| *total_pkt_in* | Number of total packets in; a positive integer. |
| *total_pkt_out* | Number of total packets out; a positive integer. |
| *hello_in* | Number of hello packets in; a positive integer. |
| *hello_out* | Number of hello packets out; a positive integer. |
| *db_desc_in* | Number of DB descriptor packets in; a positive integer. |
| *db_desc_out* | Number of DB descriptor packets out; a positive integer. |
| *ls_req_in* | Number of LS Request packets in; a positive integer. |
| *ls_req_out* | Number of LS Request packets out; a positive integer. |
| *ls_upd_in* | Number of LS Update packets in; a positive integer. |
| *ls_upd_out* | Number of LS Update packets out; a positive integer. |
| *ls_ack_in* | Number of LS ACK packets in; a positive integer. |
| *ls_ack_out* | Number of LS ACK packets out; a positive integer. |
| *error_drops_in* | Number of errors related to drops in; a positive integer. |
| *error_drops_out* | Number of errors related to drops out; a positive integer. |
| *error_hellosin* | Number of errors related to hellos in; a positive integer. |
| *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. |

| Element | Description |
|---|---|
| *error_wrong_area* | Number of errors related to Wrong Area; a positive integer. |
| *error_bad_version* | Number of errors related to Bad Version; a positive integer. |
| *error_bad_auth* | Number of errors related to Bad Authentication; a positive integer. |
| *error_passive* | Number of errors related to Passive; a positive integer. |
| *error_nonbr* | Number of errors related to No Neighbor; a positive integer. |
| *error_invalid_src* | Number of errors related to Invalid Source; a positive integer. |
| *error_invalid_dst* | Number of errors related to Invalid Destination; a positive integer. |
| *error_pktlength* | Number of errors related to Packet Length; a positive integer. |

## *python_ospf_get_neighbor_info()*

Gets OSPF neighbors statistics.

### Syntax

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

where:

| 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 the following format: HH:MM:SS. |

| Element | Description |
|---|---|
| *nbr_addr* | Neighbor IP address; a valid IPv4 or IPv6 address. |
| *ifp_name* | Ethernet interface name. |

## *python_ospf_get_routes_info()*

Gets OSPF routes statistics.

### Syntax

`python_ospf_get_routes_info(<vrf_name>)`

where:

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

### Returns

A dictionary containing OSPF routes statistics:

| Element | Description |
|---|---|
| *network* | Network name; a string in the following format: "AA:BB:CC:DD/MM". |
| *pathcode* | Path type; one of:<br>• `connected`<br>• `Discard`<br>• `OSPF`<br>• `OSPF inter area`<br>• `OSPF NSSA external type 1`<br>• `OSPF NSSA external type 2`<br>• `OSPF external type 1`<br>• `OSPF external type 2` |
| *pathCount* | Number of ecmp paths; a positive integer. |
| *route_path_cost* | Route-path cost; a positive integer. |

| Element | Description |
|---|---|
| *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:<br>• `interface`: Neighbor IP address; a valid IPv4 or IPv6 address.<br>• `area_id`: Neighbor area ID; a valid IPv4 or IPv6 address.<br>• `neighbor_addr`: Neighbor IP address; a valid IPv4 or IPv6 address. |

## *python_ospf_get_database_info()*

Gets OSPF database statistics.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *vrf_name* | (Optional) Virtual Routing and Forwarding name; one of the VRF name or "`default`".<br>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:<br>• `Router-LSA`<br>• `Network-LSA`<br>• `Summary-LSA`<br>• `ASBR-summary-LSA`<br>• `AS-external-LSA`<br>• `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. |

| Element | Description |
|---|---|
| *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`. |

## *python_ospf_get_border_routers_info()*

Gets OSPF border routers.

### Syntax

`python_ospf_get_border_routers_info(<`*vrf_name*`>)`

where:

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

### Returns

A dictionary containing OSPF Area Border Router (ABR) and Autonomous System Boundary Router (ASBR) statistics.:

| Element | Description |
|---|---|
| *abr_id* | The ABR ID (string); shows the type, router ID or cost. |
| *abr_route_type* | Type of router related to ABR (string). |
| *abr_route_metric* | Metric of router related to ABR (string). |
| *asbr_id* | The ASBR ID (string); shows the type, router ID or cost. |
| *asbr_route_type* | Type of router related to ASBR (string). |
| *asbr_route_metric* | Metric of router related to ASBR (string). |
| *type_border_router* | The border router type (string); one of `ABR` or `ASBR`. |
| *abr_via* | The next-hop IP for ABR (string); a valid IP address. |
| *asbr_via* | The next-hop IP for ABSBR (string); a valid IP address. |
| *abr_transit_area* | The transit area ID for ABR (string); a valid IP address. |
| *asbr_transit_area* | The transit area ID for ASBR (string);<br>a valid IP address. |

| Element | Description |
|---|---|
| *abr_area_ifname* | The OSPF interface for ABR (string).<br>For example: *Ethernet1/X* or *VLAN interface*. |
| *asbr_area_ifname* | The OSPF interface for ABSR (string).<br>For example: *Ethernet1/X* or *VLAN interface*. |

## *python_ospf_get_summary_address_info()*

Gets OSPF summary address.

### Syntax

`python_ospf_get_summary_address_info(<`*vrf_name*`>)`

where:

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

### Returns

A dictionary showing OSPF summary address information:

| Element | Description |
|---|---|
| *router_id* | Router ID in IP address format (string); a valid IP address. |
| *ospf_id* | OSPF identifier (integer). Default value: `0`. |
| *vrf_name* | Default VRF name (string).<br>Default value: `default`. |
| *prefix* | The IP prefix (string); in the following format: *XX.XX.XX.XX/XX.* |
| *metric* | The metric value; an integer from 0-16777214. |
| *tag* | External/NSSA LSAs tag;<br>a positive integer from 0-4294967295. |
| *summary_address_state* | The summary address status (string);<br>one of `Active`, `Pending`. |

## *python_ospf_get_interface_info()*

Gets OSPF interface information.

### Syntax

`python_ospf_get_interface_info(<`*if_name*`>)`

where:

| Variable | Description |
|---|---|
| *if_name* | (Optional) Ethernet interface name (string).<br>**Note:** The interface must exist. |

## Returns

A dictionary showing OSPF interface information:

| Element | Description |
|---|---|
| *if_name* | The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface*. |
| *vrf_name* | Default VRF name (string). Default value: `default`. |
| *ospf_id* | OSPF process identifier (integer). Default value: `0`. |
| *ospf_status* | The status of the OSPF protocol (string);<br>one of `Up`, `Down`. |
| *if_addr* | The IP address or mask (string); a valid IP address or mask. |
| *if_area_id* | The area ID (string); a valid IP address. |
| *if_mtu* | The maximum transmission unit;<br>a positive integer from 576-65535. |
| *router_id* | The router ID in IP address format (string); a valid IP address. |
| *if_network_type* | The network type (string);<br>one of `Broadcast`, `Point-to-Point`. |
| *if_output_cost* | Interface output cost; a positive integer from 1-65535. |
| *if_transmit_delay* | The interface transmit delay, in seconds;<br>an integer from 1-3600. |
| *priority* | The router priority; an integer from 0-255. |
| *if_state* | The operation state of the interface (string);<br>one of `DR`, `Backup`, `DRother`. |
| *designated_router* | Designated router ID (string); a valid IP address. |
| *designated_router_addr* | The IP address for the designated router (string). |
| *backup_designated_router* | The backup router ID for the designated router (string); a valid IP address. |
| *backup_designated_router_addr* | The backup router ID for the designated router. (string). |

| Element | Description |
|---|---|
| *hello_interval* | The hello interval, in seconds;<br>an integer from 1-65535. |
| *dead_interval* | The dead interval, in seconds;<br>an integer from 1-65535. |
| *retransmit_interval* | The retransmit interval, in seconds;<br>an integer from 1-65535. |
| *if_hello_timer* | The hello interval timer expiration time (string). |
| *neighbor_count* | The neighbor count (integer); a positive integer. |
| *hello_in* | Number of total hello packets in; a positive integer. |
| *hello_out* | Number of total hello packets out; a positive integer. |
| *ls_req_in* | Number of total LS Request packets in;<br>a positive integer. |
| *ls_req_out* | Number of total LS Request packets out;<br>a positive integer. |
| *ls_upd_in* | Number of total LS Update packets in;<br>a positive integer. |
| *ls_upd_out* | Number of total LS Update packets out;<br>a positive integer. |
| *ls_ack_in* | Number of total LS ACK packets in;<br>a positive integer. |
| *ls_ack_out* | Number of total LS ACK packets out;<br>a positive integer. |
| *db_desc_in* | Number of total DB Descriptors packets in;<br>a positive integer. |
| *db_desc_out* | Number of total DB Descriptors packets out;<br>a positive integer. |
| *discarded* | Number of total discarded packets;<br>a positive integer. |
| *auth_type* | The type of authentication;<br>one of `Message-Digest`, `Simple`, `Null`. |
| *key_id* | The Key-ID, if the authentication type is<br>MD5/SHA256; an integer from 1-255. |
| *if_mtu_ignore* | The maximum transmission unit status;<br>one of `Enable`, `Disable`. |
| *passive_interface* | The passive interface status;<br>one of `Enable`, `Disable`. |

| Element | Description |
|---|---|
| *if_bfd* | The BDF status; one of `Enable`, `Disable`. |
| *db_filter_all_out* | Database filter all out; one of `Enable`, `Disable`. |

## *python_ospf_get_vlinks_info()*

Gets OSPF virtual-links.

### Syntax

`python_ospf_get_vlinks_info(`*<area_id>*,*<nbr_router_id>*,*<vlink_name>*`)`

where:

| Variable | Description |
|---|---|
| *area_id* | (Optional) The transit area ID (string); a valid IP address. |
| *nbr_router_id* | (Optional) The neighbor router ID (string); a valid IP address. |
| *vlink_name* | (Optional) The virtual-link name (string). |

### Returns

A dictionary containing OSPF virtual-links statistics:

| Element | Description |
|---|---|
| *vrf_name* | Default VRF name. Default value: `default`. |
| *vlink_name* | The virtual-link name (string). |
| *nbr_router_id* | The neighbor router ID (string); a valid IP address. |
| *ifp_name* | The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface*. |
| *local_address* | The local interface IP address (string);<br>a valid IP address. |
| *remote_address* | The remote interface IP address (string);<br>a valid IP address. |
| *area_id* | The transit area ID (string); a valid IP address. |
| *transmit_delay* | The transmission delay interval, in seconds;<br>an integer from 1-3600. |
| *vlink_state* | The Virtual Link status (string); one of `Up`, `Down`. |
| *hello_interval* | The hello interval, in seconds;<br>an integer from 1-65535. |

| Element | Description |
|---|---|
| *dead_interval* | The dead interval, in seconds;<br>an integer from 1-65535. |
| *wait_interval* | The wait interval, in seconds;<br>an integer from 1-65535. |
| *retransmit_interval* | The retransmit interval, in seconds;<br>an integer from 1-65535. |
| *hello_due* | The due time to send the next hello (string). |
| *adjacency_state* | The adjacency state across the virtual-link (string). |
| *auth_type* | The type of authentication (string);<br>one of `Message-Digest`, `Simple`, `Null`. |
| *key_id* | The Key-ID, if the authentication type is<br>MD5/SHA256; an integer from 1-255. |
| *bfd* | The status of BFD (string);<br>one of `Enable`, `Disable`. |

## *python_set_ospf_if_ospf_status()*

Sets OSPF on an interface.

### Syntax

`python_set_ospf_if_ospf_status`(<*interface*>,<*ospf_status*>)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *ospf_status* | (Mandatory) The status of the OSPF protocol (string);<br>one of `Enable`, `Disable`. |

### Returns

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

## *python_set_ospf_if_hello_interval*

Sets the hello interval for the OSPF interface.

### Syntax

`python_set_ospf_if_hello_interval`(<*interface*>,<*hello_interval*>)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *hello_interval* | (Mandatory) The hello interval, in seconds;<br>an integer from 1-65535. |

### Returns

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

## *python_set_ospf_if_dead_interval()*

Sets the dead interval for the OSPF interface.

### Syntax

`python_set_ospf_if_dead_interval`(*<interface>*,*<dead_interval>*)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *dead_interval* | (Mandatory) The dead interval, in seconds;<br>an integer from 1-65535. |

### Returns

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

## *python_set_ospf_if_retransmit_interval()*

Sets the retransmit interval for the OSPF interface.

### Syntax

`python_set_ospf_if_retransmit_interval`(*<interface>*,
*<retransmit_interval>*)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *retransmit_interval* | (Mandatory) The retransmit interval, in seconds;<br>an integer from 1-65535. |

### Returns

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

## *python_set_ospf_if_transmit_delay()*

Sets the transmit delay for the OSPF interface.

### Syntax

`python_set_ospf_if_transmit_delay`(<*interface*>, <*if_transmit_delay*>)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *if_transmit_delay* | (Mandatory) The interface transmit delay;<br>an integer from 1-3600. |

### Returns

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

## *python_set_ospf_if_priority()*

Sets the OSPF interface priority.

### Syntax

`python_set_ospf_if_priority`(<*interface*>,<*priority*>)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *priority* | (Mandatory) The router priority; an integer from 0-255. |

### Returns

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

## *python_set_ospf_if_output_cost()*

Sets the OSPF interface output cost.

### Syntax

`python_set_ospf_if_output_cost`(*<interface>*,*<cost>*)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *cost* | (Mandatory) The OSPF cost; an integer from 0-65535. |

### Returns

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

## *python_set_ospf_if_network_type()*

Sets the interface network type.

### Syntax

`python_set_ospf_if_network_type`(*<interface>*,*<network_type>*)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *network_type* | (Mandatory) The network type (string);<br>one of `Broadcast`, `Point-to-Point`. |

### Returns

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

## *python_set_ospf_if_mtu()*

Sets the Maximum Transmission Unit (MTU) for the OSPF interface.

### Syntax

`python_set_ospf_if_mtu`(*<interface>*,*<if_mtu>*)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *if_mtu* | (Mandatory) The maximum transmission unit;<br>a positive integer from 576-65535. |

### Returns

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

## *python_set_ospf_if_mtu_ignore()*

Sets interface MTU to ignore.

### Syntax

`python_set_ospf_if_mtu_ignore`(*<interface>*,*<if_mtu_ignore>*)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *if_mtu_ignore* | (Mandatory) Interface MTU ignore (string);<br>one of `Enable`, `Disable`. |

### Returns

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

## *python_set_ospf_if_bfd()*

Sets interface BFD.

### Syntax

`python_set_ospf_if_bfd`(*<interface>*,*<if_bfd>*)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *if_bfd* | (Mandatory) Interface BFD (string);<br>one of `Enable`, `Disable`. |

### Returns

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

## *python_set_ospf_if_authentication_type()*

Sets the interface authentication type.

### Syntax

`python_set_ospf_if_authentication_type`(*<interface>*,*<auth_type>*)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *auth_type* | The type of authentication (string);<br>one of `Message-Digest`, `Simple`, `Null`. |

### Returns

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

## *python_set_ospf_if_authentication_key()*

Sets the interface authentication key.

### Syntax

`python_set_ospf_if_authentication_key`(*<interface>*,*<auth_key>*)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *auth_key* | The authentication key (string). |

### Returns

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

## *python_set_ospf_if_message_digest_key()*

Configures the MD5/SHA-256 key.

### Syntax

`python_set_ospf_if_message_digest_key`(*<interface>*,*<key_id>*,*<key_type>*, *<password>*)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *key_id* | (Mandatory) The key identifier; an integer from 1-255. |
| *key_type* | (Mandatory) The key type, MD5:1 or SHA-256:2 (integer); one of `1`, `2`. |
| *password* | (Mandatory) The encrypted MD5 or SHA-256 password (string). |

### Returns

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

## *python_set_ospf_if_passive()*

Sets the interface as passive.

### Syntax

`python_set_ospf_if_passive`(<*interface*>,<*if_passive_cfg*>)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *if_passive_cfg* | (Mandatory) The interface passive configuration (string); one of `Enable`, `Disable`. |

### Returns

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

## *python_set_ospf_if_area_id()*

Sets the area ID for the OSPF interface.

### Syntax

`python_set_ospf_if_area_id`(<*interface*>,<*area_id*>)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *area_id* | (Mandatory) The area ID (string); a valid IP address. |

### Returns

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

## *python_set_ospf_if_db_filter_out()*

Configures the database filter.

### Syntax

`python_set_ospf_if_db_filter_out`(*<interface>,<db_filter_out>*)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *db_filter_out* | (Mandatory) The database filter out configuration (string); one of `Enable`, `Disable`. |

### Returns

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

## *python_set_ospf_if_unset_config()*

Unset the list of interface configurations.

### Syntax

`python_set_ospf_if_unset_config`(*<interface>,<unset_list>*)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *unset_list* | (Mandatory) Displays a list of strings:<br>• `auth_key`<br>• `auth_type`<br>• `hello_interval`<br>• `dead_interval`<br>• `if_transmit_delay`<br>• `retransmit_interval`<br>• `if_output_cost`<br>• `priority`<br>• `if_mtu` |

### Returns

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

## *python_set_ospf_if_unset_message_digest_key_config()*

Unset the OSPF interface MD5 key.

### Syntax

`python_set_ospf_if_unset_message_digest_key_config` (*<interface>*,*<key_id>*)

where:

| Variable | Description |
|---|---|
| *interface* | (Mandatory) The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface.* |
| *key_id* | (Mandatory) The MD5 key; an integer from 1-255. |

### Returns

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

## *python_set_ospf_virtual_link()*

Configures the OSPF virtual-link.

### Syntax

`python_set_ospf_virtual_link`(*<area_id>*,*<nbr_router_id>*)

where:

| Variable | Description |
|---|---|
| *area_id* | (Mandatory) The area ID (string); a valid IP address. |
| *nbr_router_id* | (Mandatory) The neighbor router ID (string); a valid IP address. |

### Returns

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

## *python_set_ospf_virtual_link_disable()*

Disables the virtual-link.

### Syntax

`python_set_ospf_virtual_link_disable`(*<area_id>*,*<nbr_router_id>*, *<disable>*)

where:

| Variable | Description |
|---|---|
| *area_id* | (Mandatory) The area ID (string); a valid IP address. |
| *nbr_router_id* | (Mandatory) The neighbor router ID (string); a valid IP address. |
| *disable* | (Mandatory) Disable the state of the virtual-link (string); one of `Yes`, `No`. |

### Returns

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

## *python_set_ospf_virtual_hello_interval()*

Configures the hello interval for the virtual-link.

### Syntax

`python_set_ospf_virtual_hello_interval`(*<area_id>*,*<nbr_router_id>*, *<hello_interval>*)

where:

| Variable | Description |
|---|---|
| *area_id* | (Mandatory) The area ID (string); a valid IP address. |
| *nbr_router_id* | (Mandatory) The neighbor router ID (string); a valid IP address. |
| *hello_interval* | (Mandatory) The hello interval, in seconds; an integer from 1-65535. |

### Returns

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

## *python_set_ospf_virtual_dead_interval()*

Configures the dead interval for the virtual-link.

### Syntax

`python_set_ospf_virtual_dead_interval`(*<area_id>*,*<nbr_router_id>*, *<dead_interval>*)

where:

| Variable | Description |
|---|---|
| *area_id* | (Mandatory) The area ID (string); a valid IP address. |
| *nbr_router_id* | (Mandatory) The neighbor router ID (string); a valid IP address. |
| *dead_interval* | (Mandatory) The dead interval, in seconds; an integer from 1-65535. |

### Returns

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

## *python_set_ospf_virtual_retransmit_interval()*

Configures the retransmit interval for the virtual-link.

### Syntax

`python_set_ospf_virtual_retransmit_interval`(*<area_id>*, *<nbr_router_id>*,*<retransmit_interval>*)

where:

| Variable | Description |
|---|---|
| *area_id* | (Mandatory) The area ID (string); a valid IP address. |
| *nbr_router_id* | (Mandatory) The neighbor router ID (string); a valid IP address. |
| *retransmit_interval* | (Mandatory) The retransmit interval, in seconds; an integer from 1-65535. |

### Returns

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

## *python_set_ospf_virtual_transmit_delay()*

Configures the transmit delay for the virtual-link.

### Syntax

`python_set_ospf_virtual_transmit_delay`(*<area_id>*,*<nbr_router_id>*, *<transmit_delay>*)

where:

| Variable | Description |
|---|---|
| *area_id* | (Mandatory) The area ID (string); a valid IP address. |
| *nbr_router_id* | (Mandatory) The neighbor router ID (string); a valid IP address. |
| *transmit_delay* | (Mandatory) The interface transmit delay, in seconds; an integer from 1-3600. |

### Returns

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

## *python_set_ospf_virtual_bfd()*

Configures the BFD for virtual-link.

### Syntax

`python_set_ospf_virtual_bfd`(*<area_id>*,*<nbr_router_id>*, *<bfd_config>*)

where:

| Variable | Description |
|---|---|
| *area_id* | (Mandatory) The area ID (string); a valid IP address. |
| *nbr_router_id* | (Mandatory) The neighbor router ID (string); a valid IP address. |
| *bfd_cfg* | (Mandatory) The interface BDF configuration (string); one of `Enable`, `Disable`. |

### Returns

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

## *python_set_ospf_virtual_authentication_type()*

Configures the authentication type for the virtual-link.

### Syntax

`python_set_ospf_virtual_authentication_type`(*<area_id>*, *<nbr_router_id>*,*<auth_type>*)

where:

| Variable | Description |
|---|---|
| *area_id* | (Mandatory) The area ID (string); a valid IP address. |
| *nbr_router_id* | (Mandatory) The neighbor router ID (string); a valid IP address. |
| *auth_type* | (Mandatory) The type of authentication (string); one of `Message-Digest`, `Simple`, `Null`. |

### Returns

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

## *python_set_ospf_virtual_authentication_key()*

Configures the authentication key for the virtual link.

### Syntax

`python_set_ospf_virtual_authentication_key`(*<area_id>*, *<nbr_router_id>*,*<auth_key>*)

where:

| Variable | Description |
|---|---|
| *area_id* | (Mandatory) The area ID (string); a valid IP address. |
| *nbr_router_id* | (Mandatory) The neighbor router ID (string); a valid IP address. |
| *auth_key* | (Mandatory) The authentication key (string). |

### Returns

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

## *python_set_ospf_virtual_ message_digest _key()*

Configures the MD5 key for the virtual-link.

### Syntax

`python_set_ospf_virtual_message_digest_key`(*<area_id>*, *<nbr_router_id>*,*<key_id>*,*<key_type*,*<password>*)

where:

| Variable | Description |
|---|---|
| *area_id* | (Mandatory) The area ID (string); a valid IP address. |
| *nbr_router_id* | (Mandatory) The neighbor router ID (string); a valid IP address. |
| *key_id* | (Mandatory) The key identifier; an integer from 1-255. |
| *key_type* | (Mandatory) The key type, MD5:1 or SHA-256:2 (integer); one of `1`, `2`. |
| *password* | (Mandatory) The encrypted MD5 or SHA-256 password (string). |

Returns

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

## *python_set_ospf_virtual_unset_config()*

Unset the list of configurations for the virtual-link.

### Syntax

`python_set_ospf_virtual_unset_config`(*<area_id>*,*<nbr_router_id>*, *<unset_list>*)

where:

| Variable | Description |
|---|---|
| *area_id* | (Mandatory) The area ID (string); a valid IP address. |
| *nbr_router_id* | (Mandatory) The neighbor router ID (string); a valid IP address. |
| *unset_list* | (Mandatory) Displays a list of strings:<br>• `auth_key`<br>• `auth_type`<br>• `hello_interval`<br>• `dead_interval`<br>• `transmit_delay`<br>• `retransmit_interval` |

### Returns

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

## *python_set_ospf_virtual_unset_message_digest_key_config()*

Unset the MD5/SHA-256 key configuration for virtual-link.

### Syntax

`python_set_ospf_virtual_unset_message_digest_key_config` (*<area_id>*,*<nbr_router_id>*,*<key_id>*)

where:

| Variable | Description |
|---|---|
| *area_id* | (Mandatory) The area ID (string); a valid IP address. |
| *nbr_router_id* | (Mandatory) The neighbor router ID (string); a valid IP address. |
| *key_id* | (Mandatory) The MD5 or SHA-256 key to unset (string); an integer from 1-255. |

Returns

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

## *python_ospf_get_proc_info()*

Gets the OSPF process information.

### Syntax

`python_ospf_get_proc_info`(*<vrf_name>*)

where:

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

### Returns

*<dict_ospfProcInfo>* returns a dictionary containing the following OSPF processes:

| Element | Description |
|---|---|
| *ospfId* | The OSPF process identifier (integer). Default value: `0`. |
| *routerId* | Router ID in IP address format (string); a valid IP address. |

| Element | Description |
|---|---|
| *uptime* | The OSPF process uptime; a string in the following format: HH:MM:SS. |
| *vfrName* | Default VRF name (string).<br>Default value: `default.` |
| *abrType* | The ABR type (integer); displays the valid ABR types. |
| *spfStartDelaySec* | The SPF schedule start delay, in seconds;<br>an integer from 0-600. |
| *spfStartDelayUsec* | The SPF schedule start delay, in microseconds;<br>an integer from 0-1000. |
| *spfMinDelaySec* | The minimum SPF schedule delay time;<br>an integer from 1-600. |
| *spfMinDelayUsec* | The minimum SPF schedule delay time, in microseconds; an integer from 1-1000. |
| *lsdbCount* | The number of external LSAs; zero or a positive integer. |
| *lsdbChecksum* | LAS checksum value (integer). |
| *lsdbOverflow* | The number of LSAs exceeding the limit; zero or a positive integer. |
| *originateNewLsas* | The number of new originated LSAs; zero or a positive integer. |
| *rxNewLsas* | The number of new LSAs received; zero or a positive integer. |
| *distance_all* | The distance to all destinations; zero or a positive integer. |
| *distance_intra* | The distance to intra-area destinations; zero or a positive integer. |
| *distance_inter* | The distance to inter-area destinations; zero or a positive integer. |
| *distance_external* | The distance to external destinations; zero or a positive integer. |
| *auth_type* | The type of authentication (integer);<br>one of `Null`, `zero` or `cryptographic`. |
| *mode* | The IS area shortcut (an integer);<br>one of `Shortcut`, `none`. |
| *area_id* | The area ID; an integer from 0-4294967295 |
| *area_type* | The area type (integer);<br>one of `Default`, `stub` or `nssa`. |

| Element | Description |
|---|---|
| *active_if_count* | The number of active interfaces in an area; zero or a positive integer. |
| *area_if_count* | The number of interfaces in an area; zero or a positive integer. |
| *full_virt_nbr_count* | Virtual neighbors count; zero or a positive integer. |
| *full_nbr_count* | Total number of neighbors; zero or a positive integer. |
| *spf_calc_count* | The number of SPF calculations; zero or a positive integer. |
| *area_lsdb_count* | The number of LSAs in the area; zero or a positive integer. |
| *area_lsbd_checksum* | The valid checksum of the link state database; a positive integer. |

## *python_ospf_get_multiarea_info()*

Gets OSPF multi-area neighbor information.

### Syntax

`python_ospf_get_multiarea_info`(<*vrf_name*>)

where:

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

### Returns

<*dict_ospf_multiarea*> returns a dictionary containing the following OSPF processes:

| Element | Description |
|---|---|
| *ifName* | The interface name (string).<br>For example: *Ethernet1/X* or *VLAN interface*. |
| *nbr_addr* | Neighbor IP address (string); a valid IP address. |
| *type* | The area type (integer);<br>one of `Default`, `stub` or `nssa`. |
| *ifIpAddress* | The interface IP address (string); a valid IP address. |
| *ifAreaId* | The interface area ID; an integer from 0-4294967295. |
| *ifMTU* | The maximum transmission unit (integer). |

| Element | Description |
|---|---|
| *proc_id* | The OSPF process identifier (integer).<br>Default value: `0`. |
| *ifRouterId* | The router ID in IP address format (string);<br>a valid IP address. |
| *ifNetworkType* | The interface network type (integer).<br>Default value: `Point-to-Point`. |
| *if_output_cost* | Interface output cost; zero or a positive integer. |
| *if_transmit_delay* | The interface transmit delay, in seconds (integer). |
| *Transmit_if_state* | The interface state type (integer). |
| *d_router* | Designated router ID (string); a valid IP address. |
| *d_router_address* | The IP address for the designated router (string). |
| *bd_router* | The backup router ID for the designated router (string); a valid IP address. |
| *bd_router_address* | The backup address for the designated router. (string). |
| *hello_interval* | The hello interval, in seconds (integer). |
| *dead_interval* | The dead interval, in seconds (integer). |
| *retransmit_interval* | The retransmit interval, in seconds (integer). |
| *neighbor_count* | The number of multi-area adjacent neighbors (integer). |

## *python_ospf_get_ribcounters_info()*

Gets OSPF rib counter information.

### Syntax

`python_ospf_get_ribcounters_info(`*<vrf_name>*

where:

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

### Returns

*<dict_ospf_ribcounter>* returns a dictionary containing the following processes:

| Element | Description |
|---|---|
| *ospf2rib_route_add* | The OSPF route addition calls made to RIB (integer). |
| *ospf2rib_route_add_error* | The OSPF to RIB route addition call errors (integer). |
| *ospf2rib_route_delete* | The OSPF to RIB route deletion calls (integer). |
| *ospf2rib_route_delete_error* | The OSPF to RIB route deletion call errors (integer). |
| *ospf2rib_route_add* | The OSPF route addition calls (integer). |
| *ospf2rib_route_dels* | The OSPF route deleted calls (integer). |
| *ospf_route_adds_ignored* | The OSPF ignored route addition calls (integer). |
| *ospf_route_dels_ignored* | The OSPF ignored route deleted calls (integer). |
| *rib2ospf_route_add* | The number of route additional calls from RIB to OSPF (integer). |
| *rib2ospf_route_add_error* | The number of route additional call errors from RIB to OSPF (integer). |
| *rib2ospf_route_del* | The number of route deleted calls from RIB to OSPF (integer). |
| *rib2ospf_route_del_error* | The number of route deleted call errors from RIB to OSPF (integer). |

## *python_ospf_get_redist_config()*

Gets OSPF redistribute configuration.

### Syntax

`python_ospf_get_redist_config`(*<vrf_name>*

where:

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

## Returns

<*dict_ospf_redist*> returns a dictionary containing the following processes:

| Element | Description |
|---|---|
| *redist_direct* | Redistribute the direct configuration (string); one of `Enable`, `Disable`. |
| *direct_metric* | Redistribute the direct cost; an integer from 0-16777214. |
| *direct_metric_type* | The external metric type (integer); one of `1`,`2`. |
| *direct_tag* | The tag value; an integer from 0-4294967295. |
| *direct_rmap_name* | The route-map name (string). |
| *redist_bgp* | Whether redistribute BGP is enabled; one of `Enable`, `Disable`. |
| *bgp_metric* | Redistribute BGP cost; an integer from 0-16777214. |
| *bgp_metric_type* | The external metric type (integer); one of `1`,`2.` |
| *bgp_tag* | The BGP tag value; an integer from 0-4294967295. |
| *bgp_rmap_name* | The BGP route map name (string). |
| *redist_static* | Whether redistribute static is enabled (string); one of `Enable`, `Disable`. |
| *static_metric* | Redistribute static cost; an integer from 0-16777214. |
| *static_metric_type* | The external metric type (integer); one of `1`,`2.` |
| *static_tag* | The tag value; an integer from 0-4294967295. |
| *static_rmap_name* | The static route map name (string). |

## *python_ospf_put_redist_config()*

Sets OSPF redistribute configuration.

### Syntax

`python_ospf_put_redist_config`(*<dict_redist>*,*<vrf_name>*

where:

| Variable | Description |
|---|---|
| *dict_redist* | (Mandatory) The dictionary containing the OSPF redistribute configuration details. |
| *vrf_name* | (Optional) Virtual Routing and Forwarding name; one of the VRF name or "`default`".<br>Default value: `default`. |

The *dict_redist* dictionary contains the following variables:

| Variable | Description |
|---|---|
| *redist_direct* | (Mandatory) Redistribute the direct configuration (string); one of `Enable`, `Disable`. |
| *redist_static* | (Mandatory) Whether redistribute static is enabled (string); one of `Enable`, `Disable`. |
| *redist_bgp* | (Mandatory) Whether redistribute BGP is enabled; one of `Enable`, `Disable`. |
| *direct_metric* | Redistribute the direct cost;<br>an integer from 0-16777214. |
| *direct_metric_type* | The external metric type (integer); one of `1,2`. |
| *direct_tag* | The tag value; an integer from 0-4294967295. |
| *direct_rmap_name* | The route-map name (string). |
| *bgp_metric* | Redistribute BGP cost; an integer from 0-16777214. |
| *bgp_metric_type* | The external metric type (integer); one of `1,2.` |
| *bgp_tag* | The BGP tag value; an integer from 0-4294967295. |
| *bgp_rmap_name* | The BGP route map name (string). |
| *static_metric* | Redistribute static cost; an integer from 0-16777214. |
| *static_metric_type* | The external metric type (integer); one of `1,2.` |
| *static_tag* | The tag value; an integer from 0-4294967295. |
| *static_rmap_name* | The static route map name (string). |

### Returns

`Success` or `Error`.

## *python_ospf_get_nssa_config()*

Shows the OSPF NSSA area configuration.

### Syntax

`python_ospf_get_nssa_config`(*<vrf_name>*

where:

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

### Returns

A dictionary containing the following elements:

| Element | Description |
|---|---|
| *nssa_area* | The NSSA area configuration (string); one of `Enable`, `Disable`. |
| *nssa_area_id* | The NSSA area ID IP address (string); a valid IP address. |
| *nssa_def_info* | The NSSA default information originate configuration (string); one of `Enable`, `Disable`. |
| *nssa_def_metric* | The NSSA default metric; an integer from 0-16777214. |
| *nssa_def_metric_type* | The NSSA external metric type (integer); one of `1`,`2`. |
| *nssa_no_redist* | Whether to stop redistribution in the NSSA area (string); one of `Enable`, `Disable.` |
| *nssa_no_summary* | Whether to stop summary LSAs into the NSSA area (string); one of `Enable`, `Disable.` |
| *nssa_translate_always* | `Always translate type7 LSA` (string); one of `Enable`, `Disable.` |
| *nssa_stability_interval* | The NSSA stability interval; an integer from 0-2147483647. |

## *python_ospf_put_nssa_config()*

Sets the OSPF NSSA area configuration.

### Syntax

`python_ospf_put_nssa_config(`*<dict_nssa>*,*<vrf_name>*

where:

| Variable | Description |
|---|---|
| *dict_nssa* | (Mandatory) The dictionary containing the OSPF NSSA configuration details. |
| *vrf_name* | (Optional) Virtual Routing and Forwarding name; one of the VRF name or "`default`". Default value: `default`. |

The *dict_nssa* dictionary contains the following variables:

| Variable | Description |
|---|---|
| *nssa_area* | (Mandatory) The NSSA area configuration (string); one of `Enable`, `Disable`. |
| *nssa_area_id* | (Mandatory) The NSSA area ID IP address (string); a valid IP address. |
| *nssa_def_info* | The NSSA default information originate configuration (string); one of `Enable`, `Disable`. |
| *nssa_def_metric* | The NSSA default metric; an integer from 0-16777214. |
| *nssa_def_metric_type* | The NSSA external metric type (integer); one of `1`,`2`. |
| *nssa_no_redist* | Whether to stop redistribution in the NSSA area (string); one of `Enable`, `Disable.` |
| *nssa_no_summary* | Whether to stop summary LSAs into the NSSA area (string); one of `Enable`, `Disable.` |
| *nssa_translate_always* | `Always translate type7 LSA` (string); one of `Enable`, `Disable.` |
| *nssa_stability_interval* | The NSSA stability interval; an integer from 0-2147483647. |

### Returns

`Success` or `Error`.

## *python_ospf_put_ area_def_cost_config()*

Sets the OSPF area default cost configuration.

### Syntax

`python_ospf_put_area_def_cost_config`(*<dict_def_cost>*,*<vrf_name>*

where:

| Variable | Description |
|---|---|
| *dict_def_cost* | (Mandatory) The dictionary containing the default cost configuration details. |
| *vrf_name* | (Optional) Virtual Routing and Forwarding name; one of the VRF name or "`default`". Default value: `default`. |

The *dict_def_cost* dictionary contains the following variables:

| Variable | Description |
|---|---|
| *area_id* | (Mandatory) The area ID IP address (string); a valid IP address. |
| *state* | Whether the default cost is enabled (string); one of `Enable`, `Disable.` |
| *default_cost* | The default summary cost value; an integer from 0-16777214. |

### Returns

`Success` or `Error`.

## *python_ospf_put_area_auth_config()*

Sets area authentication configuration.

### Syntax:

`python_ospf_put_area_auth_config`(*<dict_auth>*,*<vrf_name>*)

where:

| Variable | Description |
|---|---|
| *dict_auth* | (Mandatory) The dictionary containing the authentication configuration details. |
| *vrf_name* | (Optional) Virtual Routing and Forwarding name; one of the VRF name or "`default`". Default value: `default`. |

The *dict_auth* dictionary contains the following variables:

| Variable | Description |
|---|---|
| *area_id* | The area ID IP address (string); a valid IP address. |
| *auth* | The authentication configuration (string); one of `Enable`, `Disable.` |
| *auth-type* | The type of authentication (integer); one of `0-null`, `1-simple` or 2- `cryptographic`. |

### Returns

`Success` or `Error`.

## *python_ospf_put_summary_addr_config()*

Sets summary address configuration.

### Syntax

`python_ospf_put_summary_addr_config`(*<dict_summary>,<vrf_name>*)

where:

| Variable | Description |
|---|---|
| *dict_summary* | (Mandatory) The dictionary containing the authentication address configuration details. |
| *vrf_name* | (Optional) Virtual Routing and Forwarding name; one of the VRF name or "`default`". Default value: `default`. |

The *dict_summary* dictionary contains the following variables:

| Variable | Description |
|---|---|
| *summary_addr* | Whether to enable summary address configuration (string); one of `Enable`, `Disable.` |
| *prefix* | The IP address (string); a valid IP address. |
| *masklen* | The mask length; an integer from 0-32. |
| *not-advertise* | Whether to suppress routes that match the prefix (string); one of `Enable`, `Disable.` |
| *tag* | The tag value; an integer from 0-4294967295. |

### Returns

`Success` or `Error.`

## *python_ospf_put_area_range_config()*

Sets OSPF area range configuration.

### Syntax

`python_ospf_put_area_range_config`(*<dict_range>*,*<vrf_name>*)

where:

| Variable | Description |
|---|---|
| *dict_range* | (Mandatory) The dictionary containing the area range configuration details. |
| *vrf_name* | (Optional) Virtual Routing and Forwarding name; one of the VRF name or "`default`".<br>Default value: `default`. |

The *dict_range* dictionary contains the following variables:

| Variable | Description |
|---|---|
| *area_id* | The area ID IP address (string); a valid IP address. |
| *range* | Whether to enable the range of IP addresses (string); one of `Enable`, `Disable.` |
| *prefix* | The IP address (string); a valid IP address. |
| *mask* | The mask length; an integer from 0-32. |
| *not-advertise* | Whether to suppress routes that match the prefix (string); one of `Enable`, `Disable.` |

### Returns

`Success` or `Error.`

## *python_ospf_put_overflow_db_config()*

Sets overflow database configuration.

### Syntax

`python_ospf_put_overflow_db_config`(<*dict_overflow*>,<*vrf_name*>)

where:

| Variable | Description |
|---|---|
| *dict_overflow* | (Mandatory) The dictionary containing the overflow database configuration details. |
| *vrf_name* | (Optional) Virtual Routing and Forwarding name; one of the VRF name or "`default`".<br>Default value: `default`. |

The *dict_overflow* dictionary contains the following variables:

| Variable | Description |
|---|---|
| *db_overflow* | Whether to enable the database overflow configuration (string); one of `Enable`, `Disable`. |
| *max_lsas* | The maximum LSA limit;<br>an integer from 0-4294967294. |
| *limit* | The database limit type (string);<br>one of `Hard`, `soft`. |
| *external* | Whether to enable the external LSA limit (string);<br>one of `Enable`, `Disable`. |
| *ext_max_lsas* | The maximum external LSA limit;<br>an integer from 0-2147483647. |
| *recovery_time* | Tme time to recover from the external LSA limit, in seconds; an integer from 0-65535. |

### Returns

`Success` or `Error.`

## *python_ospf_put_refbw_config()*

Sets the autocost reference bandwith configuration.

### Syntax

`python_ospf_put_refbw_config`(*<dict_refbw>*,*<vrf_name>*)

where:

| Variable | Description |
|---|---|
| *dict_refbw* | (Mandatory) The dictionary containing the autocost reference bandwidth configuration details. |
| *vrf_name* | (Optional) Virtual Routing and Forwarding name; one of the VRF name or "`default`".<br>Default value: `default`. |

The *dict_refbw* dictionary contains the following variables:

| Variable | Description |
|---|---|
| *autocost_refbw* | Whether to enable the autocost reference bandwidth configuration (string);<br>one of `Enable`, `Disable`. |
| *bw_value* | The bandwidth value; an integer from:<br>• 1-4294 for Gbps<br>• 1-4294967 for Mbps |
| *bw_unit* | The bandwidth unit (string); one of `Gbps`, `Mbps`. |

### Returns

`Success` or `Error.`

## *python_ospf_put_stub_config()*

Sets the stub area configuration.

### Syntax

`python_ospf_put_stub_config`(*<dict_stub>*,*<vrf_name>*)

where:

| Variable | Description |
|---|---|
| *dict_stub* | (Mandatory) The dictionary containing the stub area configuration details. |
| *vrf_name* | (Optional) Virtual Routing and Forwarding name; one of the VRF name or "`default`".<br>Default value: `default`. |

The *dict_stub* dictionary contains the following variables:

| Variable | Description |
|---|---|
| *area-id* | The area ID IP address (string); a valid IP address. |
| *stub* | Whether to enable the stub area configuration (string); one of `Enable`, `Disable`. |
| *no_summary* | Whether to not inject summary routes into the stub configuration (string); one of `Enable`, `Disable`. |

### Returns

`Success` or `Error`.

## *python_ospf_put_clear_config()*

Sets the remove commands for process, statistics, traffic statistics and neighbors.

### Syntax

`python_ospf_put_clear_config`(*<dict_clear_cfg>*,*<vrf_name>*)

where:

| Variable | Description |
|---|---|
| *dict_clear_cfg* | (Mandatory) The dictionary containing the remove commands details. |
| *vrf_name* | (Optional) Virtual Routing and Forwarding name; one of the VRF name or "`default`".<br>Default value: `default`. |

The *dict_clear-cfg* dictionary contains the following variables:

| Variable | Description |
|---|---|
| *process* | Remove the OSPF process configurations (string); one of `Enable`, `Disable`. |
| *statistics* | Remove the OSPF statistic configurations (string); one of `Enable`, `Disable`. |
| *traffic* | Remove the OSPF process traffic statistic configurations (string); one of `Enable`, `Disable`. |
| *neighbors* | Remove the OSPF neighbor configurations (string); one of `Enable`, `Disable`. |

### Returns

`Success` or `Error`.

## *python_ospf_put_procinfo_config()*

Sets the OSPF process information.

### Syntax

`python_ospf_put_procinfo_config`(<*dict_procinfo*>,<*vrf_name*>)

where:

| Variable | Description |
|---|---|
| *dict_procinfo* | (Mandatory) The dictionary OSPF process information details. |
| *vrf_name* | (Optional) Virtual Routing and Forwarding name; one of the VRF name or "`default`".<br>Default value: `default`. |

The *dict_procinfo* dictionary contains the following variables:

| Variable | Description |
|---|---|
| *routerId* | The OPSF router ID in IP address format (string); a valid IP address. |
| *defaultMetric* | The default metric cost;<br>an integer from 1-16777214. |
| *distance_all* | The administrative distance, an integer from 1-255. |
| *bfd* | Whether to enable BFD configuration (string);<br>one of `Enable`, `Disable`. |
| *shutdown* | Whether to enable the shutdown OSPF process (string); one of `Enable`, `Disable`. |

### Returns

`Success` or `Error.`

# 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()*

Gets the properties of one interface.

#### Syntax

`get_interface(`*<if_name>*`)`

where:

| Variable | Description |
|---|---|
| *if_name* | (Optional) The Ethernet interface name (string).<br>Default value: `none`.<br>**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`. |

| Element | Description |
|---|---|
| *mac_addr* | The MAC address in the following format: xxxx.xxxx,xxxx (string). |
| *speed* | The communication speed of the interface (string); one of:<br>• `auto` (auto negotiate)<br>• `10` (10Mb/s)<br>• `100` (100Mb/s)<br>• `1000` (1Gb/s)<br>• `10000` (10Gb/s)<br>• `40000` (40Gb/s).<br>**Note:** Values can vary based on system configuration. |

## *get_mac()*

Gets the MAC address of a physical port.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The Ethernet interface name (string).<br>**Note:** The interface must exist. |

### Returns

The MAC address:

| Element | Description |
|---|---|
| *mac_addr* | The MAC address in the following format: xxxx.xxxx,xxxx (string). |

## *set_mac()*

Sets the MAC address of the specified interface.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The Ethernet interface name (string).<br>**Note:** The interface must exist. |
| *mac_addr* | The MAC address in the following format: xxxx.xxxx,xxxx (string). |

### Returns

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

## *is_enabled()*

Checks whether the port is enabled

### Syntax

`is_enabled(<if_name>)`

where:

| Variable | Description |
|---|---|
| *if_name* | The Ethernet interface name (string).<br>**Note:** The interface must exist. |

### Returns

Enabled status:

| Element | Description |
|---|---|
| *is_enabled* | Port enabled status (string); one of `up`, `down`. |

## *set_enabled()*

Enables or disables the specified port.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The Ethernet interface name (string).<br>**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()*

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

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The Ethernet interface name (string).<br>**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()*

Gets the speed of the specified port.

### Syntax

`get_port_speed(`*<if_name>*`)`

where:

| Variable | Description |
|---|---|
| *if_name* | The Ethernet interface name (string).<br>**Note:** The interface must exist. |

### Returns

The port speed:

| Element | Description |
|---|---|
| *speed* | The communication speed of the interface (string); one of:<br>• `auto` (auto negotiate)<br>• `10` (10Mb/s)<br>• `100` (100Mb/s)<br>• `1000` (1Gb/s)<br>• `10000` (10Gb/s)<br>• `40000` (40Gb/s).<br>**Note:** Values can vary based on system configuration. |

## *set_port_speed()*

Sets the speed of the specified port.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The Ethernet interface name (string).<br>**Note:** The interface must exist. |
| *speed* | The communication speed of the interface (string); one of:<br>• `auto` (auto negotiate)<br>• `10` (10Mb/s)<br>• `100` (100Mb/s)<br>• `1000` (1Gb/s)<br>• `10000` (10Gb/s)<br>• `40000` (40Gb/s).<br>**Note:** Values can vary based on system configuration. |

### Returns

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

## *get_duplex()*

Gets the duplex for the port.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The Ethernet interface name (string).<br>**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()*

Sets the duplex for the port.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The Ethernet interface name (string).<br>**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()*

Gets the port statistics for the specified interface.

### Syntax

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

where:

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

### Returns

The following packet statistics:

| Element | Description |
|---|---|
| *good_pkts_sent* | Number of sent packets (integer). |
| *in_un_pkts* | Number of received unicast packets (integer). |
| *bad_crc* | Number of bad CRCs (integer). |
| *brdc_pkts_rcv* | Number of received broadcast packets (integer). |
| *mc_pkts_sent* | Number of sent multicast packets (integer). |
| *undersize_pkts* | Number of undersise packets (integer). |
| *mc_pkts_rcv* | Number of received multicast packets (integer). |
| *in_discards* | Number of discarded in packets (integer). |
| *good_octets_rcv* | Number of received octets (integer). |
| *oversize_pkts* | Number of oversize packets (integer). |
| *brdc_pkts_sent* | Number of sent broadcast packets (integer). |
| *good_octets_sent* | Number of sent octets (integer). |
| *out_uc_pkts* | Number of sent unicast packets (integer). |
| *good_pkts_rcv* | Number of received packets (integer). |

## *clear_stats*

Resets statistics for the specified switch interface.

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

### Syntax

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

where:

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

### Returns

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

# Private VLAN Module

The following class configures Private VLAN properties. To use this module, in the Python file or in the Python interpreter, enter:

```
import pvlanApi
```

## class Pvlan()

This class has methods for getting and setting Private VLAN configurations.

### *manageGlobalPvlan()*

Globally enables or disables the private VLAN.

#### Syntax

`manageGlobalPvlan(`*<enable>*`)`

where:

| Variable | Description |
|---|---|
| *enable* | (Mandatory) The global status of the private VLAN; one of `true` to enable or `false` to disable. |

#### Returns

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

### *managePvlan()*

Creates or deletes a private VLAN.

#### Syntax

`managePvlan(`*<pvlanType>*`,`*<vlanID>*`,`*<createPvlan>*`)`

where:

| Variable | Description |
|---|---|
| *pvlanType* | (Mandatory) The private VLAN type (string); one of `primary`, `community`, `isolated`. |
| *vlanID* | (Mandatory) The VLAN value; an integer from 2-4093. |
| *createPvlan* | (Mandatory) Whether the private VLAN is created or deleted; one of `true`, to create the private VLAN or `false`, to delete the private VLAN. |

#### Returns

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

## *managePvlanAssoc()*

Creates or deletes a private VLAN association.

### Syntax

`managePvlanAssoc(`*<primaryVlanID>*,*<secondaryVlanID>*,*<createAssoc>*`)`

where:

| Variable | Description |
|---|---|
| *primaryVlanID* | (Mandatory) The primary VLAN value; an integer from 2-4093. |
| *secondaryVlanID* | (Mandatory) The secondary VLAN value; an integer from 2-4093. |
| *createAssoc* | (Mandatory) Whether the private VLAN association is created or deleted; one of `true`, to create the private VLAN association or `false`, to delete the private VLAN association. |

### Returns

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

## *setPortMode()*

Enables or disables the private VLAN mode on an interface.

### Syntax

`setPortMode(`*<ifType>*,*<chassisNumber>*,*<ifNumber>*,*<ifSubNumber>*,*<enable>*`)`

where:

| Variable | Description |
|---|---|
| *ifType* | (Mandatory) The interface type (string); one of `ethernet`, for Ethernet interfaces, `po` for port-channel interfaces. |
| *chassisNumber* | (Mandatory) The chassis number for the Ethernet interfaces (integer). For example: `1` for Ethernet1/2, or `none` for port-channel interfaces. |
| *ifNumber* | (Mandatory) The interface number for port-channel interfaces or Ethernet interfaces (integer). For example: `1` for port-channel 1 interface, `2` for Ethernet 1/2. |

| Variable | Description |
|---|---|
| *IfSubNumber* | (Mandatory) The sub-interface number for the Ethernet interfaces (integer). For example: `4` for Ethernet1/50/3, or `none` for port-channel interfaces. |
| *enable* | (Mandatory) The status of the private VLAN mode; one of `true`, to enable the private VLAN mode or `false` to disable the private VLAN mode. |

### Returns

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

## *managePortMapping()*

Creates or removes a Private VLAN port-mapping.

### Syntax

`managePortMapping(`*<ifType>*,*<chassisNumber>*,*<ifNumber>*,*<ifSubNumber>*, *<primaryVlanID>*,*<enable>*`)`

where:

| Variable | Description |
|---|---|
| *ifType* | (Mandatory) The interface type (string); one of `ethernet`, for Ethernet interfaces, `po` for port-channel interfaces. |
| *chassisNumber* | (Mandatory) The chassis number for the Ethernet interfaces (integer). For example: `1` for Ethernet1/2, or `none` for port-channel interfaces. |
| *ifNumber* | (Mandatory) The interface number for port-channel interfaces or Ethernet interfaces (integer). For example: `1` for port-channel 1 interface, `2` for Ethernet 1/2. |
| *IfSubNumber* | (Mandatory) The sub-interface number for the Ethernet interfaces (integer). For example: `4` for Ethernet1/50/3, or `none` for port-channel interfaces. |
| *primaryVlanID* | (Mandatory) The primary VLAN value; an integer from 2-4093. |
| *enable* | (Mandatory) Whether the private VLAN mapping is created or removed. One of `true`, to map the private VLAN or `false` to remove the private VLAN mapping. |

### Returns

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

## *managePortAssociation()*

Creates or removes a private VLAN association on an interface.

### Syntax

`managePortAssociation(`*<ifType>*,*<chassisNumber>*,*<ifNumber>*,*<ifSubNumber>*, *<primaryVlanID>*,*<secondaryVlanID>*,*<enable>*`)`

where:

| Variable | Description |
|---|---|
| *ifType* | (Mandatory) The interface type (string); one of `ethernet`, for Ethernet interfaces, `po` for port-channel interfaces. |
| *chassisNumber* | (Mandatory) The chassis number for the Ethernet interfaces (integer). For example: `1` for Ethernet1/2, or `none` for port-channel interfaces. |
| *ifNumber* | (Mandatory) The interface number for port-channel interfaces or Ethernet interfaces (integer). For example: `1` for port-channel 1 interface, `2` for Ethernet 1/2. |
| *IfSubNumber* | (Mandatory) The sub-interface number for the Ethernet interfaces (integer). For example: 4 for Ethernet1/50/3, or `none` for port-channel interfaces. |
| *primaryVlanID* | (Mandatory) The primary VLAN value; an integer from 2-4093. |
| *secondaryVlanID* | (Mandatory) The secondary VLAN value; an integer from 2-4093. |
| *enable* | (Mandatory) Whether the private VLAN association is created or removed. One of `true`, to create the private VLAN or `false` to remove the private VLAN association. |

### Returns

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

## *showPvlanInfo()*

Displays the private VLAN information.

### Syntax

```
showPvlanInfo
```

### Returns

A dictionary containing private VLAN information:

| Element | Description |
|---|---|
| *vlanID* | The VLAN ID; an integer from 2-4093. |
| *type* | The private VLAN type (string);<br>one of `primary`, `community`, `isolated`. |
| *primaryVlanID* | The primary VLAN ID; an integer from 2-4093. Available only for community and isolated private VLANs. |

## *showPvlanInterfaceInfo()*

Displays information related to the Private VLAN interface.

### Syntax

```
showPvlanInterfaceInfo
```

### Returns

A dictionary containing private VLAN interface information:

| Element | Description |
|---|---|
| *ifName* | The interface name (string). |
| *portMode* | The interface port mode (string);<br>one of `trunk`, `access`. |
| *pvlanPortMode* | The private VLAN port mode (string);<br>one of `host`, `promiscuous`, `configured`. |
| *vlan* | A list of private VLAN IDs configured on the port; an integer from 2-4093. |

# RADIUS Module

The class 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 (string). |
| *key_form* | (Optional) The encryption method of the authentication key (integer); one of `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; an integer from 0-5. |

### Returns

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

## *get_global_timeout*

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

### Syntax

```
get_global_timeout()
```

### Returns

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; an integer from 1-60. |

### Returns

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

## *get_host*

Displays information about an already configured RADIUS server.

### Syntax

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

where:

| 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;<br>an integer from 1-65535. |
| *acct-port* | The port used for accounting;<br>an integer from 1-65535. |
| *retransmit* | The number of retries the switch will make to establish a connection with a RADIUS server after the initial attempt failed; an integer from 0-5. |

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

## *get_all_hosts*

Displays information about all configured RADIUS servers.

### Syntax

```
get_all_hosts()
```

### Returns

A list of dictionaries, each showing RADIUS server information:

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

### Returns

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

## *delete_host*

Deletes an already configured RADIUS server.

### Syntax

`delete_host(<ip_addr>)`

where:

| 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); one of 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); one of 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).<br>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()*

Adds a static IPv4 route for a subnet mask.

#### Syntax

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

where:

| Variable | Description |
|---|---|
| *ip_addr* | The IPv4 address of the route. |
| *ip_prefix_len* | The IP prefix length;<br>an integer from 0-32, with 0 being the default gateway. |
| *ip_gw* | The gateway IP address. |
| *dist* | The distance value for the route;<br>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).<br>**Note:** The named VRF must exist. |
| *if_name* | Interface name used for communication (string).<br>**Note:** The interface must exist. |

#### Returns

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

## *delete_route()*

Deletes a static IPv4 route.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *ip_addr* | The IPv4 address of the route. |
| *ip_prefix_len* | The IP prefix length;<br>an integer from 0-32, with 0 being the default gateway. |
| *ip_gw* | The gateway IP address. |
| *dist* | The distance value for the route;<br>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).<br>**Note:** The named VRF must exist. |
| *if_name* | Interface name used for communication (string).<br>**Note:** The interface must exist. |

### Returns

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

## *get_route()*

Gets all IPv4 routes from the routing table.

### Syntax

`get_route(`*<vrf_name>*`,`*<ip_dest>*`,`*<ip_prefix_len>*`)`

where:

| Variable | Description |
|---|---|
| *vrf_name* | The VRF name (string).<br>**Note:** The named VRF must exist. |
| *ip_dest* | (Optional) The destination IP address (string).<br>Default value: `None`. |
| *ip_prefix_len* | (Optional) The IP prefix length;<br>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;<br>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;<br>an integer from 0-32, with 0 being the default gateway. |
| *desc* | Description of the static route (string). |
| *vrf_name* | The VRF name (string). |
| *if_name* | Interface name used for communication (string). |

# Routemap Module

The class and function in this module manage Routemaps. To use this module, in the Python file or in the Python interpreter, enter:

```
import routemapApi
```

## class Routemap

This class provides a function for managing Routemaps.

### *get_all_routemap_entry()*

Gets configured routemaps on switch.

#### Syntax

```
get_all_routemap_entry()
```

#### Returns

The list of available routemaps.

# Security Mode Module

The class 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);<br>one of `secure_mode`, `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:<br>• `module`<br>• `air-flow`<br>• `speed-percentage`<br>• `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:<br>• `Name`<br>• `Manufacturer`<br>• `Model`<br>• `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: |
| *ambient* | A dictionary containing the following:<br>• `temp`: the temperature, in Celsius (integer)<br>• `state`: the power supply state; one of `OK`, `FAULT` |

| Element | Description |
|---|---|
| *hotspot* | A dictionary containing the following:<br>• `temp`: the temperature, in Celsius (integer)<br>• `state`: the power supply state; one of `OK`, `FAULT` |
| *temperature threshold* | A dictionary containing the following:<br>• `system warning`: the temperature at which a system warning is issued<br>• `system shutdown:` the temperature at which the system will automatically shut down<br>• `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;<br>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()*

Gets 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 class 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*

Checks 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* | (Optional) The type of TACACS+ encryption key (integer); one of: `0` (clear text password) or `7` (encrypted password).<br>**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);<br>one of: IPv4 address, IPv6 address or hostname. |
| *port* | The port used to connect to TACACS+ server;<br>an integer from 1-65535. |
| *key* | The status of the server encryption key configuration (string); one of `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+ server (string);<br>one of: IPv4 address, IPv6 address or hostname. |
| *port* | The port used to connect to TACACS+ server;<br>an integer from 1-65535. |
| *key* | The status of the server encryption key configuration (string); one of `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 address of the TACACS+ server (string); one of: IPv4 address, IPv6 address or hostname. |
| *port* | (Optional) The port used to connect to TACACS+ server; an integer from 1-65535. |
| *key* | (Optional) The server encryption key used to communicate with the TACACS+ server (string). |
| *key_form* | (Optional) The type of the server encryption key (string); one of: `0` (clear text password) or `7` (encrypted password).<br>**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 address of the TACACS+ server (string); one of: 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 283. |

## *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_CancelRequest()

The function in this class clears a prior Buffer Statistics Tracking (BST) request.

### *set_bst_cancel_request ()*

Cancels the periodic report timer and stops ongoing periodic requests.

#### Syntax

`set_bst_cancel_request(`<*dict_cancel_request*>`)`

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

| Element | Description |
|---|---|
| *req-id* | The unique request identifier (integer). |
| *cancel-req-id* | The request identifier for the cancellation request (integer). |

#### Returns

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

## class TelemetryBST_Tracking()

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

### *set_bst_tracking()*

Sets buffer statistics tracking parameters on the switch.

#### Syntax

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

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

| Element | Description |
|---|---|
| *track-peak-stats* | Set to 1 to enable peak statistics tracking, 0 to disable this feature. Default value: `0`. |
| *track-ingress-port-priority-group* | Set to 1 to enable ingress port priority group tracking, 0 to disable this feature. Default value: `1`. |
| *track-ingress-port-service-pool* | Set to 1 to enable ingress port service pool tracking, 0 to disable this feature. Default value: `1`. |
| *track-ingress-service-pool* | Set to 1 to enable ingress service pool tracking, 0 to disable this feature. Default value: `1`. |
| *track-egress-port-service-pool* | Set to 1 to enable egress port service pool tracking, 0 to disable this feature. Default value: `1`. |
| *track-egress-service-pool* | Set to 1 to enable egress service pool tracking, 0 to disable this feature. Default value: `1`. |
| *track-egress-rqe-queue* | Set to 1 to enable egress RQE queue tracking, 0 to disable this feature. Default value: `1`. |
| *track-egress-cpu-queue* | Set to 1 to enable egress CPU queue tracking, 0 to disable this feature. Default value: `1`. |
| *track-egress-uc-queue* | Set to 1 to enable egress unicast queue tracking, 0 to disable this feature. Default value: `1`. |
| *track-egress-mc-queue* | Set to 1 to enable egress multicast queue tracking, 0 to disable this feature. Default value: `1`. |
| *track-device* | Set to 1 to enable tracking of this device, 0 to disable this feature. Default value: `1`. |

### Returns

A dictionary containing the following elements:

| Element | Description |
|---|---|
| *return status* | Return status of the API call:<br>• TRUE: Successfully set the Buffer statistics.<br>• FALSE: Setting Buffer statistics failed |
| *error message* | String value containing the reason for the API failure:<br>• none: When Return Status is TRUE.<br>• error string: When Return Status is FALSE |

## *get_bst_tracking()*

Gets buffer statistics tracking parameters.

### Syntax

```
get_bst_tracking()
```

### Returns

A dictionary containing the following elements:

| Element | Description |
|---|---|
| *track-peak-stats* | Set to 1 to peak statistics tracking, 0 to disable this feature. Default value: 0. |
| *track-ingress-port-priority-group* | Set to 1 to enable ingress port priority group tracking, 0 to disable this feature.<br>Default value: 1. |
| *track-ingress-port-service-pool* | Set to 1 to enable ingress port service pool tracking, 0 to disable this feature.<br>Default value: 1. |
| *track-ingress-service-pool* | Set to 1 to enable ingress service pool tracking, 0 to disable this feature. Default value: 1. |
| *track-egress-port-service-pool* | Set to 1 to enable egress port service pool tracking, 0 to disable this feature.<br>Default value: 1. |
| *track-egress-service-pool* | Set to 1 to enable egress service pool tracking, 0 to disable this feature. Default value: 1. |
| *track-egress-rqe-queue* | Set to 1 to enable egress RQE queue tracking, 0 to disable this feature. Default value: 1. |
| *track-egress-cpu-queue* | Set to 1 to enable egress CPU queue tracking, 0 to disable this feature. Default value: 1. |

| Element | Description |
|---|---|
| *track-egress-uc-queue* | Set to 1 to enable egress unicast queue tracking, 0 to disable this feature. Default value: `1`. |
| *track-egress-mc-queue* | Set to 1 to enable egress multicast queue tracking, 0 to disable this feature. Default value: `1`. |
| *track-device* | Set to 1 to enable tracking of this device, 0 to disable this feature. Default value: `1`. |

## class TelemetryBST_Feature()

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

### *set_bst_feature()*

Sets buffer statistics and tracking feature parameters on the switch.

#### Syntax

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

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

| Element | Description |
|---|---|
| *bst-enable* | Set to 1 to enable BST, 0 to disable it. Enabling BST allows the switch to track buffer utilization statistics. Default value: `0`. |
| *send-async-reports* | Set to 1 to enable the transmission of periodic asynchronous reports, 0 to disable this feature. Default value: `0`. |
| *collection-interval* | The collection interval, in seconds. This defines how frequently periodic reports will be sent to the configured controller. 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. Default value: `1`. |
| *async-full-report* | Set to 1 to enable the async full report feature, 0 to disable it. Default value: `0.`<br>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:<br>• `TRUE`: Successfully set the Buffer statistics feature.<br>• `FALSE`: Setting Buffer statistics feature failed |
| *error message* | String value containing the reason for the API failure:<br>• none: When Return Status is `TRUE`.<br>• error string: When Return Status is `FALSE` |

## *get_bst_feature()*

Gets buffer statistics and tracking feature parameters.

### Syntax

```
get_bst_feature()
```

### Returns

A dictionary containing the following elements:

| Element | Description |
|---|---|
| *bst-enable* | Set to 1 to enable BST, 0 to disable it. Enabling BST allows the switch to track buffer utilization statistics. Default value: `0` |
| *send-async-reports* | Set to 1 to enable the transmission of periodic asynchronous reports, 0 to disable this feature. Default value: `0` |

| Element | Description |
|---|---|
| *collection-interval* | The collection interval, in seconds. This defines how frequently periodic reports will be sent to the configured controller;<br>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`;<br>an integer from 1-5. Default value: `1` |
| *trigger-rate-limit-interval* | The trigger rate limit interval, in seconds;<br>an integer from 10-60. Default value: `1`. |
| *send-snapshot-on-trigger* | Set to 1 to enable sending a complete snapshot of all buffer statistics counters when a trigger happens, 0 to disable this feature. Default value: `1`. |
| *async-full-report* | Set to 1 to enable the async full report feature, 0 to disable it. Default value: `0`.<br><br>When this feature is enabled, the agent sends full reports containing data related to all counters. When the feature is disabled, the agent sends incremental reports containing only the counters that have changed since the last report. |

## class TelemetryBST_ClearCgsnDrop

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

### *get_bst_clear_thresholds()*

Resets the congestion drop counters to the default values.

#### Syntax

```
get_bst_clear_thresholds
```

#### Returns

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

## class TelemetryBST_Cgsn_Drop_Ctr()

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

### *get_bst_cgsn_drop_ctr()*

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

#### Syntax

`get_bst_cgsn_drop_ctr(`*<dict_bst_cdropctr>*`)`

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

| Element | Description |
|---|---|
| *req-id* | The unique request identifier (integer). |
| *request-type* | One of the following:<br>• `top-drops`: Show ports with maximum congestion on the switch and their drop-counters<br>• `top-port-queue-drops`: Show top port-queue level drop-counters on the switch<br>• port-queue-drops: Show per port-queue level drop-counters on the switch<br>• `port-drops`: Show per-port total drop counters on the switch |
| *request-params* | Request parameters; one of the following:<br>• `count`: Number of ports required in the report. The ports are sorted with the port suffering maximum congestion at the top; an integer.<br>• `queue-type`: Filters the report on the queue type; one of the following strings:<br>– `ucast`: Unicast queues<br>– `mcast`: Multicast queues<br>– `all`: All supported queues<br>• `interface-list`: Comma-separated list of ports for the congestion drop counter report; an array. A value of all requests all the ports.<br>• `queue-list`: An array of queue numbers to be considered for the drop report.<br>• `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:<br>• `top-drops`: Show ports with maximum congestion on the switch and their drop-counters<br>• `top-port-queue-drops`: Show top port-queue level drop-counters on the switch<br>• port-queue-drops: Show per port-queue level drop-counters on the switch<br>• `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:<br>• `interface`: Interface name<br>• `ctr`: Counter value; a string.<br>• `queue-type`; one of ucast, mcast<br>• `queue-drop-ctr`; one of:<br>– `queue number`: an integer from 1-8.<br>– `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()*

Clears buffer statistics and tracking counters.

### Syntax

```
get_bst_clear_stats()
```

### Returns

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

## class TelemetryBST_ClearThresholds()

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

### *get_bst_clear_thresholds()*

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

#### Syntax

```
get_bst_clear_thresholds()
```

#### Returns

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

## class TelemetryBST_Limits()

The functions in this class provide access to the valid range of parameters used to configure the telemetry Buffer Statistics Tracking (BST) features.

### *get_bst_limits()*

Gets the limits associated to the BST realm indexes.

#### Syntax

```
get_bst_limits
```

#### Returns

A dictionary containing the following elements:

| Element | Description |
|---|---|
| *service-pool* | The service pool range;, an integer from 0-1. |
| *priority-group* | The priority group range; an integer from 0-7. |
| *user-queue* | The user queue range; an integer from 0-7. |
| *unicast-queue* | The absolute unicast queue range; an integer from 0-16383. |
| *multicast-queue* | The absolute multicast queue range; an integer from 0-2600. |
| *cpu-queue* | The CPU queue range; an integer from 0-47. |
| *rqe-queue* | The RQE queue range; an integer from 0-10. |
| *queue-group* | The queue group range; an integer from 0-127. |

## class TelemetryBST_Report()

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

### *get_bst_report()*

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

#### Syntax

`get_bst_report(`*<dict_get_bst_report>*`)`

#### Returns

*<dict_get_bst_report>* is a dictionary containing the following elements:

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

*<dict_get_bst_report>* is a dictionary containing the following elements

| Element | Description |
|---|---|
| *realm* | Identifies a group of buffer utilization statistic counters (string). One of:<br>• `ingress-port-priority-group`<br>• `ingress-port-service-pool`<br>• `ingress-service-pool`<br>• `egress-port-service-pool`<br>• `egress-service-pool`<br>• `egress-rqe-queue`<br>• `egress-cpu-queue`<br>• `egress-mc-queue`<br>• `egress-uc-queue`<br>• `device` |
| *interface* | The interface name (string).<br>For example: *Ethernet1/X*. |
| *service-pool* | The service pool range, from 0-1. |
| *priority-group* | The priority group range, from 0-7. |
| *user-queue* | The user queue range, from 0-7. |
| *unicast-queue* | The absolute unicast queue range, from 0-16383. |
| *multicast-queue* | The absolute multicast queue range, from 0-2600. |
| *cpu-queue* | The CPU queue range, from 0-47. |
| *rqe-queue* | The RQE queue range, from 0-10. |
| *queue-group* | The queue group range, from 0-127. |
| *threshold* | The threshold level, in percentage, from 1-100. |
| *um-share-threshold* | The threshold level for unicast and multicast shared buffer utilization, in percentage, from 1-100. |
| *uc-share-threshold* | The threshold level for unicast shared buffer utilization, in percentage, from 1-100. |
| *mc-share-threshold* | The threshold level for multicast shared buffer utilization expressed in percentage, from 1-100. |

## class TelemetryBST_Threshold()

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

### *set_bst_threshold()*

Configures the buffer statistics and tracking thresholds.

#### Syntax

`set_bst_threshold(`*<bst_threshold_cfg>*`)`

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

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

**Note:** The `egress-uc-queue` and `egress-mc-queue` thresholds can be configured using the absolute queue value. You can also specify the pair (interface, user-queue), where the user-queue is a number from 0-7. The following table contains the parameters that can be used as indexes for each realm:

| Element | Description |
|---|---|
| *realm* | Identifies a group of buffer utilization statistic counters (string). One of:<br>• `ingress-port-priority-group`<br>• `ingress-port-service-pool`<br>• `ingress-service-pool`<br>• `egress-port-service-pool`<br>• `egress-service-pool`<br>• `egress-rqe-queue`<br>• `egress-cpu-queue`<br>• `egress-mc-queue`<br>• `egress-uc-queue`<br>• `device` |
| *interface* | The interface name (string).<br>For example: *Ethernet1/X*. |
| *service-pool* | The service pool range, from 0-1. |
| *priority-group* | The priority group range, from 0-7. |
| *user-queue* | The user queue range, from 0-7. |
| *unicast-queue* | The absolute unicast queue range, from 0-16383. |
| *multicast-queue* | The absolute multicast queue range, from 0-2600. |
| *cpu-queue* | The CPU queue range, from 0-47. |
| *rqe-queue* | The RQE queue range, from 0-10. |
| *queue-group* | The queue group range, from 0-127. |
| *threshold* | The threshold level, in percentage, from 1-100. |
| *um-share-threshold* | The threshold level for unicast and multicast shared buffer utilization, in percentage, from 1-100. |
| *uc-share-threshold* | The threshold level for unicast shared buffer utilization, in percentage, from 1-100. |
| *mc-share-threshold* | The threshold level for multicast shared buffer utilization expressed in percentage, from 1-100. |

## Returns

A dictionary containing the following elements:

| Element | Description |
|---|---|
| *return status* | Return status of the API call:<br>• TRUE: Successfully set the Buffer statistics.<br>• FALSE: Setting Buffer statistics failed |
| *error message* | String value containing the reason for the API failure:<br>• none: When Return Status is TRUE.<br>• error string: When Return Status is FALSE. |

## *get_bst_threshold()*

Gets buffer statistics and tracking threshold values.

### Syntax

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

### Returns

*<dict_get_bst_threshold>* is a dictionary containing the following elements:

| Realm | Index # 1 | Index # 2 | Statistics |
|---|---|---|---|
| `ingress-port-priority-group` | *interface* (such as `Ethernet1/7`) | `priority-group` | `um-share-threshold`<br>`um-head room-threshold` |
| `ingress-port-service-pool` | *interface* (such as `Ethernet1/7`) | `service-pool` | `um-share-threshold` |
| `ingress-service-pool` | `service-pool` | | `um-share-threshold` |
| `egress-port-service-pool` | *interface* (such as `Ethernet1/7`) | `service-pool` | `uc-share-threshold`, `um-share-threshold`, `mc-share-threshold`, |
| `egress-service-pool` | `service-pool` | | `um-share-threshold`, `mc-share-threshold` |
| `egress-rqe-queue` | `queue` | | `rqe-threshold` |
| `include-device` | | | `threshold` |
| `egress-uc-queue` | `unicast-queue` | | `uc-threshold` |
| | `interface` | `user-queue` | `uc-threshold` |
| `egress-mc-queue` | `multi-cast-queue` | | `mc-threshold` |
| | `interface` | `user-queue` | `mc-threshold` |
| `egress-cpu-queue` | `queue` | | `cpu-threshold` |

The following table contains the parameters that can be used as indexes for each realm:

| Element | Description |
|---|---|
| *realm* | Identifies a group of buffer utilization statistic counters (string). One of:<br>• `ingress-port-priority-group`<br>• `ingress-port-service-pool`<br>• `ingress-service-pool`<br>• `egress-port-service-pool`<br>• `egress-service-pool`<br>• `egress-rqe-queue`<br>• `egress-cpu-queue`<br>• `egress-mc-queue`<br>• `egress-uc-queue`<br>• `device` |
| *interface* | The interface name (string).<br>For example: *Ethernet1/X*. |
| *service-pool* | The service pool range, from 0-1. |
| *priority-group* | The priority group range, from 0-7. |
| *user-queue* | The user queue range, from 0-7. |
| *unicast-queue* | The absolute unicast queue range, from 0-16383. |
| *multicast-queue* | The absolute multicast queue range, from 0-2600. |
| *cpu-queue* | The CPU queue range, from 0-47. |
| *rqe-queue* | The RQE queue range, from 0-10. |
| *queue-group* | The queue group range, from 0-127. |
| *threshold* | The threshold level, in percentage, from 1-100. |
| *um-share-threshold* | The threshold level for unicast and multicast shared buffer utilization, in percentage, from 1-100. |
| *uc-share-threshold* | The threshold level for unicast shared buffer utilization, in percentage, from 1-100. |
| *mc-share-threshold* | The threshold level for multicast shared buffer utilization expressed in percentage, from 1-100. |

## class TelemetryDevice_Feature

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

### *set_sys_feature()*

Configures the system features.

#### Syntax

`set_sys_feature(<dict_sys_feature>)`

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

| 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. Default value: `1`. |
| *msg-interval* | Determines the interval with which the registration and heartbeat messages are sent to the collector; units of seconds, range 1-600. Default value: `5`. |

#### Returns

A dictionary containing the following elements::

| Element | Description |
|---|---|
| *return status* | Return status of the API call:<br>• `TRUE`: Successfully set the Buffer statistics.<br>• `FALSE`: Setting Buffer statistics failed |
| *error message* | String value containing the reason for the API failure:<br>• none: When Return Status is `TRUE`.<br>• error string: When Return Status is `FALSE`. |

### *get_sys_feature()*

Gets the system features parameters set on the switch.

#### Syntax

`get_sys_feature()`

### Returns

A dictionary containing the following elements:

| 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. Default value: `1`. |
| *msg-interval* | Determines the interval with which the registration and heartbeat messages are sent to the collector; units of seconds, range 1-600. Default value: `5`. |

## class Telemetry_DeviceProp()

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

## *get_swprop()*

Retrieves the device properties.

### Syntax

```
get_swprop()
```

### Returns

A dictionary containing the following elements:

| Element | Description |
|---|---|
| *time-stamp* | Time of the report generation. |
| *number-of asics* | Number of ASICS in the switch (integer). |
| *asic-info* | A dictionary consisting of the following values:<br>• `asic-id`: ASIC identifier on the switch (string)<br>• `chip-id`: part number of the silicon (string)<br>• `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). |

| Element | Description |
| --- | --- |
| *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()*

Gets properties of a VLAN if the VLAN identifier (*vid*) is a valid VLAN ID or of all VLANs if *vid* is `None`.

#### Syntax

`python_get_vlan(<vid>)`

where:

| Variable | Description |
|---|---|
| *vid* | The VLAN ID (integer). |

#### 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:<br>• `pvid`<br>• `bridgeport_mode`<br>• `if_name` |
| *pvid* | Native VLAN ID for a trunk port;<br>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()*

Creates a VLAN.

### Syntax

`python_create_vlan(`*<vlan_info>*`)`

where:

| Variable | Description |
|---|---|
| *vlan_info* | A dictionary with the following VLAN information:<br>• `vlan_name`<br>• `vlan_id`<br>• `admin_state`<br>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()*

Deletes 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()*

Updates VLAN name.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *vid* | VLAN ID (integer). |
| *vlan_name* | The VLAN name (string). |

### Returns

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

## *python_update_vlan_admin_state()*

Updates VLAN admin state.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *vid* | VLAN ID (integer). |
| *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()*

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

#### Syntax

`python_get_vlan_properties(`<*if_name*>`)`

where:

| 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;<br>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.<br>String with possible values:<br>• `tagged`<br>• `untagged` |
| *egress_type_vlans* | A list of VLANs on which the switch tags egress traffic;<br>an integer from 1-3999. |

## *python_update_vlan_properties()*

Updates 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:<br>• `if_name`<br>• `bridgeport_mode`<br>• `pvid`<br>• `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;<br>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.<br>String with possible values:<br>• `tagged`<br>• `untagged` |
| *egress_type_vlans* | A list of VLANs on which the switch tags egress traffic; an integer from 1-3999. |

### Returns

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

# VRF Module

The class in this module manages Virtual Routing and Forwarding (VRF). To use this module, in the Python file or in the Python interpreter, enter:

```
import vrfApi
```

## class VRF

This class provides a function for managing VRF.

### *get_vrf_entry()*

Gets all VRF details.

#### Syntax

`get_vrf_entry(<vrf_name>)`

where:

| Variable | Description |
|---|---|
| *vrf_name* | (Optional) The name of the virtual router (string).<br>Default value: `none`.<br>**Note:** The VR can 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()*

Gets 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: `None`. |
| *if_name* | (Optional) Interface name used for communication (string). Default value: `None`.<br>**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: `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`, `no`. Default value: `yes`. |
| *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);<br>one of `up`, `down`. Default value: `up`. |
| *oper_state* | The operation state of the VR (string);<br>one of `master`, `backup`, `init`. |
| *track_if* | The interface to track by this VR (string).<br>Default value: `none`.<br>**Note:** If an interface is specified, it must exist. |
| *accept_mode* | Enables or disables the accept mode for this session (string);<br>one of `yes`, `no`. Default value: `yes`. |
| *switch_back_delay* | The switch back delay interval;<br>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 value: `no`. |

## *get_vrrp_intf()*

Gets properties of all VRRP VRs of specified interfaces.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *if_name* | The Ethernet interface name (string).<br>**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;<br>an integer from 1-255. Default value: `0`. |
| *ip_addr* | The IP address of the VR; a valid IPv4 address. |
| *ad_intvl* | Advertisement interval (The number of centi-seconds between advertisements for VRRPv3); a multiple of 5 from 5-4095. Default value: 100 centi-seconds. |
| *preempt* | Enable the preemption of a lower priority master;<br>one of `yes`, `no`. Default value: `yes`. |

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

## *get_vrrp_accept_mode()*

Determines whether a virtual router in Master state will accept packets.

### Syntax

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

where:

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

### Returns

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

## *get_vrrp_advt_interval()*

Gets 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).<br>**Note:** The interface must exist. |

`where:`

| Variable | Description |
|---|---|
| *vid* | The VLAN ID (integer). |

### 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()*

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

### Syntax

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

where:

| Variable | Description |
|---|---|
| *vr_id* | The VRRP session Virtual Router (VR) ID; an integer from 1-255. |
| *af_type* | The Address Family type (integer); `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6). |
| *if_name* | Ethernet interface name used for communication (string)<br>**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;<br>one of `yes`, `no`. Default value: `yes`. |

## *get_vrrp_priority()*

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

### Syntax

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

where:

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

Sets the accept mode for a VRRP session when VRPP V3 is enabled.

### Syntax

`set_vrrp_accept_mode(`*<vr_id>*`,`*<af_type>*`,`*<if_name>*`,`*<accept_mode>*`)`

where:

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

### Returns

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

## *set_vrrp_advt_interval()*

Sets the advertisement interval of a virtual router.

### Syntax

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

where:

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

### Returns

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

## *set_vrrp_preempt_mode()*

Enables or disables the preempt mode for a session.

### Syntax

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

where:

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

### Returns

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

## *set_vrrp_priority()*

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

### Syntax

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

where:

| Variable | Description |
|---|---|
| *vr_id* | The VRRP session Virtual Router (VR) ID; an integer from 1-255. |
| *af_type* | The Address Family type (integer); `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6). |
| *if_name* | The Ethernet interface name (string).<br>**Note:** The interface must exist. |
| *prio* | The priority of the VR on the switch;<br>an integer from 1-254. Default value: 100. |

### Returns

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

## *set_vrrp_switch_back_delay()*

Gets the IGMP snooping status for a VLAN.

### Syntax

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

where:

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

### Returns

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

## *set_vrrp_oper_primary_ipaddr()*

Sets the primary IP address of the VRRP virtual router.

### Syntax

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

where:

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

### Returns

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

## *set_vrrp_monitored_circuit()*

Gets the IGMP snooping status for a VLAN.

### Syntax

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

where:

| Variable | Description |
|---|---|
| *vr_id* | The VRRP session Virtual Router (VR) ID; an integer from 1-255. |
| *af_type* | The Address Family type (integer); `2` for AF_INET (IPv4) and `10` for AF_INET6 (IPv6). |
| *if_name* | The Ethernet interface name (string).<br>**Note:** The interface must exist. |
| *track_if* | The interface to track by this VR.<br>**Note:** If an interface is specified, it must exist. |

### Returns

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

## *delete_vrrp_vr()*

Deletes a VRRP VR.

### Syntax

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

where:

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

### Returns

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

## *set_vrrp_vr()*

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

### Syntax

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

where:

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

### Returns

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

# VXLAN Module

The following class and functions configure Virtual Extensible LAN (VXLAN) properties. To use this module, in the Python file or in the Python interpreter, enter:

```
import vxlanApi
```

## *python_vxlan_interface_ena()*

Enables VXLAN mode on a specific interface.

### Syntax

`python_vxlan_interface_ena(<interface_name>)`

where:

| Variable | Description |
|---|---|
| *interface_name* | (Mandatory) Whether the VXLAN is enabled on a specific interface (string) For example: *Ethernet1/1*. |

### Returns

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

## *python_vxlan_interface_dis()*

Disables VXLAN mode on a specific interface.

### Syntax

`python_vxlan_interface_dis(<interface_name>)`

where:

| Variable | Description |
|---|---|
| *interface_name* | (Mandatory) Whether the VXLAN is disabled on a specific interface (string) For example: *Ethernet1/1*. |

### Returns

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

## *python_vxlan_interface_get()*

Gets the VXLAN status for a specific interface.

### Syntax

`python_vxlan_interface_get(<interface_name>)`

where:

| Variable | Description |
|---|---|
| *interface_name* | (Mandatory) Whether the VXLAN is disabled on a specific interface (string); For example: *Ethernet1/1*. |

### Returns

A dictionary containing the following:

| Element | Description |
|---|---|
| *vxlan* | The VXLAN interface status (string);<br>one of `Enabled`, `Disabled`. |

## *python_vxlan_get()*

Gets the VXLAN configuration.

### Syntax

```
python_vxlan_get()
```

### Returns

A dictionary containing the following:

| Element | Description |
|---|---|
| *tunnel_ip_addr* | The local Virtual Tunnel End Point (VTEP), the IP address (string), in the following format: `"10.1.2.1"` |
| *vlan_bindings* | A list of Virtual Network Instance (VNI) VLAN maps. |
| *remote_vtep* | A list of VTEP with VNI. |

## *python_vxlan_vni_cnt_get()*

Gets all of the VXLAN VNI counters.

### Syntax

```
python_vxlan_vni_cnt_get()
```

### Returns

A dictionary containing the following:

| Element | Description |
|---|---|
| *bin* | Bytes received; a positive integer. |
| *pktin* | The number of packets received; a positive integer. |

| Element | Description |
|---|---|
| *bout* | Bytes sent; a positive integer. |
| *pkout* | The number of packets sent; a positive integer. |
| *vni* | The VNI identifier; an integer from 1-16000000. |

## *python_vxlan_vp_get()*

Gets all of the VXLAN Virtual Ports (VP).

### Syntax

```
python_vxlan_vp_get()
```

### Returns

A dictionary containing the following:

| Element | Description |
|---|---|
| *count* | The number of virtual ports (integer). |
| *Virtual port* | A list of virtual ports. |
| *remoteTEP* | The type of VTEP (string); one of `LOCAL`, `REMOTE`. |
| *vlan* | The VLAN number (string); the valid VLAN values range from 1-4000. |
| *port* | The switch interface name (string). |

## *python_vxlan_vp_cnt_get()*

Gets all of the VXLAN VP counters.

### Syntax

```
python_vxlan_vp_cnt_get()
```

### Returns

A dictionary containing the following:

| Element | Description |
|---|---|
| *bin* | Bytes received; a positive integer. |
| *pktin* | The number of packets received; a positive integer. |
| *bout* | Bytes sent; a positive integer. |
| *pkout* | The number of packets sent; a positive integer. |
| *vni* | The VNI identifier; an integer from 1-16000000. |

## *python_vxlan_vp_cnt_del()*

Clears all the VXLAN VP counters.

### Syntax

```
python_vxlan_vp_cnt_del()
```

### Returns

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

## *python_vxlan_vn_cnt_del()*

Clears all the VXLAN VNI counters.

### Syntax

```
python_vxlan_vn_cnt_del()
```

### Returns

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

## *python_vxlan_tunnel_get()*

Gets all of the VXLAN VTEP tunnels.

### Syntax

```
python_vxlan_tunnel_get()
```

### Returns

A dictionary containing the following:

| Element | Description |
|---|---|
| *count* | The number of elements in the list (integer). |
| *tunnel* | The VTEP description; a list of dictionaries. |
| *status* | The BFD status of VTEP (string); one of `Up`, `Down`. |
| *remote* | The remote IP address (string), in the following format: `"1.1.2.3"`. |
| *local* | The local IP address (string), in the following format: `"1.2.3.4"`. |
| *type* | The VTEP type (string); one of `VTEP`, `local`. |

## *python_vxlan_mac_address_get()*

Gets all of the VXLAN MAC addresses.

### Syntax

```
python_vxlan_mac_address_get()
```

### Returns

A dictionary containing the following:

| Element | Description |
|---|---|
| *local-count* | The number of local MAC addresses (integer). |
| *remote-count* | The number of remote MAC addresses (integer). |
| *local-umac*<br>*remote-umac* | The local/remote MAC; a list of dictionaries. |
| *tunnel* | The VTEP IP address (string), in the following format: `"1.1.2.3"`. |
| *mac* | The MAC address (string), in the following format: `"00:50:56:84:E3:2E"`. |
| *local* | The local IP address (string), in the following format: `"1.2.3.4"`. |
| *type* | The VTEP type (string); one of `SN(Active)`, `SN(Backup)`, `VTEP`, `local`. |

## *python_vxlan_vni_get()*

Gets all of the VXLAN VNI.

### Syntax

```
python_vxlan_vni_get()
```

### Returns

A dictionary containing the following:

| Element | Description |
|---|---|
| *count* | The number of VNI elements in the list (integer). |
| *vni* | The list of VNI (integer). |

## *python_vxlan_interface_vni_map_set()*

Gets all of the VXLAN VNI mappings.

### Syntax

`python_vxlan_interface_vni_map_set(`*<GLOBAL>*`,`*<action>*`,`*<VLAN>*`, `*<VNI>*`)`

where:

| Variable | Description |
|---|---|
| *global* | Wether the VNI-VLAN mappings are global (string). Default value: `GLOBAL`. |
| *action* | (Mandatory) Indicates if a VTEP is added or removed (integer); one of `0` to add a VTEP, or `2` to delete a VTEP. |
| *vlan* | The list of VLAN; a valid VLAN list. |
| *vni* | The list of VNI; a valid VNI list. |

### Returns

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

# class VxlanCfg()

This class has methods for setting VXLAN configurations.

## *set_tunnel_ip()*

Sets a Network Virtualization VXLAN tunnel IP.

### Syntax

`set_tunnel_ip(`*<tunnel_ip>*`)`

where:

| Variable | Description |
|---|---|
| *tunnel_ip* | (Mandatory) The new tunnel IP (string); a valid IP address in the following format: `“10.0.1.1”.` |

### Returns

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

## *del_tunnel_ip()*

Deletes the Network Virtualization VXLAN local tunnel IP.

### Syntax

```
del_tunnel_ip ()
```

### Returns

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

## *set_remote_vtep()*

Deletes the Network Virtualization VXLAN tunnel IP.

### Syntax

`set_remote_vtep(`*<action>*`,`*<vni>*`,`*<ip_list>*`)`

where:

| Variable | Description |
|---|---|
| *action* | (Mandatory) Indicates if a VTEP is added or removed (integer); one of `0` to add a VTEP, or `2` to delete a VTEP. |
| *vni* | (Mandatory) Indicates the VNI;<br>a positive integer from 1-16000000. |
| *ip_list* | (Mandatory) The list of IP addresses (string), in the following format: `["10.0.1.1", "10.2.2.1"]`. |

### 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 Lenovo ServerProven website to make sure that the hardware and software is supported by your product.
- Go to the Lenovo Support portal to check for information to help you solve the problem.
- Gather the following information to provide to the service technician. This data will help the service technician quickly provide a solution to your problem and ensure that you receive the level of service for which you might have contracted.
  - Hardware and Software Maintenance agreement contract numbers, if applicable
  - Machine type number (if applicable–Lenovo 4-digit machine identifier)
  - Model number
  - Serial number
  - Current system UEFI and firmware levels
  - Other pertinent information such as error messages and logs
- Start the process of determining a solution to your problem by making the pertinent information available to the service technicians. The IBM service technicians can start working on your solution as soon as you have completed and submitted an Electronic Service Request.

You can solve many problems without outside assistance by following the troubleshooting procedures that Lenovo provides in the online help or in the Lenovo product documentation. The Lenovo product documentation also describes the diagnostic tests that you can perform. The documentation for most systems, operating systems, and programs contains troubleshooting procedures and explanations of error messages and error codes. If you suspect a software problem, see the documentation for the operating system or program.

# Appendix C. Notices

Lenovo may not offer the products, services, or features discussed in this document in all countries. Consult your local Lenovo representative for information on the products and services currently available in your area.

Any reference to a Lenovo product, program, or service is not intended to state or imply that only that Lenovo product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Lenovo intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any other product, program, or service.

Lenovo may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Lenovo (United States), Inc.
1009 Think Place - Building One
Morrisville, NC 27560
U.S.A.

Attention: Lenovo Director of Licensing

LENOVO PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. Lenovo may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

The products described in this document are not intended for use in implantation or other life support applications where malfunction may result in injury or death to persons. The information contained in this document does not affect or change Lenovo product specifications or warranties.

Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of Lenovo or third parties. All information contained in this document was obtained in specific environments and is presented as an illustration. The result obtained in other operating environments may vary.

Lenovo may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any references in this publication to non-Lenovo Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this Lenovo product, and use of those Web sites is at your own risk.

Any performance data contained herein was determined in a controlled environment. Therefore, the result obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

# Trademarks

Lenovo, the Lenovo logo, Flex System, System x, NeXtScale System, and X-Architecture are trademarks of Lenovo in the United States, other countries, or both.

Intel and Intel Xeon are trademarks of Intel Corporation in the United States, other countries, or both.

Internet Explorer, Microsoft, and Windows are trademarks of the Microsoft group of companies.

Linux is a registered trademark of Linus Torvalds.

Other company, product, or service names may be trademarks or service marks of others.

# Important Notes

Processor speed indicates the internal clock speed of the microprocessor; other factors also affect application performance.

CD or DVD drive speed is the variable read rate. Actual speeds vary and are often less than the possible maximum.

When referring to processor storage, real and virtual storage, or channel volume, KB stands for 1 024 bytes, MB stands for 1 048 576 bytes, and GB stands for 1 073 741 824 bytes.

When referring to hard disk drive capacity or communications volume, MB stands for 1 000 000 bytes, and GB stands for 1 000 000 000 bytes. Total user-accessible capacity can vary depending on operating environments.

Maximum internal hard disk drive capacities assume the replacement of any standard hard disk drives and population of all hard-disk-drive bays with the largest currently supported drives that are available from Lenovo.

Maximum memory might require replacement of the standard memory with an optional memory module.

Each solid-state memory cell has an intrinsic, finite number of write cycles that the cell can incur. Therefore, a solid-state device has a maximum number of write cycles that it can be subjected to, expressed as total bytes written (TBW). A device that has exceeded this limit might fail to respond to system-generated commands or might be incapable of being written to. Lenovo is not responsible for replacement of a device that has exceeded its maximum guaranteed number of program/erase cycles, as documented in the Official Published Specifications for the device.

Lenovo makes no representations or warranties with respect to non-Lenovo products. Support (if any) for the non-Lenovo products is provided by the third party, not Lenovo.

Some software might differ from its retail version (if available) and might not include user manuals or all program functionality.

# 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].<br>• 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.<br>• The deliquescent relative humidity of the particulate contamination must be more than 60%[2].<br>• The room must be free of conductive contamination such as zinc whiskers. |
| Gaseous | • Copper: Class G1 as per ANSI/ISA 71.04-1985[3]<br>• 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.

CE 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 Betriebsmittein**

Dieses Produkt entspricht dem „Gesetz über die elektromagnetische Verträglichkeit von Betriebsmitteln“ EMVG (früher „Gesetz über die elektromagnetische Verträglichkeit von Geräten“). Dies ist die Umsetzung der EU-Richtlinie 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。