Lenovo RackSwitch G8264

Python Programming Guide

For Networking OS 8.3

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 2015)

Preface

The *Lenovo Networking OS™ 8.3 Python Programming Guide for the RackSwitch G8264* describes how to configure and use the Networking OS 8.3 software on the RackSwitch G8264 (referred to as G8264 throughout this document). For documentation on installing the switch physically, see the *Installation Guide* for your G8264.

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 G8264 is available in the following guides:

- *Lenovo RackSwitch G8264 Installation Guide*
- *Lenovo RackSwitch G8264 Application Guide for Networking OS 8.3*
- *Lenovo RackSwitch G8264 ISCLI Reference Guide for Networking OS 8.3*

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
Boot Script	A python script file used in netboot module
Function	Lists an action and associated arguments, for example: cfg_enableNTP(*args, **kwds)
Function Arguments	Objects passed to a function when it is called inside a script
N/OS Python API	Extensions to the Python library provided by Lenovo
N/OS Python ISCLI Wrapper Method	The N/OS Python ISCLI method performs the following actions: • Takes a CLI command string as the input argument • Sends the CLI command to N/OS • Creates an `analyzer` object based on the output of the CLI command from N/OS and returns it to the calling script
Python scheduler	An engine to run scripts when specified events occurs.
Script Arguments	Strings passed to a script at run time

Typographic Conventions

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

Table 2. *Typographic Conventions*

Typeface or Symbol	Meaning	Example
`ABC123`	This type is used for names of commands, files, and directories used within the text. It also depicts on-screen computer output and prompts.	View the `readme.txt` file. `Main#`
`ABC123`	This bold type appears in command examples. It shows text that must be typed in exactly as shown.	`Main#` **`sys`**
<*ABC123*>	This italicized type appears in command examples as a parameter placeholder. Replace the indicated text with the appropriate real name or value when using the command. Do not type the brackets. This also shows book titles, special terms, or words to be emphasized.	To establish a Telnet session, enter: `host#` **`telnet`** <*IP address*> Read your *User's Guide* thoroughly.
[ ]	Command items shown inside brackets are optional and can be used or excluded as the situation demands. Do not type the brackets.	`host#` **`ls`** [**`-a`**]
|	The vertical bar (|) is used in command examples to separate choices where multiple options exist. Select only one of the listed options. Do not type the vertical bar.	`host#` **`set left|right`**
AaBbCc123	This block type depicts menus, buttons, and other controls that appear in Web browsers and other graphical interfaces.	Click the **Save** button.

How to Get Help

If you need help, service, or technical assistance, visit our web site at the following address:

http://www.lenovo.com/support

The warranty card received with your product provides details for contacting a customer support representative. If you are unable to locate this information, please contact your reseller. Before you call, prepare the following information:

- Serial number of the switch unit
- Software release version number
- Brief description of the problem and the steps you have already taken
- Technical support dump information (# **`show tech-support`**)

Contents

Chapter 1. Introduction to Python Scripting

The Lenovo Networking OS (N/OS) version 8.3 Python function API is a set of libraries of API functions that are embedded into the Lenovo RackSwitch G8264 Industry-Standard Command-Line Interface (ISCLI) to support script execution.

N/OS and Python scripting extend the standard Python library, which allows you to run and manage Python scripts on the G8264. This feature provides the following benefits:

- Automated switch provision and management
- Performing switch monitoring tasks
- Using `netboot` to download and execute the Python scripts at RackSwitch G8264 bootup
- Automatic G8264 firmware update
- Automatic configuration file generation
- Send notifications to users via email or `syslog` messages

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 2.6.3 is installed on the G8264 version 8.3.

Ways to Run N/OS Python Scripts

Python scripts can be scheduled to run either at startup or when a system or time event occurs. These scripts can send configuration and show commands to the G8264, save variables, and send syslog messages. A script can be loaded via netboot instead of a configuration file. "Introduction to Python Scripting" on page 5 explains ways to run scripts.

About the Python API Functions

Python libraries allow you to perform the following tasks:

- Import and call Lenovo Python APIs, for example, `mgmt.get_swMac()`
- Send CLI commands to the G8264 using the CLI method, for example, `Pylib.cmd ('show version')` and capture CLI output

See "Writing Python Scripts" on page 21 for details about all of N/OS Python modules.

Chapter 2. Running Python Scripts via the ISCLI

The most straightforward method to run a script on the RackSwitch G8264 is to execute it directly:

```
RS G8264# python <script filename> [arguments list]
```

The script will be run in the foreground. You can use "Ctrl +C" to stop the script' execution. An execution log will be recorded into a file.

For information about transferring scripts to the G8264, see Chapter 3, "Managing Scripts.

Running a Basic Script

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

```
RS G8264#
RS G8264# show script helloWorld.py

print "Hello world!"

RS G8264# python helloWorld.py
Hello world!
Python shell or script completed successfully
RS G8264#
```

Running a Basic Script with Arguments

Here is an example of a basic script with arguments:

```
RS G8264# show script scriptWithArgs.py
import sys

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

RS G8264# python scriptWithArgs.py "1 secondArgument  3rdArgument"
Argument 1 is: 1
Argument 2 is: secondArgument
Argument 3 is: 3rdArgument
Python shell or script completed successfully
RS G8264#
```

Entering a Python Shell

You can enter the Python shell, the interactive mode of the Python interpreter, directly via ISCLI 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, type **`python`** at the switch command prompt.

```
RS G8264# python
>>>
```

To view online help, type `help` followed by the command on which you want information:

```
>>> from pylib import pylib
>>> help(pylib.set_var)
>>>
>>> from pylib import pylib
Help on function stover in module pylib.pylib:
set_var(varName, objValue)
    Description: Store the objValue to persistent variable name varName. This
variable resides in the Python scheduler memory until the switch reloads or power
cycles.
    Input: varName (Str) the variable name to store the objValue, the length must
not exceed 31 characters. The objValue (object) will be serialized into a string
format and
    this string must not be larger than 3039 characters. After the serialization
the value string is an empty string, the variable, if exists, will be deleted. Only
20 persistent
    variables are supported. This method will raise the following exception
ValueError (variable or value length exceed), MemoryError (Maximum persistent var.
exceed)
    Output: None

>>> help(pylib.get_var)
Help on function get_var in module pylib.pylib:

get_var(varName)
    Description: Get the value of persistent variable.
    Input: varName (Str) the variable name to store the objValue, the length must
not exceed 31 characters. This method will raise the following exceptions
ValueError
    (var. name too long), NameError (var. name not found).
    Output: The value object or None on errors


>>> pylib.set_var("testName","testvalue")
>>> var1=pylib.get_var("testName")
>>> print var1
testvalue
>>> exit()
Python shell or script completed successfully
```

Chapter 3. Managing Python Scripts

Script files are saved in persistent storage on the G8264, while the script log files are saved to volatile storage. The maximum storage for script files is 2 M bytes, and 80M bytes for script log.

Lenovo Networking OS (N/OS) for the G8264 provides the following managing actions on scripts:

- Download a script file from a TFTP server
- Backup a script file to a TFTP server
- Backup a script log file to a TFTP server
- Show available scripts
- Show the content of a specific script
- Create a new script or edit an existing script

For a detailed list of CLI commands, please refer to the *RackSwitch G8264 ISCLI Command Reference Guide*.

Downloading a Script from a TFTP Server

Use the following command to download a TFTP script:

```
RS G8264# copy tftp script address <IPv4 address> filename <script filename>
[data-port|mgt-port]
```

Uploading a Script to a TFTP Server

Use the following command to upload a script to a TFTP server:

```
RS G8264# copy script tftp address <IPv4 address> filename <script filename>
[data-port|mgt-port]
```

Editing a Script Directly on the G8264

Use the following command to edit a script directly on the G8264:

```
RS G8264# edit script <script filename>
```

Deleting a Script

Use the following command to delete a script:

```
RS G8264# no script <script filename>
```

Showing A List of Script Files

Use the following command to view a list of script files:

```
RS G8264# show script
```

Showing Script File Content

Use the following command to view script file contents:

```
RS G8264# show script <script filename>
```

Viewing Configured Scheduler Jobs

Use the following command to view the scheduler jobs configured on the G8264:

```
RS G8264# show scheduler job
```

Running jobs will be displayed in the running configuration display.

For more information about ISCLI scripts, see the *Lenovo RackSwitch G8264 Industry-Standard CLI Reference for N/OS 8.3.*

Chapter 4. The Python Scheduler

The scheduler's main responsibility is to provide a programmatic mechanism to run Python scripts when specified events occur. These events are defined by RackSwitch G8264 administrators and can be triggered by a timer, the `syslog`, or counter events.

You can schedule scripts to run as a response to an event using scheduler jobs and monitor the script execution. The maximum number of scheduler jobs configured on switch is 20; an error message will be displayed if this number is exceeded.

Using the Python Scheduler

The Python scheduler is an engine that runs outside of N/OS to execute Python scripts. The administrator defines scheduler job maps an event to an action. When the specified event occurs, the corresponding script will be executed, as shown in Figure 1.

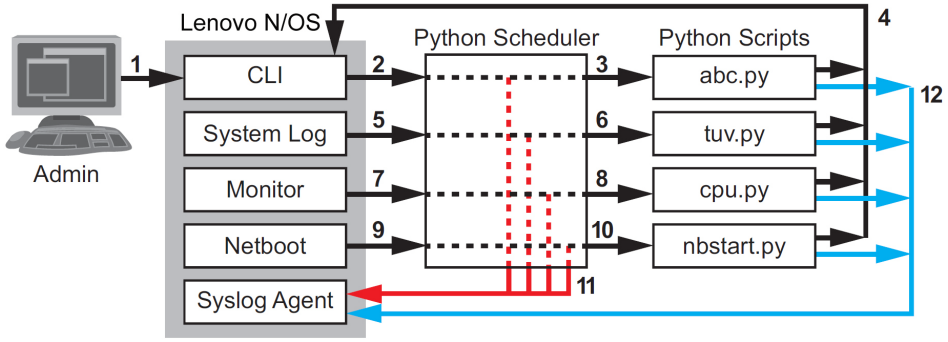
Note: You cannot modify an active scheduler job. The workaround is to delete the scheduler job and create a new one.

This section describes the CLI commands available to:

- configure a job
- remove a job
- show information about a job
- show currently running jobs
- kill a running job

There is also a configuration example.

Figure 1. A Python Script in Action



The Python Scheduler Sequence

The following events occur when a Python scheduler sequence is executed on a switch:

1. An administrator enters CLI commands to download a prepared script file to the G8264 and defines a `scheduler` job via CLI for the script execution. In this example:
   - The administrator schedules a job with a periodic `5s` timer event to execute script `abc.py`.
   - The administrator schedules a job to execute the script `tuv.py` when a `LINK_DOWN` event occurs.
   - The administrator schedules a job to execute the script `cpu.py` when CPU usage exceeds a specified threshold.
2. The CLI forwards job definitions to the Python scheduler..
3. The timer event `5s` fires, causing the Python scheduler to spawn a process to execute the `abc.py`script.

4. The Python script executes corresponding CLI commands issued by the activated script.
5. The `syslog` detects a `LINK_DOWN` event and forwards event data to Python scheduler.
6. The scheduler receives `LINK_DOWN` event from the `syslog` and spawns the `tuv.py` script.
7. A monitor thread periodically checks a counter for CPU usage or packet drops.
8. When the monitored counter reaches configured threshold, the Python scheduler will spawn an associated Python script.
9. When the G8264 boots up, `netboot` is sent to the scheduler.
10. The Python scheduler will execute the downloaded script when the booting process is complete.
11. The Python scheduler generates a syslog message if an error occurs, such as when script execution is aborted or unplanned.
12. The Python scripts can also generate syslog messages, These syslog messages can be displayed on the terminal or sent to a remote syslog server, like other N/OS syslog messages.

Example of a Python Scheduler Job

Following is an example of how to configure a scheduler job that is based on an event:

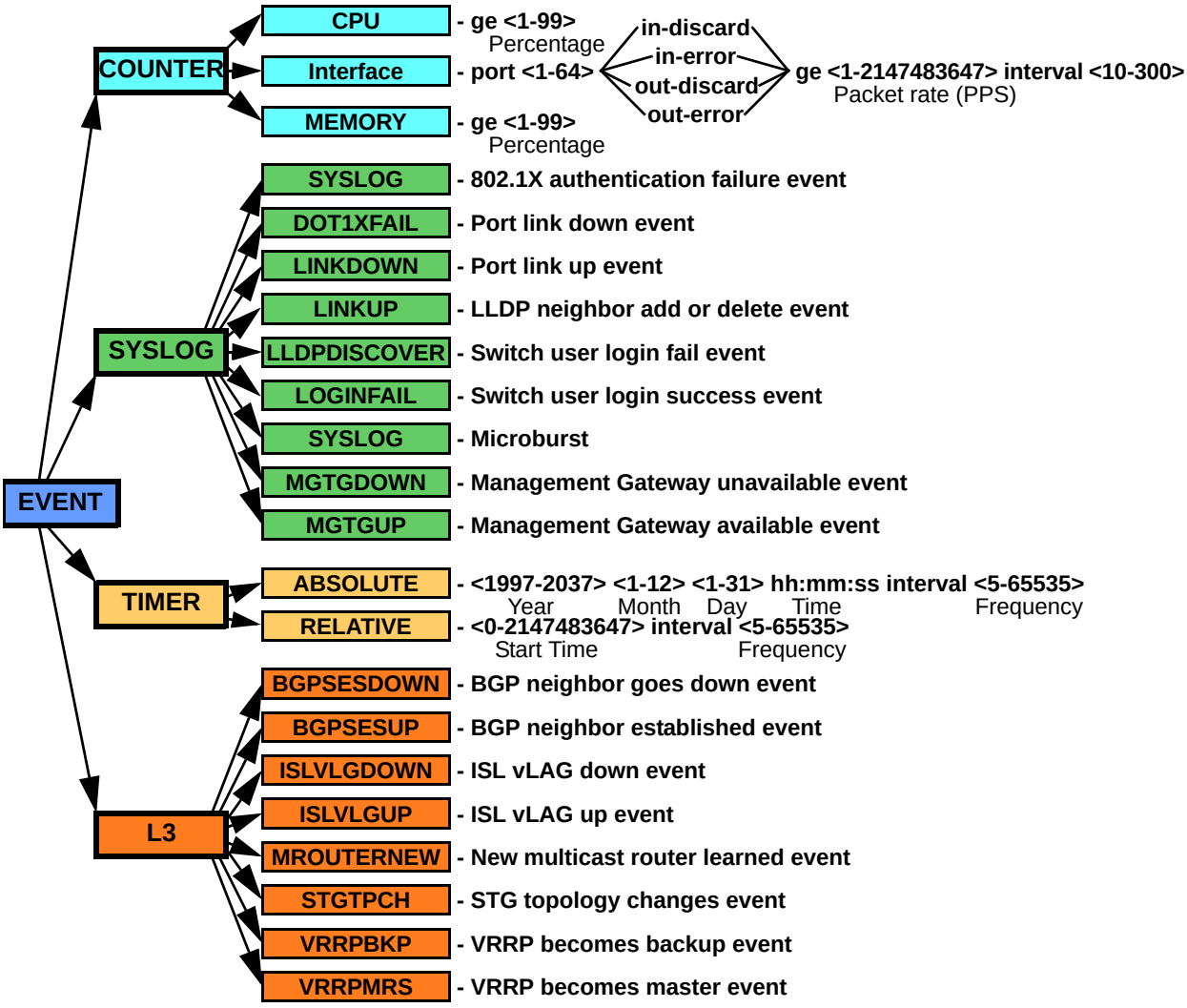
```
RS G8264(config)# scheduler job name JOB1
RS G8264(config-job)# event syslog LINKDOWN
RS G8264(config-job)# action helloWorld.py
RS G8264(config-job)# exit
RS G8264(config)# show scheduler job
------------------------------------------------------------------------------
Job name: "JOB1"
syslog event: LINKDOWN
action: "helloWorld.py"
------------------------------------------------------------------------------
------------------------------------------------------------------------------
Job name: "JOB1"
State: idle
Previous process id: 0
Next execution time: NA
Previous execution time: Yet to be executed or completed
Previous user msec: 0
Previous sys msec: 0
Previous cpu usage: 0.0 (%) Execution count:0
```

Events

Scheduler jobs are triggered by counter, timer, or `syslog` events. You can configure events to be triggered when a specified `syslog` or a pre-defined system event occurs.

Figure 2 illustrates the events types: counter, timer, and `syslog` messages.

Figure 2. Events



A scheduler job binds an event to an action. An action specifies the name of the script to be executed, along with the argument list.

```
RS G8264(config)# scheduler job name <job name>
RS G8264(config-job)# event <event type> <event description>
RS G8264(config-job)# action <script file name> [argument list]
RS G8264(config-job)# exit
```

Lenovo N/OS receives user-defined scheduler jobs lists. The running configuration is updated when a scheduled job is added or deleted. You can specify an arguments list to pass to the script when it is executed. You must provide these arguments when you define a scheduler job.

Counter Events

N/OS periodically checks counters for specified parameters. The counter is checked periodically by N/OS; the minimum interval is 10 seconds. N/OS will notify the scheduler if the counter exceeds the threshold in the specified interval.

Following is a list of functions to manage counter events.

To trigger a script execution when N/OS CPU usage exceeds a user-defined threshold:

```
RS G8264(config-job)# event counter cpu ge <1-99>
```

The fixed period interval is 60 seconds.

To trigger a script execution when N/OS memory usage percentage exceeds a user-defined threshold:

```
RS G8264(config-job)# event counter memory ge <1-50>
```

To trigger running a script when the rate ingress/egress discards frame rate on a port reaches a user-defined threshold:

```
RS G8264(config-job)# event counter interface port <x> in-discard ge
<1-2147483647>
[interval <time value >]
RS G8264(config-job)# event counter interface port <x> out-discard ge
<1-2147483647>
[interval <time value >]
```

Note: The minimum interval is 10s; the maximum is 300s, and default value is 60s. In the preceding commands, ge is equal to or greater than, and *x* is an index of a port. Counters for port channel and port range are not supported.

Following is an example of a scheduler counter event:

```
RS G8264(config)# scheduler job name <job name>
RS G8264(config-job)# event counter <counter name> ge <value> [interval <time value>]
RS G8264(config-job)# action <script file name> <argument list>
RS G8264(config-job)# exit
```

Table 3 lists values used in counter scheduler jobs.

Table 3. *Counter Values*

Counter Name	Comments
cpu	CPU usage in fixed interval (60s)
memory	memory usage in fixed interval (60s)

Table 3. *Counter Values (continued)*

Counter Name	Comments
`interface port` *x* `in-discard` `interface port` *x* `out-discard`	Specified port *x* discard-frames rate in a configurable interval
`interface port` *x* `in-error` `interface port` *x* `out-error`	Specified port *x* error frames rate in a configurable interval
`interface port` *x* `igmp-group ge` <*number of IGMP groups*>	Specified port *x* trigger a script execution when the number of IGMP groups on a port is greater than or equal to a configurable threshold. **Note:** Port channel and port range are not supported with this counter value.

Timer Events

You can schedule a script to be executed by a defined timer (relative or absolute), and it can be executed once or periodically, with the minimum interval of 5 seconds

N/OS converts the relative start time to an absolute time when the user specifies commands to show the running configuration or to save the start-up configurations. When the system time zone is changed, all absolute timer jobs are rescheduled, so their execution time in the new time zone still matches the time specified in the configuration.

Following are examples of time-related scripts:

```
RS G8264(config)# scheduler job name <job name>
RS G8264(config-job)# event time {relative <time value>|absolute year month day
HH:MM:SS}
[interval <time value>]
RS G8264(config-job)# action <script  file name> <argument list>
RS G8264(config-job)# exit
```

The following code is an example of running a scheduler based on a relative timer interval:

```
RS G8264(config)# scheduler job name testtime
RS G8264(config-job)# event time relative 10 interval 10
RS G8264(config-job)# action test.py
RS G8264(config-job)# exit
Nov 19 23:59:59 192.168.49.50 INFO
scheduler: Started job="testtime" script="test.py"
log_file="testtime_test.py_2013-10-19_235945_2033.log"
Nov 19 23:59:59 192.168.49.50 NOTICE scheduler: Job="testtime" script="test.py"
log_file="testtime_test.py_2013-10-19_235945_2033.log" time=0.056 sec completed
successfully
Nov 20  0:00:09 192.168.49.50 INFO
scheduler: Started job="testtime"
script="test.py" log_file="testtime_test.py_2013-10-19_235955_2033.log"
Nov 20  0:00:09 192.168.49.50 NOTICE scheduler: Job="testtime" script="test.py"
log_file="testtime_test.py_2013-10-19_235955_2033.log" time=0.134 sec completed
successfully
Nov 20  0:00:19 192.168.49.50 INFO
scheduler: Started job="testtime" script="test.py"
log_file="testtime_test.py_2013-10-20_000005_2033.log"
Nov 20  0:00:19 192.168.49.50 NOTICE scheduler: Job="testtime" script="test.py"
log_file="testtime_test.py_2013-10-20_000005_2033.log" time=0.095 sec completed
successfully
```

syslog Events

syslog events trigger the G8264 to execute scripts. Following is a list of some of the events that can be logged:

- A user succeeds or fails to log in to a switch
- A microburst occurs
- A port LLDP neighbor is discovered on a user-defined port
- The link status change on the a user defined port
- Management Gateway Connectivity State Change

You can use scripts to respond to events when certain syslog messages are generated. For example:

```
RS G8264(config)# scheduler job name <job name>
RS G8264(config-job)# event syslog < mnemonic >
RS G8264(config-job)# action <script file name> <argument list>
RS G8264(config-job)# exit
```

See Table 4 for a list of syslog mnemonics. Following is an example of how the syslog mnemonic messages are passed to the Python script.

```
RS G8264(config-job)# event syslog <mnemonic>
```

Table 4. *Mnemonics and syslog Response*

Mnemonics	syslog Description
LOGINSUCC	Successful login from the console, telnet, or SSH
LOGINFAIL	Unable to login from the console, telnet, or SSH
MBURST	Microburst
LLDPDISCOVER	A port LLDP neighbor is discovered on the switch ports. **Note:** To enable an LLDP trap, use the command: `RS G8264(config-if)# lldp trap-notification`
LINKUP | ? LINKDOWN	The link status change on the switch ports
MGTGWDOWN | MGTGWUP	Management gateway connectivity state change
DOT1XFAIL	802.1 X authentication failure
BGPSESUP	BGP neighbor session established
BGPSESDOWN	BGP neighbor session failed
MROUTERNEW	New Multicast router learned
STGTPCH	STG topology change
ISLVLGUP | SLVLGDWN	vLAG ISL status change
VRRPMRS	VRRP master status change
VRRPBKP	VRRP backup status change

Note: You must enable `files` with corresponding features to make those features work.

The RackSwitch G8264 checks all ports' microburst state every 10 seconds. When the G8264 finds a port in a bursting state or an LLDP trap, it will send an event to scheduler to trigger a job. The `syslog` file is not displayed.

CLI Commands to Manage Scheduler Jobs

Configuring a Job

You can schedule a script to run in response to event triggers. The script runs in the background. The G8264 can manage up to 20 scheduler jobs.

Use the following commands to configure a job:

```
RS G8264(config)# scheduler job name <job name>
RS G8264(config-job)# event <event type> <event description>
RS G8264(config-job)# action <script file name> <argument list>
RS G8264(config-job)# exit
```

Deleting a Job

Use the following command to delete a job:

```
RS G8264(config)# no scheduler job <jobname>
```

Monitoring a Running Job

To monitor a running job:

- Observe the `syslog` message displayed on the terminal.

  There are log entries when the job is started and completed. If the specified job is running, it will be deleted automatically after its completion.

- Check the job-run status using the ISCLI:

```
RS G8264# show scheduler job running
```

Stopping a Running Job

The following are ways to stop jobs:

- Automatic: You can configure limits for running scripts based on CPU usage and configurable elapsed time in seconds. If this threshold exceeds, the script will be gracefully terminated.

  Note: The following limits are disabled by default except when CPU usage exceeds 50%.

- The following commands stop jobs based on user-configured limits:

```
RS G8264(config)# scheduler job cpu-limit <cpu limit (5-50)>
RS G8264(config)# no scheduler job cpu-limit
RS G8264(config)# scheduler job time-limit <10 - 600>
RS G8264(config)# no scheduler job time-limit
```

- Manual:

```
RS G8264# kill scheduler job name <jobname>
```

  This command will forcibly terminate the specified job.

Chapter 5. Using netboot to Execute Scripts

`netboot` allows the G8264 to automatically download a configuration file over the network during reboot, and apply the new configuration. With Python scripting support, `netboot` can also download a Python script and execute it. The script is also called a boot script.

After reboot, the G8264 includes the following options in its DHCP requests:

- Option 66 (TFTP server address)
- Option 67 (file path)

If the DHCP server returns this information, the G8264 will initiate a TFTP file transfer and loads the configuration file into the active configuration block or execute the boot script file from the TFTP server. As the G8264 boots up, it applies the new configuration file. Note that the option 66 TFTP server address must be specified in IP-address format: the host name is not supported.

If DHCP is not enabled, or the DHCP server does not return the required information, the G8264 uses the manually-configured TFTP server address and file path. `netboot` is executed in the background. Regardless of whether `netboot` is finished, the G8264 can restart.

After `netboot` executes the boot script, it does not automatically reboot the G8264. If the script updates the image, it can reboot the G8264 by itself, or you can reboot the G8264 later using ISCLI commands. The downloaded boot script is saved into memory. After the G8264 reboots, the boot script is deleted.

Note: Using the same boot script file name twice does not cause conflicts, because the boot script is deleted after it is run. You do not need to manually delete the script file.

Important `netboot` updates are logged in the SYSLOG. You can view them from the serial console.

Using netboot with Python on the G8264

`netboot` combined with Python scripting enhances functionality on the G8264. During reboot, the G8264 can automatically download either a configuration file, or a script file from a TFTP server. If the downloaded file is a script, it will be executed; if the downloaded file is a configuration file, it will be applied on switch.

`netboot` supports downloading Python boot scripts and startup-configuration. `netboot` does not support direct download of new firmware. However, a Python boot script can contain a firmware-update operation. When the G8264 runs, the boot script, the firmware will be updated.

Netboot Commands

N/OS has not changed or added to `netboot` functions. For details of `netboot` commands, see the *Lenovo Networking OS RackSwitch G8264 ISCLI Guide.*

Sample netboot Configuration Script

```
RS G8264(config)# boot netboot enable
Netconf mode enabled
RS G8264(config)# boot netboot tftp 10.30.58.2
RS G8264(config)# boot netboot cfgfile helloWorld.py
RS G8264(config)# show boot
Currently set to boot software image1, active config block.
NetBoot: enabled, NetBoot tftp server: 10.30.58.2, NetBoot cfgfile: helloWorld.py
Current boot Openflow protocol version: 1.0
USB Boot: disabled
Currently profile is default, set to boot with default profile next time. Current CLI
mode set to ISCLI with selectable prompt enabled.
Current FLASH software:
  image1: version 7.8.1, downloaded 0:50:57 Tue Jan 29, 2013
          NormalPanel
  image2: version 7.7.6.17, downloaded 19:38:21 Mon Jan 14, 2013
          NormalPanel
boot kernel: version 7.8.1
Currently scheduled reboot time: none

RS G8264(config)#
```

Note: For more detailed descriptions of CLI commands used, please refer to the *ISCLI Command Reference Guide.*

Table 5. *netboot Functions*

Functions	Description
RS G8264(config)# **boot netboot enable**	Enables netboot. When enabled, the G8264 boots into factory-default configuration, and attempts to download a new configuration file
RS G8264(config)# **no boot netboot enable**	Disables netboot
RS G8264(config)# **[no] boot netboot tftp** *<IP address>*	Configures the IP address of the TFTP server used for manual configuration. This server is used if DHCP is not enabled, or if the DHCP server does not return the required information
RS G8264(config)# **show boot**	Displays the current netboot parameters
RS G8264(config)# **boot netboot cfgfile**	Configures the name of the configuration file or script file to be downloaded

Chapter 6. Writing Python Scripts

This chapter describes script components, modules, and the API functions and arguments that you can use to create Python scripts to run on the RackSwitch G8264. Python API functions extends the standard Python library to provide configuration, management and monitoring abilities, which are located in several Python modules.

Script Components

The N/OS API Python modules contain the following sub-parts:

- Initialization: Import `pylib` modules, and opens the connection to the local G8264, and get the CLI object.
- Provisioning: Add, delete, edit the G8264's configuration or query for information from the G8264 using CLI objects by calling the ISCLI `cmd()` function
- Notifications (Optional): Send to `syslog` or email messages.
- Close the script session (Optional): Close the CLI session to the local G8264 and clean up.

The following is an example script that shows the script components:.

Figure 3. Script Components

```
from ibmpylib import ibmpylib, mgmt# Import ibmpylib and API modules
import os # Import Python module
''' Create Pylib object '''
Pylib = ibmpylib.objPylib# Get CLI object

# Declare notification params.
smtpAddr = 'smtp.company.com'
admin = ['admin@company.com']

# Open CLI session to local switch
mgmt.exe_login()

Pylib.echo('======> Test large output command, check CPU utilization <======')
Pylib.cmd('conf terminal')# Go to global configuration mode

# Add vlans from 2 to 4
Pylib.cmd('vlan 2-4')
Pylib.cmd('enable')
Pylib.cmd('exit')

# Create Ip interface from 1 to 4, assign IP address, vlan then enable it
for inf in range(1, 5):
    Pylib.cmd('interface ip {ifNo}'.format(ifNo=inf))
    Pylib.cmd('ip address 172.16.{ifNo}.128 255.255.255.0 enable'.format(ifNo=inf))
    Pylib.cmd('vlan {vlanId}'.format(vlanId=inf))

Pylib.cmd('end')# Go back to privilege mode
# Call APIs to remove IP interfaces
layer3.cfg_remIpInf(ifNo =2)
layer3.cfg_remIpInf(ifNo =3)
layer3.cfg_remIpInf(ifNo =4)
# Call APIs to remove Vlans
layer2.cfg_remVlans(vlanId = 2)
layer2.cfg_remVlans(vlanId = 3)
layer2.cfg_remVlans(vlanId = 4)

# Send log file via email
mgmt.exe_sendMail(smtpSrv = smtpAddr, lstRcp = admin, subj ='Mail from Python script', body='Execution completes',
                  attFiles = [Pylib.logFile], sender = 'ibmpylib@company.com')

# Close CLI session
result = mgmt.exe_logout()
# Remove the log file
os.remove(Pylib.logFile)
```

Viewing Online Help

The following is a sample of online help from the Python shell:

```
# import the API modules
from pylib import pylib, mgmt, layer2, layer3, acl
from pprint import pprint
>>> pprint(dir(mgmt))
# <<< List all API functions in mgmt module
…
'cfg_banner',
'cfg_configBlock',
'cfg_enableNTP',
'cfg_enableRadius',
'cfg_enableTacacs',
'cfg_hostName',
'cfg_netBoot',
'cfg_ntp',
'cfg_radiusPrimary',
'cfg_radiusSecondary',
'cfg_radiusServerOptions',
'cfg_remHostname',…
>>> help(mgmt.exe_tftpGetImage)
Help on function exe_tftpGetImage in module pylib.mgmt:
exe_tftpGetImage()
  API's description: This API will download Images from tftp server to the switch
Mandatory Arguments
imgDest(Str) in ['image1', 'image2', 'boot'], serverIP(String), filename (String)
Optional Arguments
bootNewImage = True, port='mgt-port' port in ['mgt-port', 'data-port']
    OutputBoolean (True if Get successfully otherwise False)
>>> help(mgmt)
Help on module pylib.mgmt in pylib:
NAME
    pylib.mgmt
FILE
    /usr/lib/python2.6/site-packages/pylib/mgmt.py
FUNCTIONS
    cfg_banner()
        API's description: This API will set or clear banner message for CLI
sessions (Does not support multi-lines)
…
-- more --
```

Python API Function Arguments

Python API functions have mandatory and optional arguments. Mandatory arguments must be set with correct types. Non-defined 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 G8264.

All Python API functions use keyword arguments. For example:

```
mgmt.exe_tftpPutCfg ( srvIP='9.70.2.36', filename='saveconfig.txt')
```

Python API Modules

Following are the N/OS Python modules:

- The `pylib` module provides functions such as `set_var`, `get_var`, `get_event`, `send_Syslog`, and others. See "System-Level Modules" on page 104.
- The `pydgb` module provides functions such as `cmd`, `get_analyzer`, `echo`, and `sleep`. See "System-Level Modules" on page 104.
- `mgmt.py`. provides management and system functions, such as `configure hostname`, `NTP`, `get_serial()`, `get_swMAC`, `exe_saveCfg()`, and others. See "Management Functions" on page 41.
- `layer2.py` provides Layer 2-related functions, such as port configuration, VLANs, LLDP., BGP, VRRP, and IGMP. See "Layer 2 Module" on page 58.
- `layer3.py` provides Layer 2-related functions, such as `ip interface, static route`, and others. See "Layer 3 Module" on page 77.
- `acl` functions allow you to configure and delete access control lists. See "ACL Module" on page 102.

Configuring Alerts from Scripts

You can configure scripts to send alerts through `SYSLOG` messages using the existing Web operating system infrastructure and email messages.

Types of API Functions

All N/OS API functions all use keyword arguments. Below is a list of the supported commands with a description, arguments, and output. Some syntax examples are included.

Management Functions

cfg_banner()

Sets the banner message for CLI sessions. Calling this function with no argument clears the banner message.

Note: This command does not support multiple lines.

Syntax

```
cfg_banner(*arg, **kwds)
```

Mandatory Arguments

None

Optional Arguments

`banner=''` String maximum is 80 characters

`enable=True`

Returns

Boolean: (`True` if the command sends successfully; otherwise `False`)

cfg_configBlock()

Sets the configuration block to `active`, `backup`, or `factory`

Syntax

```
cfg_configBlock(*args, **kwds)
```

Mandatory Arguments

`cfgBlock(Str)` in `['active','backup','factory']`

Optional Arguments

None

Returns

Boolean (`True` if send command successfully; otherwise `False`)

cfg_enableNTP()

Enables or disables Network Time Protocol (NTP) on the G8264.

Syntax

```
cfg_enableNTP(*args, **kwds)
```

Mandatory Arguments

```
enable(Bool)
```

Optional Arguments

None

Returns

Boolean (`True` if send command successfully; otherwise `False`)

cfg_enableRadius()

Enables RADIUS server authentication with back-door options

Syntax

```
cfg_enableRadius(*args, **kwds)
```

Mandatory Arguments

`enable`(Bool)

Optional Arguments

```
backdoor=False secBackdoor=False
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_enableTacacs()

Enables TACACS authentication with backdoor options.

Syntax

```
cfg_enableTacacs(*args, **kwds)
```

Mandatory Arguments

```
enable(Bool)
```

Optional Arguments

```
backdoor=False secBackdoor=False
```

Returns

`Boolean(True` if the command is successful; otherwise `False)`

get_hostName

Returns the host name of the G8264. If host name is not configured, it returns the product name.

Syntax

```
get_hostName(*args, **kwds)
```

Mandatory Arguments

`hostname`

Optional Arguments

None

Output

Boolean (`True` if send command successfully otherwise False)

cfg_netBoot()

Description

Enables `netboot` parameters

Syntax

`cfg_netBoot(`*args, **kwds)

Mandatory Arguments

`enable (Bool), srvIP(Str), cfgFile(Str)`

Optional Arguments

None

Output

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

cfg_ntp()

Configures the NTP server's parameters.

Syntax

```
cfg_ntp(*arg, **kwds)
```

Mandatory Arguments

```
priSrv(Str)
```

Optional Arguments

```
priPort = 'mgt-port', secSrv = '0.0.0.0', secPort =
'mgt-port', interval = 1440,

priPort and secPort in ['mgt-port', 'data-port']
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_radiusPrimary()

Configures the RADIUS primary server

Syntax

```
cfg_radiusPrimary(*args, **kwds)
```

Mandatory Arguments

```
srvIP(str), key(str)
```

Optional Arguments

```
port='mgt-port'

port in ['mgt-port', 'data-port']
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

Examples

```
>>> mgmt.cfg_radiusPrimary(srvIP = '9.70.2.36', key='secretkey')
# CALL API WITHOUT OPTIONAL ARGS.

True
>>> mgmt.cfg_radiusPrimary(srvIP = '9.70.2.36', key='secretkey', port =
'mgt-port')
# CALL API WITH OPTIONAL ARGS. True
```

cfg_radiusSecondary()

Configures the RADIUS secondary server

Syntax

```
cfg_radiusSecondary(*args, **kwds)
```

Mandatory Arguments

```
srvIP(str), key(str)
```

Optional Arguments

```
port='mgt-port', port in ['mgt-port', 'data-port']
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_radiusServerOptions()

Configures Radius server parameters

Syntax

```
cfg_radiusServerOptions(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

```
tcpPort = 1645, timeout= 3, retry= 3
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_remHostname()

Removes the G8264's hostname

Syntax

```
cfg_remHostname(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_remNetboot()

Disables and removes the `netboot` configuration

Syntax

```
cfg_remNetboot(*args, **kwds)
```

Mandatory Arguments

```
srvIP(Str), cfgFile(Str)
```

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_remRadius()

Disables RADIUS authentication and removes its configuration.

Syntax

```
cfg_remRadius(*args, **kwds)
```

Mandatory Arguments

```
srvType = 'primary' (Str) in ['primary','secondary']
```

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_remTacacs()

Disables TACACS authentication and removes its configuration

Syntax

```
cfg_remTacacs(*args, **kwds)
```

Mandatory Arguments

```
srvType = 'primary' (Str) in ['primary','secondary']
```

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_ssh()

Enables SSH access on the G8264

Syntax

```
cfg_ssh(*args. **kwds)
```

Mandatory Arguments

```
enable(Boolean)
```

Optional Arguments

```
tcpPort=22
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_sysNotice()

Sets (appends new line) or, if no argument is given, clears the system notice message for CLI sessions.

Syntax

```
cfg_sysNotice(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

```
sysNotice ='', enable = True
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_tacacsPrimary()

Configures the primary TACACS server

Syntax

```
cfg_tacacsPrimary(*args, **kwds)
```

Mandatory Arguments

```
srvIP(str), key(Str)
```

Optional Arguments

```
port='mgt-port' port in ['mgt-port', 'data-port']
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_tacacsSecondary()

Configures the secondary TACACS server

Syntax

```
cfg_tacacsSecondary(*args, **kwds)
```

Mandatory Arguments

```
srvIP(str), key(Str)
```

Optional Arguments

```
port='mgt-port' port in ['mgt-port', 'data-port']
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_tacacsServerOptions()

Configure optional TACACS server parameters

Syntax

```
cfg_tacacsServerOptions(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

`tcpPort = 49`, `timeout= 5`, `retry= 3`, `cmdLog = False`, `accounting= False`

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_telnet()

Enable or disable `telnet` access on the G8264

Syntax

```
cfg_telnet(*args, **kwds)
```

Mandatory Arguments

```
enable(Bool)
```

Optional Arguments

`tcpPort=23`

Output

Boolean (`True` if send command successfully; otherwise `False`)

exe_login()

Opens scripts session to the local G8264 and enter PRIV mode

Syntax

```
exe_login(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

exe_logout()

Closes the current script session to the local G8264

Syntax

```
exe_logout(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

exe_ping()

Executes a `ping` command from the G8264.

Note: This command does not support `tos` and `sourceID` options.

Syntax

```
exe_ping(*args, **kwds)
```

Mandatory Arguments

```
host(Str)
```

Optional Arguments

```
count=5, delay = 1000 (msec), payload = 0, port='mgt-port' in
['mgt-port', 'data-port']
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

Example

```
>>> mgmt.exe_ping(host = '9.70.2.36')
True
```

exe_reload()

Reloads the local G8264.

Note: Use this command only at the end of a script because the connection to the G8264 will be lost when it is reloading.

Syntax

```
exe_reload(*args,**kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

exe_saveCfg()

Saves the running configuration to the startup configuration

Syntax

```
exe_save(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

exe_sendMail()

Sends an email message to a list of recipients with optional attachments

Syntax

```
exe_sendMail(*args, **kwds)
```

Mandatory Arguments

```
smtpSrv(Str), lstRcp(list), subj(Str), body(Str)
```

Optional Arguments

```
attFiles = [], sender='', attFiles=[], srvPort=25
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

exe_telnet()

Opens a `telnet` connection to an external host from the G8264 ISCLI

Syntax

```
exe_telnet(*args, **kwds)
```

Mandatory Arguments

```
serverIP(String), filename(String)
```

Optional Arguments

```
serverIP(String), filename(String)
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

exe_tftpGetCfg()

Downloads a configuration file from a TFTP server and applies it to the running configuration

Syntax

```
exe_telnet(*args, **kwds)
```

Mandatory Arguments

```
serverIP(String), filename(String)
```

Optional Arguments

```
port='mgt-port' port in ['mgt-port', 'data-port']
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

exe_tftpGetImage()

Downloads boot images from a TFTP server to the G8264

Syntax

```
exe_tftpGetImage(args, **kwds)
```

Mandatory Arguments

imgDest(Str) in ['image1', 'image2', 'boot'], serverIP(String), filename (String)

Optional Arguments

```
bootNewImage = True, port='mgt-port' port in ['mgt-port',
'data-port']
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

exe_tftpPutCfg()

Saves the running configuration to a TFTP server

Syntax

```
exe_tftpPutCfg(*args, **kwds)
```

Mandatory Arguments

```
serverIP(String), filename(String)
```

Optional Arguments

```
port='mgt-port’
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

get_cpuUtilization()

Returns CPU use as a dictionary

Syntax

```
get_cpuUtilization(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Dictionary on success, dictionary with no values for empty output, `False` on failure

Example

```
{'1mCpuUtil': '43.19%', '1sCpuUtil': '49.95%', '5mCpuUtil':
'42.45%', '5sCpuUtil':

'49.11%','highestUtil': {'status': 'resumable', '1sCpuUtil':
'28.54%', '5mCpuUtil':

'31.44%', 'ThreadId': 5, '5sCpuUtil': '31.37%', '1mCpuUtil':
'30.08%', 'Name': 'CONS'}}
```

get_curUsers()

Returns information about currently logged-in users

Syntax

```
get_curUsers(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Current users as a dictionary or `False` if fails to get users

Example

```
{User0:{'USER':'admin', 'SESSIONID':0, 'COS': 'Admin',
'TERM': 'Console', 'LOGINTIME':

'22:34:22', 'FROMIP':'0.0.0.0', 'LASTCMD': 'show who'},
User2:{'USER':'admin','SESSIONID':2, 'COS': 'Admin', 'TERM':
'Telnet', 'LOGINTIME':

'22:40:22', 'FROMIP':'9.6.70.10', 'LASTCMD': 'enable'},
User1:{'USER':'admin','SESSIONID':1, 'COS': 'Admin', 'TERM':
'SSHv2', 'LOGINTIME':

'22:38:22', 'FROMIP':'192.168.2.1', 'LASTCMD': 'show
version'}}
```

get_curVersion()

Returns the current image version

Syntax

```
get_curVersion(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Version number as string on success, `None` for empty value, `False` on failure

get_hostName()

Returns the hostname of the G8264. If the hostname is not configured, returns the product name

Syntax

```
get_hostName(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Hostname of the switch as a string. Return product name (like G8264) if the switch uses the default hostname

get_memInfo()

Returns memory information (unit: kB)

Syntax

```
get_memInfo(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Dictionary on success, dictionary with no values for empty output, `False` on failure

Example

```
{'MemTotal': Int,'MemFree' : Int,'Buffers' : Int,'Cached' :
Int,'HighTotal' : Int,

'HighFree' : Int,'LowTotal': Int,'LowFree' : Int,'MemUsed' :
Int,'MemHiWater' : Int}
```

get_nextBootImage()

Returns the next boot image (`Image1` or `Image2`)

Syntax

```
get_nextBootImage(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Next boot image as string on success, `None` for empty value, `False` on failure

get_radiusInfo()

Gets RADIUS server configuration information

Syntax

```
get_radiusInfo(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Dictionary on success, dictionary. with no values for empty output, `False` on failure

Example

```
{'Retries': 3, 'Secondary Secret': 'encrypted', 'Secure
backdoor': 'enabled',

'Secondary RADIUS': '5.5.5.5', 'Primary Secret': 'encrypted',
'Backdoor': 'disabled',

'current server': 'radius-auth.lenovo.com', 'Timeout': 3,
'Primary RADIUS':

'radius-auth.lenovo.com', 'RADIUS Server': 'OFF', 'Port':
1645}
```

get_serialNum()

Returns the serial number of the G8264

Syntax

```
get_serialNum(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Serial number as string on success; `None` for empty value, otherwise `False`)

get_swMac()

Returns the MAC address of the G8264 and sets the MAC delimiter

Syntax

```
get_swMAC(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

```
delimiter = ':'
```

Output

MAC address of the switch as a string, the octet delimiter will be replaced by delimiter argument

get_tacacsInfo()

Gets TACACS configuration information

Syntax

```
get_tacacsInfo(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Dictionary on success, dictionary with no values for empty output, `False` on failure

Example

```
{'Retries': 3, 'Secondary Secret': 'empty', 'Secure
backdoor': 'disabled', 'Primary

Secret': 'encrypted', 'Backdoor': disabled, 'current server':
'0.0.0.0', 'index 0':

{'Local mapping': None, 'Remote privilege': None}, 'Timeout':
5, 'Primary TACACS+':

'0.0.0.0', 'Password change': 'disabled', 'Directed request':
'disabled', 'Command logging': 'disabled', 'TACACS+ Server':
'OFF', 'Login attempts': 2, 'user privilege mappings':
'disabled', 'Secondary TACACS+': '0.0.0.0', 'Port': 49,
'Command authorization': 'disabled', 'Accounting': 'OFF',
'Enable bypass': 'enabled'}
```

Layer 2 Module

cfg_addLacpPortch()

Adds an LACP portchannel

Syntax

```
cfg_addLacpPortch(*args, **kwds)
```

Mandatory Arguments

```
ports (Int or Str), key (Int), mode in ['active', 'off',
'passive']
```

Optional Arguments

```
channelId ='' (Str), priority = 32768 (Int)
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_addMac()

Adds a MAC address entry to the MAC address table

Syntax

```
cfg_addMAC(*args, **kwds)
```

Mandatory Arguments

```
macAddr(Str), type(Str) in ['static', 'multicast'],
vlan(Int), port(Int)
```

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_addPrivateVlan()

Configures a private VLAN

Syntax

```
cfg_addPrivateVlan(*args, **kwds)
```

Mandatory Arguments

```
vlanId(Int), pVlanType(Str) in
['community','isolated','primary']
```

Optional Arguments

associatedvid='' (Str)

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_addPrivateVlanPorts()

Adds ports to a private VLAN

Syntax

```
cfg_addPrivateVlanPorts(*args, **kwds)
```

Mandatory Arguments

```
ports(Str),  primaryVlan(In)
```

Optional Arguments

```
secondaryVlan = ''
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_addStaticPortch()

Adds a static portchannel

Syntax

```
cfg_addStaticPortch(*args, **kwds)
```

Mandatory Arguments

```
channelId(Int), enable(Bool), ports (Int or Str)
```

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_addVlagHealthchk()

Configures a health check peer for a VLAG configuration

Syntax

```
cfg_addVlagHealthchk(*args, **kwds)
```

Mandatory Arguments

```
peerIP(Str)
```

Optional Arguments

```
retry = 30, interval = 5, attempts = 3s
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_addVlagIslTrunk()

Configures an ISL trunk for a VLAG

Syntax

```
cfg_addVlagIslTrunk(*args, **kwds)
```

Mandatory Arguments

`trunkType (Str)` in `['lacp', 'portchannel'], trunkId (Int)` is `lacp admin key` or `portchannel Identifier`

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_addVlagTrunk()

Adds a trunk (LACP or Portchannel) to a VLAG configuration

Syntax

```
cfg_addVlagTrunk((*args, **kwds))
```

Mandatory Arguments

`trunkType (Str)` in `['lacp', 'portchannel'], trunkId (Int)` is `lacp admin key` or `portchannel Identifier`

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_addVlans()

Adds a VLAN

Syntax

```
cfg_addVlans(*args, **kwds)
```

Mandatory Arguments

```
vlanId(Int)
```

Optional Arguments

```
name='', enable=True
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_enableLldp()

Enables or disables LLDP globally

Syntax

```
cfg_enableLldp((*args, **kwds)
```

Mandatory Arguments

```
enable(Bool)
```

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_enableMacLearning()

Enables or disables MAC learning on the specified G8264 ports

Syntax

```
cfg_enableMacLearning(*args, **kwds)
```

Mandatory Arguments

```
ports (Int or Str), enable(Bool)
```

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_enableSwitchPort()

Switches the port from a Layer 3 port to a Layer 2 port

Syntax

```
cfg_enableSwitchPort(*args, **kwds)
```

Mandatory Arguments

```
ports (Int or Str)
```

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_enableVlag()

Sets the tier-ID and enables VLAG

Syntax

```
cfg_enableVlag(*args, **kwds)
```

Mandatory Arguments

```
enable(Bool)
```

Optional Arguments

```
tierId = 1 (Int), priority = 0 (Int
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_fdbAging()

Configures the aging time of the MAC address table

Syntax

```
cfg_fdbAging(*args, **kwds)
```

Mandatory Arguments

`ageTime(Int)`

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_portChannelHash()

Configures trunk hash options

Syntax

```
cfg_portChannelHash(*args, **kwds)
```

Mandatory Arguments

`hashType(Str)` in `['l2hash', 'l3hash']`

Optional Arguments

`l2options='l2-source-destination-mac'` in `['l2-destination-mac-address','l2-source-destination-mac','l2-source-mac-ad dress'] l3options='l3-source-destination-ip'` in `['l3-destination-ip-address', 'l3-source-destination-ip','l3-source-ip-address','l3-use-l2-hash']`

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_portState()

Enables or disables ports

Syntax

```
cfg_portState(*args, **kwds)
```

Mandatory Arguments

`Int or Str), enable (Bool)`

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_portToDefaultVlan()

Removes ports from the VLAN

Syntax

```
cfg_portToDefaultVlan(*args, **kwds)
```

Mandatory Arguments

```
port(Int), mode(Str) in ['access', 'trunk']
```

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_portToVlanMapping()

Assigns ports to VLANS

Syntax

```
cfg_portToVlanMapping(*args, **kwds)
```

Mandatory Arguments

```
port(Int), mode(Str) in ['access', 'trunk'], vlanId(Int)
```

Optional Arguments

```
nativeVId = ''
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_remLacpPortch()

Removes an LACP portchannel

Syntax

```
cfg_remLacpPortch(*args, **kwds)
```

Mandatory Arguments

```
port(Int or Str)
```

Optional Arguments

```
channelId ='' (Str)
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_remMAC()

Removes a MAC address entry from the MAC address table

Syntax

```
cfg_remMAC(*args, **kwds)
```

Mandatory Arguments

`macAddr(Str), type(Str)` in `['static', 'multicast'], vlan(Int)`

Optional Arguments

port=''

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_remPrivateVlan()

Removes a private VLAN configuration

Syntax

```
cfg_remPrivateVlan(*args, **kwds)
```

Mandatory Arguments

`vlanId(Int), pVlanType(Str)` in
`['community','isolated','primary']`

Optional Arguments `associatedvid='' (Str)` Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_remPrivateVlanPorts()

Removes ports from a private VLAN

Syntax

```
cfg_remPrivateVlanPorts(*args, **kwds)
```

Mandatory Arguments

channelId (Int)

Optional Arguments

None

Output

`True` if the command is successful; otherwise `False`

cfg_remStaticPortch()

Removes a static portchannel

Syntax

```
cfg_remStaticPortch(*args, **kwds)
```

Mandatory Arguments

`channelId (Int)`

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_remVlagHealthchk()

Removes a health check peer of a VLAG configuration

Syntax

```
cfg_remVlagHealthchk(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_remVlagIslTrunk()

Removes an ISL trunk from a VLAG configuration

Syntax

```
cfg_remVlagIslTrunk(*args, **kwds)
```

Mandatory Arguments

`trunkType (Str)` in `['lacp', 'portchannel']`

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_remVlagTrunk()

Removes a trunk (LACP or `Portchannel`) from a VLAG configuration

Syntax

```
cfg_remVlagTrunk(*args, **kwds)
```

Mandatory Arguments

`trunkType (Str) in ['lacp', 'portchannel'], trunkId`

Optional Arguments

None

Output

`True` if the command is successful; otherwise `False`

cfg_remVlans()

Removes VLANs

Syntax

```
cfg_remVlans(*args, **kwds)
```

Mandatory Arguments

`vlanId(int)`

Optional Arguments

None

Output

`True` if the command is successful; otherwise `False`

cfg_vlanToStg()

Maps a VLAN to an STG

Syntax

```
cfg_vlanToStg(*args, **kwds)
```

Mandatory Arguments

```
vlanId(Int), stgId(Int)
```

Optional Arguments

None

Output

`True` if the command is successful; otherwise `False`

clr_fdbCounter()

Clears the counters of the MAC-address-table

Syntax

```
clr_fdbCounter(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

`True` if the command is successful; otherwise `False`

clr_fdbTable()

Clears all dynamic entries from the MAC-address-table

Syntax

```
clr_fdbTable(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

`True` if the command is successful; otherwise `False`

get_fdbInfo()

Gets all MAC address entries

Syntax

```
get_fdbInfo(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

`display = '' (Str)` in `['','all','static','multicast']`

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Example

```
{'Mac 4': {'State': 'FWD', 'Vlans': 1, 'Macs': '00:00:00:00:11:15', 'Ports': 30,
'Trunk': ' '}}
```

get_infCounters()

Gets interface port counters

Syntax

```
get_infCounters(*args, **kwds)
```

Mandatory Arguments

`port(Int)`

Optional Arguments

None

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Data Port Example

```
>>> layer2.get_infCounters(port = 48)
# GET_ API
{'Discards': {'MMUAgingDiscardsOut': 0, 'PolicyDiscardsIn': 0, 'NonFwdStateIn': 0,
'VlanDiscardsIn': 0, 'IBPCBPDiscardsIn': 0, 'CellErrorDiscardsOut': 0,
'FilterDiscardsIn': 0, 'MMUDiscardsOut': 0, 'HOLBlockingDiscardsOut': 0,
'OtherDiscardsOut': 0}, 'infCounters': {'DiscardsOut': 0, 'UcastPktsIn': 0,
'UcastPktsOut': 0, 'PriFlowCtrlPktsOut': 0, 'MulticastPktsOut': 295678, 'OctetsOut':
21098017, 'MulticastPktsIn': 0, 'ErrorsOut': 0, 'DiscardsIn': 0, 'BroadcastPktsOut': 0,
'ErrorsIn': 0, 'OctetsIn': 0, 'FlowCtrlPktsIn': 0, 'FlowCtrlPktsOut': 0,
'BroadcastPktsIn': 0, 'PriFlowCtrlPktsIn': 0}}
>>> layer2.get_infCounters() False
'rxFrameErrors': 0, 'txOverruns': 0}
```

get_infEtherCounters()

Gets the ethernet counters on a G8264 port

Syntax

```
get_infEtherCounters(*args, **kwds)
```

Mandatory Arguments

```
port(Int)
```

Optional Arguments

None

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Example

```
{'dot3StatsInternalMacTransmitErrors': 0, 'dot3StatsExcessiveCollisions': 0,
'dot3StatsInternalMacReceiveErrors': 0,
'dot3StatsMultipleCollisionFrames': 0, 'dot3StatsFrameTooLongs': 0,
'dot3StatsLateCollisions': 0, 'dot3StatsFCSErrors': 0,
'dot3StatsAlignmentErrors': 0, 'dot3StatsSingleCollisionFrames': 0}
```

get_infIpCounters()

Gets the IP counters on a G8264 port

Syntax

```
get_infIpCounters(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Example

```
{'ipInReceives': 0, 'ipInDiscards': 0, 'ipInHeaderError': 0}
```

get_lacpInfo()

Gets LACP information

Syntax

```
get_lacpInfo(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

`status=''` in `['down', 'up', 'off', '']`

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Example

```
{'port 62': {'status': '--', 'adminkey': 62, 'port': 62, 'operkey': 62, 'mode': 'off'},
'port 63': {'status': '--', 'adminkey': 63, 'port': 63, 'operkey': 63, 'mode': 'off'}}
```

get_lldpIfCounters()

Gets the LLDP port's counters

Syntax

```
get_lldpIfCounters(*args, **kwds)
```

Mandatory Arguments

```
port(Int)
```

Optional Arguments

None

Output

Dictionary on success, dictionary with no values for empty output, `False` on failure

Example:

```
{'Frames Received': 0, 'TLVs Unrecognized': 0, 'Frames Received in Errors': 0, 'Frames
Discarded': 0, 'Neighbors Aged Out': 0, 'Frames Transmitted': 0}
```

get_lldpReceive()

Gets LLDP receive states

Syntax

```
get_lldpReceive(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Example

```
{'index 17': {'SNMP Notify': 'false', 'Alias': 30, 'Receive State':
'RX_WAIT_FOR_FRAME', 'Too Many Neighbor': 'false',
'Bad Frame': 'false', 'Remote Changed': 'false', 'DMAC': 'NB',
'RXInfo Ageout': 'false', 'port': 30, 'RCV Frame': 'false'}}
```

get_lldpTransmit()

Gets LLDP transmit states

Syntax

```
get_lldpTransmit(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Example

```
{'index 8': {'Alias': 21, 'DMAC': 'NB', 'Delay While': 0, 'Hold multiplier': 4, 'Local
Changed': 'false',                         'Reinit Delay': 2, 'Shutdown while': 0,
'Transmit Interval':
30, 'Transmit State': 'TX_LLDP_INITIALIZE', 'Transmit delay': 2, 'TxTTL': 0, 'TxTTR':
0, 'port': 21}},
'index 16': {}}
```

get_macCounters()

Gets the MAC address table counters

Syntax

```
get_macCounters(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Example

```
{current:100, hiwat:500}
```

get_portChannelInfo()

Get `portChannel` information

Syntax

```
get_portChannelInfo(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Example:

```
{'PortChannel 0': {'Channelstate': 'Enabled', 'Protocol': 'Static', 'PortChannel': 63,
'PortState': ['34: STG  1 DOWN', '35: STG  1 DOWN']}}
```

get_portMicroburst()

Gets the microburst port log for a particular port or all ports

Syntax

```
get_portMicroburst(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

```
'all'(Str)
```

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Example

```
{30: {'Guaranteed_Threshold': 45, 'Total': 0, 'MaxUsed': 0, 'Shared_Threshold': 36403,
'Shared_InUse': 0, 'Guaranteed_InUse': 0}}
```

get_portMicroburstStatus()

Gets microburst status on a particular port or all ports

Syntax

```
get_portMicroburstStatus(*ards, **kwds)
```

Mandatory Arguments

None

Optional Arguments

`'all' (Str)`

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Example

```
{1: 'NORMAL', 5: 'NORMAL', 9: 'NORMAL', 13: 'NORMAL', 17: 'NORMAL', 18: 'NORMAL', 19:
'NORMAL', 20: 'NORMAL',
29: 'NORMAL', 47: 'NORMAL', 48: 'NORMAL', 49: 'NORMAL', 50: 'NORMAL', 63: 'NORMAL', 64:
'NORMAL'}
```

get_portState()

Gets the state of one or more ports

Syntax

```
get_portState(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

`port(Int` or `Str)`

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Example:

```
{1: 'down', 5: 'down', 9: 'down', 13: 'down', 17: 'down', 18: 'down', 19: 'down', 20:
'down', 21: 'down',
22: 'down', 23: 'down', 24: 'down', 25: 'down', 26: 'down', 27: 'down', 28: 'down', 29:
'down', 30: 'disabled',
59: 'down', 60: 'down', 61: 'down', 62: 'down', 63: 'down', 64: 'down', 'MGT': 'up'}
```

get_vlagInfo()

Gets VLAG operating information

Syntax

```
get_vlagInfo(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Example:

```
{'Auto Recovery Interval': None, 'Administrative role': None, 'Peer healthcheck IP':
None, 'Local priority': None,
'Peer priority': None, 'VLAG Tier ID': None, 'Operational role': None, 'ISL state':
None, 'Vlag state': 'disabled',
'Local MAC': None, 'VLAG system MAC': None, 'VlagId None': {'ConfigInfo': None,
'State': None,
'Associated trunk': None}, 'Startup Delay Interval': None, 'Health local': None, 'Peer
MAC': None,
'VLAG Healthcheck State': None, 'ISL trunk id': None}
```

get_vlagStats()

Gets VLAG statistics

Syntax

```
get_vlagStats(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Example

```
{'SendSystemInfo': 0, 'SendOther': 0, 'SendFDBInactiveAdd': 0, 'SendUnknown': 0,
'SendFDBDynAdd': 0, 'RecPeerInstanceDisable': 0,
'RecISLHello': 0, 'RecHealthCheck': 0, 'RecPeerInstanceEnable': 0, 'RecFDBDynAdd': 0,
'SendISLHello': 0, 'RecUnknown': 0,
'RecFDBInactiveAdd': 0, 'SendFDBDynDel': 0, 'SendRoleElection': 0, 'RecSystemInfo': 0,
'RecFDBDynDel': 0, 'RecRoleElection': 0,
'IGMPReports': 0, 'RecFDBInactiveDel': 0, 'SendPeerInstanceDisable': 0, 'IGMPLeaves':
0, 'SendPeerInstanceEnable': 0, 'SendHealthCheck': 0, 'SendFDBInactiveDel': 0,
'RecOther': 0}
```

get_vlagTrunk()

Gets VLAG trunk information

Syntax

```
get_vlagTrunk(*args, **kwds)
```

Mandatory Arguments

`trunkType (Str)` in `['lacp', 'portchannel'], trunkId (Int)` is `lacp admin key` or `portchannel Identifier`

Optional Arguments

None

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

get_vlans()

Gets all VLAN information

Syntax

```
get_vlans(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

`vlanId=' '`

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Example

```
{'PVLAN': {1: {'Status': 'enabled', 'Protocol': '2', 'EtherType': 'effe', 'Priority':
'0', 'PVLAN': '802', 'Ports': '21-23 25-27 29-31 33-35 37-39 41-43 45-47 49-51 53-55
57-59 61 62 64', 'FrameType': 'SNAP'},
2: {'Status': 'enabled', 'Protocol': '1', 'EtherType': '86dd', 'Priority': '0',
'PVLAN': '801', 'Ports': '40', 'FrameType': 'Ether2'},
3: {'Status': 'enabled', 'Protocol': '7', 'EtherType': 'f0f0', 'Priority': '0',
'PVLAN': '801', 'Ports': '42', 'FrameType': 'LLC'},
4: {'Status': 'disabled', 'Protocol': '3', 'EtherType': '6004', 'Priority': '0',
'PVLAN': '803', 'Ports': 'empty', 'FrameType': 'Ether2'}},
'Regular': {1: {'Status': 'ena', 'VLAN': '1', 'Name': 'Default VLAN', 'Ports': '1 2 4 5
7 10-14 16 20-24 26 28-33 35 39-44 46-49 53-64'},
2: {'Status': 'ena', 'VLAN': '2', 'Name': 'VLAN 2', 'Ports': '3 6 8 9 15 17-19 25 27 34
36-38 45 50-52'},
4: {'Status': 'ena', 'VLAN': '100', 'Name': 'VLAN 100', 'Ports': 'empty'},
5: {'Status': 'ena', 'VLAN': '701', 'Name': 'VLAN 701', 'Ports': '25 27 29 30 32 34 35
37 39 41 43 45 47 49 51'},
6: {'Status': 'ena', 'VLAN': '702', 'Name': 'VLAN 702', 'Ports': 'empty'},
8: {'Status': 'ena', 'VLAN': '4095', 'Name': 'Mgmt VLAN', 'Ports': 'MGT'}},
'Private': {1: {'SecondaryVlan': '100', 'PrimaryVlan': '700', 'Type': 'community',
'Ports': 'empty'},
2: {'SecondaryVlan': '701', 'PrimaryVlan': '700', 'Type': 'community', 'Ports':
'empty'},
3: {'SecondaryVlan': '', 'PrimaryVlan': '705', 'Type': 'primary', 'Ports': 'empty'},
4: {'SecondaryVlan': '702', 'PrimaryVlan': '', 'Type': 'isolated', 'Ports': 'empty'}}}
```

Layer 3 Module

cfg_addIpInf()

Adds an IP interface

Syntax

```
cfg_addIpInf(*args, **kwds)
```

Mandatory Arguments

`ifNo(Int)`, `address(Str)`, `mask(Str)`, `enable(Boolean)`

Optional Arguments

```
vlanId = 1 (Int)
```

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

cfg_addLoopInf()

Configures loopback interfaces

Syntax

```
cfg_addLoopInf((*args, **kwds)
```

Mandatory Arguments

```
ifNo(Int), address(Str), mask(Str), enable(Boolean)
```

Optional Arguments

None

Output

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

cfg_addRouterInf()

Configures a router port

Syntax

```
cfg_addRouterInf(*args, **kwds)
```

Mandatory Argument

```
port(Int), address(Str), mask(Str), enable(Boolean)
```

Optional Arguments

None

Output

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

cfg_addStaticArp()

Adds a static ARP entry

Syntax

```
cfg_addStaticArp(*args, **kwds)
```

Mandatory Arguments

```
ipAddr(Str), mac(Str), vlan(Int)
```

Optional Arguments

```
port = ''
```

Output

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

cfg_addStaticRoute()

Adds a static route entry

Syntax

```
cfg_addStaticRoute(*args, **kwds)
```

Mandatory Arguments

```
destIp(Str), destMask(Str), gateway(Str)
```

Optional Arguments

```
priServer='', secServer='', domain='', priPort='mgt-port' ,
secPort='mgt-port'
```

Output

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

cfg_dns()

Configures or removes (with an empty string as an argument) a DNS server. To remove DNS servers, set them to empty string

Note: This function supports IPv4 addresses only.

Syntax

```
cfg_dns(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

```
priServer='', secServer='', domain='', priPort='mgt-port' ,
secPort='mgt-port'
```

Output

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

cfg_gwHealthCheck()

Enables gateway health checks using ARP

Syntax

```
cfg_gwHealthCheck(*args, **kwds)
```

Mandatory Arguments

`gwNo` (Int)

Optional Arguments

Enable=True

Output

Boolean (True on success, otherwise False)

cfg_ipRouting()

Enables IP routing options

Syntax

```
cfg_ipRouting(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

```
enable=True, enableBroad=False, enableIcmp6=False,
enableNoIcmpRedir=False
```

Output

Boolean (True on success, otherwise False)

cfg_ipv4Gateway()

Adds an IPv4 gateway

Syntax

```
cfg_ipv4Gateway(*args, **kwds)
```

Mandatory Arguments

```
gwNo(Int), gwAddr(Str)
```

Optional Arguments

Enable=True

Output

Boolean (True on success, otherwise False)

cfg_remIpInf()

Removes an IP interface

Syntax

```
cfg_remIpInf(*args, **kwds)
```

Mandatory Arguments

`ifNo(Int)`

Optional Arguments

None

Output

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

cfg_remIpv4Gateway()

Removes an IPv4 gateway

Syntax

```
cfg_remIpv4Gateway(*args, **kwds)
```

Mandatory Arguments

`gwNo(Int)`

Optional Arguments

None

Output

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

cfg_remLoopInf()

Removes a loopback interface

Syntax

```
cfg_remLoopInf(*args, **kwds)
```

Mandatory Arguments

`loopIfNo(Int)`

Optional Arguments

None

Output

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

cfg_remRouterInf

Removes a router interface

Syntax

```
cfg_remRouterInf(*args, **kwds)
```

Mandatory Arguments

`port(Int)`

The port number of the router interface

Optional Arguments

None

Output

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

cfg_remStaticArp()

Removes a static ARP entry

Syntax

```
cfg_remStaticArp(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

`ipAddr='all'`

Output

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

cfg_remStaticRoute()

Removes static routes. If no arguments are defined, it removes all static routes

Syntax

```
cfg_remStaticRoute(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

`destIp=' ', destMask=' ', gateway=' ', port=' , ' ,fNo=' '`

Output

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

clr_arpCache()

Clears the ARP cache

Syntax

```
clr_arpCache(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

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

clr_ipCounters()

Clears IP statistics

Syntax

```
clr_ipCounters(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

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

get_intIp()

Returns a list of IP interfaces configured on the G8264

Syntax

```
get_intIp(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Example:

```
{'LoopInf': {'lo2': {'broadcast': '20.20.20.255', 'status': 'up', 'mask':
'255.255.255.0', 'address': '20.20.20.20'},
'lo3': {'broadcast': '8.8.8.255', 'status': 'up', 'mask': '255.255.255.0', 'address':
'8.8.8.8'}},
'IP4Inf': {'1': {'broadcast': '20.1.1.255', 'status': 'DISABLED', 'vlan': '1', 'mask':
'255.255.255.0', 'address': '20.1.1.1'}, '10':
'20': {'broadcast': '2.255.255.255', 'routerInf': True, 'status': 'DOWN', 'mask':
'255.0.0.0', 'address': '2.1.1.1'},
'128': {'broadcast': '9.70.2.255', 'status': 'up', 'vlan': '4095', 'mask':
'255.255.255.0', 'address': '9.70.2.101'}},
'IP6Inf': {'20': {'status': 'up', 'vlan': '5', 'anycast': '', 'localaddress':
'fe80::a17:f4ff:feaf:9b13', 'address': '3001:0:0:0:0:0:0:24'},
'22': {'status': 'up', 'vlan': '1', 'anycast': 'anycast', 'localaddress':
'fe80::a17:f4ff:feaf:9b15', 'address': '2001:0:0:0:0:0:0:12'},
'127': {'status': 'up', 'vlan': '4095', 'anycast': '', 'localaddress':
'fe80::a17:f4ff:feaf:9b7c', 'address': '0:0:0:0:0:0:0:0'}}
```

get_ipArp()

Gets IP ARP entries

Syntax

```
get_ipArp(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

```
display=”all”
```

Output

Dictionary on success, dictionary with `None` values for empty output, `False` on failure

Example:

```
{0: {'macAddress': '00:33:44:ed:45:3e', 'age': '48', 'vlan': '48', 'flag': 'P',
'ipAddress': '48.48.48.1', 'port': ''},
1: {'macAddress': '08:17:f4:af:9b:00', 'age': '', 'vlan': '48', 'flag': 'P',
'ipAddress': '48.48.48.8', 'port': ''},
2: {'macAddress': '00:00:06:00:03:00', 'age': '0', 'vlan': '48', 'flag': '',
'ipAddress': '48.48.48.10', 'port': '48'},
3: {'macAddress': '00:00:06:00:03:00', 'age': '0', 'vlan': '48', 'flag': '',
'ipAddress': '48.48.48.13', 'port': '48'},
4: {'macAddress': '00:33:44:ed:10:10', 'age': '48', 'vlan': '48', 'flag': 'P',
'ipAddress': '48.48.48.100', 'port': ''},
```

```
5: {'macAddress': '08:17:f4:af:9b:00', 'age': '', 'vlan': '1', 'flag': 'P',
'ipAddress': '172.16.1.128', 'port': ''},
6: {'macAddress': '08:17:f4:af:9b:00', 'age': '', 'vlan': '2', 'flag': 'P',
'ipAddress': '172.16.2.128', 'port': ''},
7: {'macAddress': '08:17:f4:af:9b:00', 'age': '', 'vlan': '3', 'flag': 'P',
'ipAddress': '172.16.3.128', 'port': ''},
8: {'macAddress': '08:17:f4:af:9b:00', 'age': '', 'vlan': '4', 'flag': 'P',
'ipAddress': '172.16.4.128', 'port': ''}}
```

cfg_addAggregateAddr()

Add BGP aggregation address range

Syntax

```
cfg_addAggregateAddr(*args, **kwds)
```

Mandatory Arguments

```
aggId (Int), aggAddr (Str), aggMask (Str)
```

Optional Arguments

```
enable = True (Bool)
```

Output

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

cfg_addBgpNeighborGroup()

Add BGP peering group

Syntax

```
cfg_addBgpNeighborGroup(*args, **kwds)
```

Mandatory Arguments

```
groupId (Int), addRange (Str), groupAs (Int)
```

Optional Arguments

```
enable = True (Bool), addMask = ’255.255.255.255’ (Str),
limit = 1 (Int), groupName = ’’ (Str), altAs = ’’ (Str),
reflectorClient=True (Bool), srcInf=0 (Int), srcLoopInf = 0
(Int), nextHopSelf= False (Bool), holdTime=180, keepAlive
=60, adv = 60, routeOrg=15, ttl= 0,ttlSecHops=0, passwd=’’
```

Output

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

cfg_addBgpPeer()

Add BGP peer router

Syntax

```
cfg_addBgpPeer(*args, **kwds)
```

Mandatory Arguments

```
peerId (Int), peerAddr (Str), peerAs (Int)
```

Optional Arguments

```
enable = True (Bool), reflectorClient=False (Bool), srcInf=0
(Int), srcLoopInf=0 (Int) nextHopSelf= False (Bool), passive
= False (Bool), holdTime=180, keepAlive =60, adv = 60,
retry=120, routeOrg=15, ttl= 0,ttlSecHops=0, passwd=''
```

Output

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

cfg_enableBgp()

Configure BGP global parameters

Syntax

```
cfg_enableBgp(*args, **kwds)
```

Mandatory Arguments

asNo (Int)

Optional Arguments

```
enable = True (Bool), reflection = True (Bool), localPref =
100 (Int),  dscp = 0 (Int), clusterId ='0.0.0.0' (Str)
```

Output

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

cfg_routerId()

Configure router ID

Syntax

```
cfg_routerId(*args, **kwds)
```

Mandatory Arguments

```
routerId (Str)
```

Optional Arguments

None

Output

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

get_bgp()

This API gets BGP current configuration

Syntax

```
get_bgp(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Boolean Dictionary on success, dict. with None values for empty output, False on failure

get_bgpAggregateAddress()

This API gets BGP aggregate addresses

Syntax

```
get_bgpAggregateAddress(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

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

get_bgpNeighborsInfo()

This API gets BGP neighbors operating information

Syntax

```
get_bgpNeighborsInfo(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Dictionary on success, dict. with None values for empty output, False on failure

get_bgpRoutes()

This API gets BGP routing information

Syntax

```
get_bgpRoutes(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Dictionary on success, dict. with None values for empty output, False on failure

cfg_remAggregateAddr()

Remove BGP aggregation address range

Syntax

```
cfg_remAggregateAddr(*args, **kwds)
```

Mandatory Arguments

`aggId (Int)`

Optional Arguments

None

Output

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

cfg_remBgpNeighborGroup()

Remove BGP peering group

Syntax

```
cfg_remBgpNeighborGroup(*args, **kwds)
```

Mandatory Arguments

```
groupId (Int)
```

Optional Arguments

None

Output

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

cfg_remBgpPeer()

Delete BGP remote peer

Syntax

```
cfg_remBgpPeer(*args, **kwds)
```

Mandatory Arguments

peerId (Int)

Optional Arguments

None

Output

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

cfg_addVrrpRouter()

Adds a VRRP virtual router and its parameters

Syntax

```
cfg_addVrrpRouter(*args, **kwds)
```

Mandatory Arguments

```
vrIndex (Int), vRouterId (Int), vAddr (Str), ipInt (Int)
```

Optional Arguments

```
enable = True (Bool), priority = 100 (Int), adv = 1 (Int),
preemptDelay = 0 (Int), preemption = True (Bool),
preemptPriOnly = False (Bool), fastAdv = False (Bool)
```

Output

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

cfg_remVrrpRouter()

Removes a VRRP virtual router

Syntax

```
cfg_remVrrpRouter(*args, **kwds)
```

Mandatory Arguments

vrIndex (Int)

Optional Arguments

None

Output

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

cfg_vrrpGroupTracking()

Configures VRRP group tracking

Syntax

```
cfg_vrrpGroupTracking(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

`trackType = 'interface'` in (`'interface','port'`), `enable = True`

Output

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

cfg_vrrpInfAuthen()

Enables or disables VRRP interface authentication

Syntax

```
cfg_vrrpInfAuthen(*args, **kwds)
```

Mandatory Arguments

`infNo (Int)`

Optional Arguments

```
enable = True, authenType = 'none' (Str) in ('password',
'none'), passwd = '' (Str)
```

Output

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

cfg_vrrpRouterGroup()

Adds a VRRP virtual router group

Syntax

```
cfg_vrrpRouterGroup(*args, **kwds)
```

Mandatory Arguments

```
gVRouterId (Int), ipInt (Int)
```

There are no mandatory and optional arguments needed when the `noGroup` argument is set to `True`.

Optional Arguments

```
noGroup = False, enable = True (Bool), restricted = False,
priority = 100 (Int), adv = 1 (Int), preemptDelay = 0, fastAdv
= False, preemption = True (Bool), priOnlyPreempt = False
(Bool)
```

Output

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

cfg_vrrpRouterTracking()

Configures VRRP interface tracking

Syntax

```
cfg_vrrpRouterTracking(*args, **kwds)
```

Mandatory Arguments

```
vrIndex (Int)
```

Optional Arguments

```
trackType = 'interface' (Str) in
('interface','nhop','port','vRouter'), vrTrackPriIncr = 2,
intTrackPriIncr = 2, nhTrackPriIncr = 2, pTrackPriIncr = 2
```

Output

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

cfg_enableVrrp()

Enables or disables VRRP globally and configures VRRP global parameters

Syntax

```
cfg_enableVrrp(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

```
enable = True (Bool), holdOff = 0 (Int)
```

Output

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

get_vrrp()

Gets the VRRP configuration

Syntax

```
get_vrrp(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

A dictionary of VRRP configuration

get_vrrpGroupInfo()

Gets VRRP tracking information

Syntax

```
get_vrrpGroupInfo(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

A dictionary of VRRP group information

get_vrrpInfo()

Gets VRRP virtual router information

Syntax

```
get_vrrpInfo(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

A dictionary of virtual router information with routerIndex as keys

get_vrrpRouter()

Gets VRRP virtual router information

Syntax

```
get_vrrpRouter(*args, **kwds)
```

Mandatory Arguments

```
vrIndex (Int)
```

Optional Arguments

None

Output

A dictionary of virtual router information

cfg_addIgmpFilter()

Adds IGMP filtering profile on switch port

Syntax

```
cfg_addIgmpFilter(*args, **kwds)
```

Mandatory Arguments

profileId (Int), port (Int), addrRange (Str) in the format of "startIp endIp"

Optional Arguments

action = 'allow' (Str) in ('deny', 'allow')

Output

Boolean (True on success, otherwise False)

cfg_addIgmpMRouter()

Adds IGMP mRouter port(s) for a snooping Vlan

Syntax

```
cfg_addIgmpMRouter(*args, **kwds)
```

Mandatory Arguments

ports (Str, Int), vlanId (Int), routerVer (Int)

Optional Arguments

None

Output

Boolean (True on success, otherwise False)

cfg_addIgmpQuerierVlan()

Enables IGMP querier on a VLAN or a group of VLANs

Syntax

```
cfg_addIgmpQuerierVlan(*args, **kwds)
```

Mandatory Arguments

vlanIds (Int or Str), sourceIp (Str)

Optional Arguments

enable = True (Bool), electBy = ipv4 (Str in ['ipv4', 'mac']
maxResponse = 100 (Int), queryInterval = 125 (Int),
robustness = 2 (Int), startupCount = 2 (Int) startupInterval
= 31 (Int), version = 3 (Int)

Output

Boolean (True on success, otherwise False)

cfg_addIgmpRelayMRouter()

Configures IGMP relay mRouter

Syntax

```
cfg_addIgmpRelayMRouter(*args, **kwds)
```

Mandatory Arguments

```
mrId (Int), ipAddr (Str)
```

Optional Arguments

```
enable = True (Bool), ver = 2 (Int), interval =2 (Int), retry
= 4 (Int), attempt = 5 (Int)
```

Output

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

cfg_addIgmpRelayVlan()

Enables IGMP relay for VLANs

Syntax

```
cfg_addIgmpRelayVlan(*args, **kwds)
```

Mandatory Arguments

```
vlanIds (Int or Str)
```

Optional Arguments

None

Output

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

cfg_addIgmpSnoopingVlan()

Enables IGMP Snooping for a single or a group of VLANs.

Syntax

```
cfg_addIgmpSnoopingVlan(*args, **kwds)
```

Mandatory Arguments

```
vlanIds (Int or Str)
```

Optional Arguments

None

Output

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

cfg_enableIgmp()

Enables or disables IGMP on the local switch and configure global IGMP parameters

Syntax

```
cfg_enableIgmp(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

```
enable = True (Bool), aggregation = True (Bool), rtrAlert =
False (Bool), robust = 2 (Int),  timeout = 10 (Int),
queryInterval = 125 (Int), fastLeavePort = 0 (int),
fastLeaveVlan = 0 (int, str)
```

Note: When enable = `False`, the rest of the arguments will be ignored.

Output

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

cfg_enableIgmpFilter()

Enables or disables IGMP filtering globally

Syntax

```
cfg_enableIgmpFilter(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

```
enable = True (Bool)
```

Output

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

cfg_enableIgmpQuerier()

Enables or disables IGMP querier on the local switch

Syntax

```
cfg_enableIgmpQuerier(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

```
enable = True (Bool)
```

Output

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

cfg_enableIgmpRelay()

Enables or disables IGMP relay globally on the local switch

Syntax

```
cfg_enableIgmpRelay(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

```
enable = True (Bool)
```

Output

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

cfg_enableIgmpSnoop()

Enables or disables IGMP snooping on the local switch and configure snooping parameters

Syntax

cfg_enableIgmpSnoop(*args, **kwds)

Mandatory Arguments

None

Optional Arguments

```
enable = True (Bool), sourceIp = '' (Str), mRTimeout = 255
(Int)
```

Output

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

cfg_igmpV3Snooping()

Enables or disables IGMPv3 snooping and configures IGMPv3 snooping parameters

Syntax

```
cfg_igmpV3Snooping(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

enable = True (Bool), isExclude = True (Bool), srcNum = 8 (Int), enableV1v2 = True (Bool)

Output

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

cfg_remIgmpFilter()

Removes IGMP filtering profile on switch port

Syntax

```
cfg_remIgmpFilter(*args, **kwds)
```

Mandatory Arguments

```
profileIds (Int, Str)
```

Optional Arguments

```
ports = 0 (Int, Str).
```

Note: If `ports=0`, per port IGMP filtering commands will not be executed

Output

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

cfg_remIgmpMRouter()

Removes IGMP mRouter port of a Vlan

Syntax

```
cfg_remIgmpMRouter(*args, **kwds)
```

Mandatory Arguments

If argument `all = False` (default), these arguments are required:

`ports (Str, Int), vlanId (Int), routerVer (Int)`

Optional Arguments

`all = False`

Output

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

cfg_remIgmpQuerierVlan()

Disables IGMP querier on VLANs

Syntax

`cfg_remIgmpQuerierVlan(*args, **kwds)`

Mandatory Arguments

`vlanIds (Int` or `Str)`

Optional Arguments

None

Output

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

cfg_remIgmpRelayMRouter()

Removes IGMP relay mRouter

Syntax

`cfg_remIgmpRelayMRouter(*args, **kwds)`

Mandatory Arguments

`mrId (Int)`

Optional Arguments

None

Output

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

cfg_remIgmpRelayVlan()

Removes IGMP relay configuration for Vlans

Syntax

```
cfg_remIgmpRelayVlan(*args, **kwds)
```

Mandatory Arguments

`vlanIds (Int` or `Str)`

Optional Arguments

None

Output

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

cfg_remIgmpSnoopingVlan()

Disables IGMP snooping on a VLAN or on a group of VLANs

Syntax

```
cfg_remIgmpSnoopingVlan(*args, **kwds)
```

Mandatory Arguments

`vlanIds (Int` or `Str)`

Optional Arguments

None

Output

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

get_igmpFilter()

Gets the IGMP filtering configuration

Syntax

```
get_igmpFilter(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

A dictionary of IGMP filtering profiles and applied ports

get_igmpGroups()

Gets IGMP groups learnt by IGMP snooping

Syntax

```
get_igmpGroups(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

`filter = ('vlan',1)` is a tuple of `(filterType, value)` with `filterType` in `('vlan', 'addr', 'interface', 'portch')`

Output

A dictionary with group addresses as keys or an empty dictionary if no group learnt

get_igmpMRouter()

Gets IGMP mRouter operating information

Syntax

```
get_igmpMRouter(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

`filter = (filterType, value)` (Tuple) with `filterType` in `('vlan', 'static', 'dynamic', 'interface','portch')`

Note: No values are required if `filterType` is "`static`" or "`dynamic`"

Output

A list of mRouter parameters as a dictionary

get_igmpQuerier()

Gets IGMP querier settings

Syntax

```
get_igmpQuerier(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

A dictionary of IGMP querier settings

get_igmpRelay()

Gets IGMP relay configuration

Syntax

```
get_igmpRelayInfo(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

A dictionary of mRouters and their settings

get_igmpSnooping()

Gets IGMP snooping settings

Syntax

```
get_igmpSnooping(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

A dictionary of IGMPv2 and IGMPv3 snooping settings

ACL Module

cfg_aclEthernet()

Configures an Ethernet access control list

Syntax

```
cfg_aclEthernet(*args, **kwds)
```

Mandatory Arguments

```
aclNo (int) , ethField (str) in ['desMAC', 'srcMAC',
'ethType', 'prio',' vlan'], action (Str) in ['deny', 'permit']
```

Optional Arguments

```
desMAC='00:01:02:03:04:05', srcMAC='00:01:02:03:04:05',
mask='', ethType='any'in ['any', 'arp', 'ip', 'ipv6', 'mpls',
'rarp'], prio=0, vlan=1
```

Output

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

cfg_aclIpv4()

Configures an IPv4 access control list

Syntax

```
cfg_aclIpv4(*args, **kwds)
```

Mandatory Arguments

```
aclNo (Int) , ipv4Field (Str) in ['desIP', 'srcIP', 'proto',
'tos'],action (Str) in ['deny', 'permit']
```

Optional Arguments

```
desIP='0.0.0.0', srcIP='0.0.0.0', mask='0.0.0.0', proto=0,
tos=0
```

Output

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

cfg_assignAcl()

Assigns an ACL to the specified ports.

Syntax

```
cfg_assignAcl(*args, **kwds)
```

Mandatory Arguments

`aclNo(Int), ports(Int` or `Str). 'ports'` can be a single port (Int) or a port range (Str)

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_delAcl()

Deletes ACLs

Syntax

```
cfg_delAcl(*args, **kwds)
```

Mandatory Arguments

`aclNo(Int)`

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

cfg_unassignAcl

Description

Removes an ACL from the specified ports.

Syntax

```
cfg_unassignAcl(*args, **kwds)
```

Mandatory Arguments

`aclNo(Int), ports(Int or Str). 'ports'` can be a single port (Int) or a port range (Str)

Optional Arguments

None

Output

`True` if the command is successful; otherwise `False`

System-Level Modules

There are two groups of system modules:

- Group1: Contains functions in `pylib.objPylib` object, including `cmd()`, `get_analyzer()`, `echo()`, and `sleep()`
- Group2: Contains functions in `pylib` module, including `sendSyslog()`, `set_var()`, `get_var()`, and `get_event()`

cmd()

This function is the CLI wrapper to send ISCLI commands to the G8264.

Syntax

```
pylib.objPylib.cmd(*cmd, **kwargs)
```

Mandatory Arguments

`cmd (Str)`

Optional Arguments

`Timeout=60 (Int)`, in the range of `5-sys.maxint`, is the maximum time in seconds to process the CLI.

`Terminator='\n'` is the command line terminator.

`Prompt (Str)` is a regular expression to match CLI prompts.

If `Prompt` is not set, the default prompts will match all possible ISCLI prompts

Output

`analyzer` object for further processing or `None` on errors

get_analyzer()

Create new `analyzer` object for the provided inspected string

Syntax

```
pylib.objPylib.get_analyzer(inspectStr, **kwargs)
```

Mandatory Arguments

`inspectStr (Str)`

Optional Arguments

None

Output

`analyzer` object for further processing or `None` on errors

sendSyslog()

Syntax

```
pylib.sendSyslog(message, priority=6, **kwargs)
```

Mandatory Arguments

`message (Str)` is the `syslog` message with maximum length of 128 characters. If the message exceeds 128 chars, it will be truncated to 128 chars.

Optional Arguments

`priority (Int)` is logging level with valid values from 0 to 7. The priority values are defined in

`pylib.LOG_LEVEL.LOG_EMERG`, `pylib.LOG_LEVEL.LOG_ALERT`, `pylib.LOG_LEVEL.LOG_CRIT`, `pylib.LOG_LEVEL.LOG_ERR`, `pylib.LOG_LEVEL.LOG_WARNING`, `pylib.LOG_LEVEL.LOG_NOTICE`, `pylib.LOG_LEVEL.LOG_INFO`, `pylib.LOG_LEVEL.LOG_DEBUG`

Output

Boolean (`True` if send command successfully; otherwise `False`)

set_var()

Get the value of persistent variable.

Syntax

```
set_var(*cmd, **kwargs)
```

Mandatory Arguments

`varName (Str)`

Note: The variable name to store in the `objValue`, the length must not exceed 31 characters. This function will display the following message:

`ValueError (var. name too long), NameError (var. name not found).`

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

get_var()

Get the value of a persistent variable

Syntax

```
get_var(*cmd, **kwargs)
```

Mandatory Arguments

`varName (Str)` is the variable name to store the `objValue`; the length must not exceed 31 characters.

`objValue (object)` will be serialized into a string format, and this string must not be larger than 3039 characters.

After the serialization, the value string is an empty string; the variable, if exists, will be deleted.

Only 20 persistent variables are supported. This method will raise the following exception `ValueError (variable or value length exceed)`, `MemoryError (Maximum persistent var. exceed)`.

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

get_event()

Get the event information as a tuple

Syntax

```
pylib.get_event(*args, **kwds)
```

Mandatory Arguments

None

Optional Arguments

None

Output

Boolean (`True` if send command successfully; otherwise `False`)

echo()

Write a message to a log file

Syntax

```
echo(*arg, **kwds)
```

Mandatory Arguments

```
msg (Str)
```

Optional Arguments

None

Output

None

sleep()

Sleep for '*secs*' seconds.

If `ver=True`, the method will also write to log-file the following message:

`"Sleeping for` *n* `second(s)"`

Syntax

```
sleep (*args, **kwds)
```

Mandatory Arguments

```
secs (float)
```

Optional Arguments

```
ver (Bool)
```

Output

Boolean (`True` if send command successfully; otherwise `False`)

Script Examples

This section provides a set of sample `imbpylib` API module scripts.

VLAN Configuration Script Example

This script configures VLANs.

```
# Import the pylib and API modules from pylib import pylib, mgmt
''' Create Pylib object ''' Pylib = pylib.objPylib

# Open CLI session
mgmt.exe_login()

     Pylib.echo('======> Test large output command, check CPU utilization <======')
     # Go to global configuration mode
     Pylib.cmd('conf terminal')


# Add vlans from 2 to 4
Pylib.cmd('vlan 2-4') Pylib.cmd('enable') Pylib.cmd('exit')

re = Pylib.cmd('show version')

print re.get_output()

if re.contains(reg='7.8.1.0'):
    print 'The Sw is running 7.8.1l0'
else:
     print 'not correct'

# Create IP interface from 1 to 124, assign IP address, vlan then enable it
for inf in range(1, 5):
    Pylib.cmd('interface ip {ifNo}'.format(ifNo=inf))
    Pylib.cmd('ip address 172.16.{ifNo}.128 255.255.255.0
enable'.format(ifNo=inf))
    Pylib.cmd('vlan {vlanId}'.format(vlanId=inf))

Pylib.cmd('end')# Go back to privilege mode

Pylib.cmd('show running')# Show running configuration

# Close CLI session
mgmt.exe_logout()
```

Monitoring Link Status Example

This script runs every two minutes to check if the RX watts on a long-range transceiver goes below a specified threshold.

```
# Import the pylib and API modules from pylib import pylib, mgmt import re, sys
# Get Pylib object for later use
Pylib = pylib.objPylib

# Set the notify user
notifyList = ['user@abc.com']
smtpAddress = 'smtp.abc.com'

# Define uplink port to monitor
uplinkPort = 48
uplinkRegex  = '.*link\s+down\s+on\s+port\s+(\d+)'

# Get the link link event
event = pylib.get_event()
if not event:
    errMsg = 'This script only works with scheduler job'
    Pylib.echo(errMsg)
    print (errMsg)
     sys.exit()

mgmt.exe_login()# Open CLI (ISCLI Only) session to the switch

hostname = mgmt.get_hostName()

# Configure something
Pylib.echo('======> Monitor uplink port <=======')

if event['type'] == 'syslog':
    Pylib.echo('Syslog event detected' + str(event['msg']))

    ana = Pylib.get_analyzer(event['msg'])
    downPort = ana.store(reg = uplinkRegex, flag = re.S)
    Pylib.echo('Down port is:' + str(downPort))
    if downPort:
        if int(downPort) == uplinkPort: Pylib.cmd('Matched')
        # The uplink port is really down => send notification
        mgmt.exe_sendMail(smtpSrv=smtpAddress,
        subj='Uplink on switch {swname} was down'.format(swname = hostname),
        sender='', body='Uplink on port {upport} was down.
Please take a look'.format(upport = uplinkPort),lstRcp=notifyList,
attFiles=[Pylib.logFile])
mgmt.exe_logout()# Close CLI session
```

Example of Adding and Removing ACL Lists

This script uses API functions to add and remove access control lists.

```
# Import the pylib and API modules
from pylib import pylib, mgmt, acl
''' Create Pylib object ''' Pylib = pylib.objPylib


# Set the notify user
notifyList = ['user@abc.com']
smtpAddress = 'smtp.abc.com'
```

```
Pylib.echo('======> Test some query API functions. Make sure we can get the desired
information <======')

mgmt.exe_login()# Open CLI (ISCLI Only) session to the switch

# Get some information from the switch
hostname = mgmt.get_hostName()# get switch hostname
Pylib.echo("Switch\'s hostname: {0}".format(hostname))

mac = mgmt.get_swMac()
Pylib.echo("Switch\'s Mac address: {0}".format(mac))

ver = mgmt.get_curVersion()
Pylib.echo("Switch\'s current running version: {0}".format(ver))

# Configure something
Pylib.echo('======> ACL IPv4 and ethernet API functions <======')

# Add IPv4 access list 102
acl.cfg_aclIpv4(aclNo=102, ipv4Field='desIP', action='permit', desIP='9.70.22.35',
mask='255.255.255.252')
# Add Ethernet access list 105
acl.cfg_aclEthernet(aclNo=105, ethField='desMAC', action='deny',
desMAC='00:ac:04:67:de:1d')

# Assign access list to port(s)
acl.cfg_assignAcl(aclNo=102, ports='35,36')
acl.cfg_assignAcl(aclNo=105, ports=45)

# Show the running configuration using CLI method config1 = Pylib.cmd('show run')
if config1.contains(reg='access-control\s+list\s+105\s+ethernet.*'):
    Pylib.echo('====> Access list 105 configuration was added to the switch
successfully')
else:
    Pylib.echo('====> Failed to add access list 105 configuration to the switch')

# Unassign access list to port(s)
acl.cfg_unassignAcl(aclNo=102, ports='35,36')
acl.cfg_unassignAcl(aclNo=105, ports=45)

# Delete access lists acl.cfg_delAcl(aclNo=102) acl.cfg_delAcl(aclNo=105)

# Show the running configuration again
config2 = Pylib.cmd('show run')
if config2.contains(reg='access-control\s+list\s+105\s+ethernet.*'):
    Pylib.echo('====> Failed to remove access list 105 configuration from the
switch')
else:
    Pylib.echo('====> Access list 105 configuration was removed successfully')

# Send notification email to administrator with execution log file as an attachment
# If smtpSrv is in a FQDN, a DNS server must be configured on the switch to resolve
the name to IP
```

```
mgmt.exe_sendMail(smtpSrv=smtpAddress, subj='Mail from pylib script conf_query.py',
body='Test mail with attachment - Use FQDN for SMTP server', lstRcp=notifyList,
attFiles=[Pylib.logFile], sender='')

mgmt.exe_logout()
# Close CLI session
```

Setting a Persistent Variable Example

This script sets a persistent variable.

```
# Import modules
from pylib import pylib, mgmt

# Create the Pylib object. Note that this object can have any name.
Pylib = pylib.objPylib

# Open CLI session
mgmt.exe_login()

# Set a persistent variable
pylib.set_var('aName', 'aValue')

# Get the value of an existent persistent variable
value = pylib.get_var('aName')
print value

# Get information related to the event that triggered current script execution.
# Note that for this function to return a non-Null value, the script must be
# configured in a Scheduler Job and the configured event must be triggered.
eventTuple = pylib.get_event()
print eventTuple

# Close CLI session
mgmt.exe_logout()
```

of sample `imbpylib` API module scripts.

Chapter 7. Executing ISCLI Commands in Scripts

The Lenovo Networking OS (N/OS) Python CLI libraries provide mechanisms to send ISCLI commands to the RackSwitch G8264 and return an `analyzer` object. Functionally, it is the same as entering commands manually to the G8264 console. For example, you can use an `analyzer` object to learn the name of the G8264.

The N/OS Python wrapper method performs the following actions:

- Takes a CLI command string as the input argument
- Sends the CLI command to N/OS
- Creates an `analyzer` object based on the output of the CLI command from N/OS and returns it to the calling script

You can also get the `analyzer` object by calling the wrapper method `pylib.objPylib.get_analyzer(inspectString)`, which creates a new `analyzer` object to inspect the provided string.

```
get_analyzer(inspectStr, **kwargs): ?
Input: inspectStr (Str) inspected string?
Output: analyzer object
```

The object returned by this CLI method is called the `analyzer` object.

- The CLI method will return an `analyzer` object on success, otherwise, it returns
- `False`. The `analyzer` object has the following methods:
- `extract(**kwargs)`: store a value in the output into a variable
- `contains(**kwargs)`: check if the output contains a specified value
- `count(**kwargs)`: count the occurrences of a value in the output
- `get_output(**kwargs):` get the output string
- `split(**kwargs)`: split the inspected string by pattern occurrences
- `substitute(**kwargs)`: substitute the matched pattern with a replacement string

Python Analyzer Object Examples

The following examples illustrate simple methods.

- An example of the `get_analyzer` method to see if a link is up:

```
>>> ana = Pylib.get_analyzer('Nov 14 17:09:23 9.70.2.102 NOTICE link: link up on
port 35')
>>> ana.contains(reg='.*link up on port \d+')
True
```

- An example of the `store` method to get version number:

```
Analyzer = Pylib.cmd('show version')
ver = Analyzer.extract(reg='(?i)^software\s+version\s+(\S+)\s.*\(flash.*\).*')
```

- An example of the `contains` method to validate the output:

```
Analyzer = Pylib.cmd('show version') if Analyzer.contains(reg='7.7.3.0'): print
'Correct version detected'
else:
print 'Wrong version'
```

- An example of the `extract` method to get a version number:

```
Analyzer = Pylib.cmd('show version')
ver = Analyzer.extract(reg='(?i)^software\s+version\s+(\S+)\s.*\(flash.*\).*')
=> ver should contains a string like '7.8.0.10'
```

- An example of the `split` method, which splits the inspected string by occurrences of patterns:

```
MacAddr = mgmt.get_swMAC()# MacAddr store '00:DB:45:34:45:C4' for example
MacAnalyzer = Pylib.get_analyzer(MacAddr)
lstOctets = MacAnalyzer.split(reg=':')
=> lstOctets should be a list like ['00', 'DB','45', '34', '45', 'C4']
```

Example Scripts and Output

Here is a basic script that calls the CLI API function and its methods:

```
print "\nImport modules"
from pylib import pylib, mgmt
Pylib = pylib.objPylib

print "\nOpen CLI session" mgmt.exe_login()

print "\nCall the CLI Wrapper function to get the output of 'show vlan' CLI command"
analyzerObj = Pylib.cmd('show vlan')

print "\nUse 'get_output' method to print on screen the command output" cmdOutput =
analyzerObj.get_output()
print cmdOutput

print "\nUse 'extract' method to save the number of the Accounting VLAN"
noOfAccountingVlan = analyzerObj.extract(reg='(\d+)\s+Accounting VLAN') print
"Acounting VLAN number is", noOfAccountingVlan

print "\nUse 'contains' method to check if the 'Accounting VLAN' string is present
in command output"
result = analyzerObj.contains(reg='Accounting VLAN')
print "The method returned the following result:", result

print "\nUse 'count' method to get the number of enabled VLANs on device"
noOfEnabledVlans = analyzerObj.count(reg='\s+ena\s+')
print "There are {0} enabled VLANs on device".format(noOfEnabledVlans)
print "\nClose CLI session" mgmt.exe_logout()
```

Following is the output from running this script:

```
RS G8264# python doc.py

Import modules

Open CLI session

Call the CLI Wrapper function to get the output of 'show vlan' CLI command

Use 'get_output' method to print on screen the command output show vlan
VLAN              Name               Status    Ports
----  ------------------------------  ------ -------------------------
1          Default VLAN                  ena    1 5-64
5          Accounting VLAN               ena    empty
4095       Mgmt VLAN                     ena    MGT
Primary   Secondary  Type               Ports
------- --------- --------------- ---------------------------------------
```

Inspecting Output

Use the `analyzer` wrapper methods to inspect strings for their content:

*extract(**kwargs)*

Find all regular expressions

Arguments

`reg (Str)` is a `regex. string, flag (Int)` is a regex. flag in `[re.S, re.I, re.L, re.M, re.U, re.X, re.DEBUG]`

where

`re` is the standard regular expression module

`type` is the Python data type used for the returned elements.

The default return type is `Int` for numbers or `String` for non-numbers

Returns

Depending on the `re` argument, the return value can be a single value, a tuple, or a list of tuples, or `None` if there is no match.

*contains(**kwargs)*

Checks if the inspected string contains a specific value at least once.

Arguments

`reg (Str)`	regex
`flag (Int)`	one of `[re.S, re.I, re.L, re.M, re.U, re.X, re.DEBUG]` where `re` is standard regular expression module
`splitCount = 0`	the number of specified strings contained. The default value `0` means that no instances of the specified strings are contained.

`reg (Str)` is `regex`.

`string, flag (Int)` is `regex`.

flag in `[re.S, re.I, re.L, re.M, re.U, re.X, re.DEBUG]`

where `re` is a standard regular expression module.

Returns

`True` if the inspected string matches the `regex`; otherwise `False.`

*split(**kwargs)*

Split the inspected string by occurrences of pattern

Arguments

`reg (Str) is regex, string, flag (Int)` is `regex. flag in [re.S, re.I, re.L, re.M, re.U, re.X, re.DEBUG]` where `"re"` is standard regular expression module

Output

A list of strings

*count(**kwargs)*

Counts the regular expression matches

Arguments

`reg (Str)` is `regex. string, flag (Int)` is `regex. flag in [re.S, re.I, re.L, re.M, re.U, re.X, re.DEBUG]` where `"re"` is standard regular expression module, `splitCount = 0` is the number of split (default value 0 means that spliting the string by all occurrences of patterns)

Returns

The number of match(es) found as an integer

*substitute(**kwargs)*

Substitutes the matched pattern with a replacement string

Arguments

`reg (Str)` is regex. `string, flag (Int)` is regex. `flag in [re.S, re.I, re.L, re.M, re.U, re.X, re.DEBUG]` where `"re"` is standard regular expression module, `replaceStr (Str), count = 0` is the number of replacements

Returns

A new string after the substitution

*get_output(**kwargs)*

Returns the inspected string

Arguments

`False`

Returns

The inspection string

Example:

```
Use 'extract' wrapper method to save the number of the Accounting VLAN Accounting
VLAN number is 5

Use 'contains' wrapper method to check if the 'Accounting VLAN' string is present in
command output
The wrapper method returned the following result: True

Use 'count' wrapper method to get the number of enabled VLANs on device
There are 3 enabled VLANs on device
Close CLI session
Python shell or script completed successfully
RS G8264#
```

CLI Wrapper Method Limitations

You can enter any CLI `show` command with the following exceptions:

- `show tech-support`
- `show information`

Note: You cannot enter any `debug` commands from the CLI wrapper method

From the CLI wrapper method, you can enter all `configuration` and `clear` commands with the following exceptions:

- OSPF
- BGP
- RIP
- VRRP
- VIRT
- PIM IGMP
- FCoE
- IPSec
- Ikev2
- RouteMap
- OpenFlow
- VNIC

Timeout and Prompt Arguments

The CLI wrapper method supports the `Timeout` argument measured in seconds. The default `Timeout` value is 60 seconds and the range is `[5 => sys.maxint]`. If the CLI wrapper method does not detect any CLI prompts after the `Timeout`, it will send an error message and go on to the next command.

The CLI wrapper method supports custom CLI prompt argument as a Python regular expression. These custom prompts are useful when you open a `telnet` connection to non-Lenovo N/OS devices.

Following is an example of using `Timeout` and custom prompt args:

```
Analyzer = Pylib.cmd('uname -a', Timeout = 60, Prompt =
'[a-z]+@\S+:.*\$')
def CLImethod_unit_007():
    Pylib.echo('Description: Test CLI method custom prompts => Single
prompt')
    host = '9.70.2.127' # Remote host that the switch will telnet to
    user = 'root' # Remote user
    passwd = 'dbps' # Remote password
    prompts = '(\w:\s+)|(.*#>\s+)'# Remote device CLI prompts

    # Open CLI session
    mgmt.exe_login()
    mgmt.exe_telnet(hostIP=host, port='mgt') # Open telnet session to
remote host

    Pylib.cmd(user, Prompt=prompts) # Enter username on the remote host
with custom prompts
    Pylib.cmd(passwd, Prompt=prompts) # Enter password on the remote host
with custom prompts

    result = Pylib.cmd('show config', Prompt=prompts) # Issue cmd on
remote host with custom prompts
    if result.contains(reg='PortServer\s+TS\s+16\s+MEI'):# Connect to
Digi terminal server
        objTest.setPassIfNotAlreadyFailed()
    else:
        objTest.setFail()

    if result.contains(reg='ip\s+:\s+{addr}'.format(addr=host)):
        objTest.setPassIfNotAlreadyFailed()
    else:
        objTest.setFail()

    Pylib.cmd('exit') # Close remote Telnet session and return back to local telnet
    # Close CLI session
    mgmt.exe_logout()
```

Chapter 8. Limitations and Known Issues

This section contains limitations and known issues regarding Lenovo N/OS and Python scripting.

N/OS Python Scripting Limitations

Lenovo N/OS-specific Python scripts have the following limitations:

- N/OS Python API scripting is supported only the RackSwitch G8264 and RackSwitch G8264 R.
- Scripts can use Python TCP/IP socket and higher-level networking functions only on management ports.
- N/OS Python supports only for standalone-switch mode.
- N/OS Python API libraries supports only standalone-switch mode; the libraries will be disabled in stacking switch mode.
- N/OS Python API libraries support only ISCLI commands.
- N/OS Python API Libraries do not support multi-threading or multi-processing.
- You can download third-party script into the G8264 and use it, but Lenovo does *not* guarantee any third-party modules.

Note: Use third-party libraries at your own risk. Lenovo strongly recommends not using third-party libraries.

- N/OS Python network-related functions do not support data ports.
- If a script is running in the background and has invoked N/OS Python API functions, you cannot enter Lenovo CLI mode after login, because the ISCLI mode is selected as the CLI mode in the Python script.
- Only one scheduler script can be run at a time.

Note: You can run a script from the Python shell without using the Python scheduler.

- You cannot modify an active scheduler job. The workaround is to delete the scheduler job and create a new one.
- Events are put into a queue and are handled serially. There could be some latency between when the script runs and the corresponding action, which is dependent on the wait queue status.
- There is no protection to prevent scripts from creating infinite loops.
- There is finite memory that a script can allocate.
- Scripts can execute unless its log file is larger than 500kB. The total size for script log files is about 80 MB. The size of the log storage space increase constantly; the oldest files are deleted automatically. to reserve space for new ones.
- The CLI command `show mp thread` does not show the Python scheduler and jobs process information.
- To remove a persistent variable, use the `pylib.set_var(varName, ‘’)` function to overwrite the variable name. You cannot delete a variable using `pylib.set_var(varName, '')`.

Known Issues

This section describes known issues with Python scripts run on the RackSwitch G8264.

Displaying all Output from Commands

`pylib.get_output()` does not print all output if `mgmt.exe_logout()` is entered after this command. Scripts with longer output might not display all output. The complete output is written to log files.

Use `time.sleep(n)` to add sufficient time to the function to display all output.

Priority Optional Parameter Not Used when VLAG is Disabled

The priority parameter is not set in `cfg_enableVlag`.

netboot Script and Configuration File Naming Conventions

Lenovo recommends using alphanumeric characters when naming script and configuration files.

Appendix A. Getting help and technical assistance

If you need help, service, or technical assistance or just want more information about Lenovo products, you will find a wide variety of sources available from Lenovo to assist you.

Use this information to obtain additional information about Lenovo and Lenovo products, and determine what to do if you experience a problem with your Lenovo system or optional device.

Note: This section includes references to IBM web sites and information about obtaining service. IBM is Lenovo's preferred service provider for the System x, Flex System, and NeXtScale System products.

Before you call, make sure that you have taken these steps to try to solve the problem yourself.

If you believe that you require warranty service for your Lenovo product, the service technicians will be able to assist you more efficiently if you prepare before you call.

- Check all cables to make sure that they are connected.
- Check the power switches to make sure that the system and any optional devices are turned on.
- Check for updated software, firmware, and operating-system device drivers for your Lenovo product. The Lenovo Warranty terms and conditions state that you, the owner of the Lenovo product, are responsible for maintaining and updating all software and firmware for the product (unless it is covered by an additional maintenance contract). Your service technician will request that you upgrade your software and firmware if the problem has a documented solution within a software upgrade.
- If you have installed new hardware or software in your environment, check the IBM ServerProven website to make sure that the hardware and software is supported by your product.
- Go to the IBM Support portal to check for information to help you solve the problem.
- Gather the following information to provide to the service technician. This data will help the service technician quickly provide a solution to your problem and ensure that you receive the level of service for which you might have contracted.
  - Hardware and Software Maintenance agreement contract numbers, if applicable
  - Machine type number (if applicable–Lenovo 4-digit machine identifier)
  - Model number
  - Serial number
  - Current system UEFI and firmware levels
  - Other pertinent information such as error messages and logs

- Start the process of determining a solution to your problem by making the pertinent information available to the service technicians. The IBM service technicians can start working on your solution as soon as you have completed and submitted an Electronic Service Request.

You can solve many problems without outside assistance by following the troubleshooting procedures that Lenovo provides in the online help or in the Lenovo product documentation. The Lenovo product documentation also describes the diagnostic tests that you can perform. The documentation for most systems, operating systems, and programs contains troubleshooting procedures and explanations of error messages and error codes. If you suspect a software problem, see the documentation for the operating system or program.

Appendix B. Notices

Lenovo may not offer the products, services, or features discussed in this document in all countries. Consult your local Lenovo representative for information on the products and services currently available in your area.

Any reference to a Lenovo product, program, or service is not intended to state or imply that only that Lenovo product, program, or service may be used. Any functionally equivalent product, program, or service that does not infringe any Lenovo intellectual property right may be used instead. However, it is the user's responsibility to evaluate and verify the operation of any other product, program, or service.

Lenovo may have patents or pending patent applications covering subject matter described in this document. The furnishing of this document does not give you any license to these patents. You can send license inquiries, in writing, to:

Lenovo (United States), Inc.
1009 Think Place - Building One
Morrisville, NC 27560
U.S.A.

Attention: Lenovo Director of Licensing

LENOVO PROVIDES THIS PUBLICATION “AS IS” WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF NON-INFRINGEMENT, MERCHANTABILITY OR FITNESS FOR A PARTICULAR PURPOSE. Some jurisdictions do not allow disclaimer of express or implied warranties in certain transactions, therefore, this statement may not apply to you.

This information could include technical inaccuracies or typographical errors. Changes are periodically made to the information herein; these changes will be incorporated in new editions of the publication. Lenovo may make improvements and/or changes in the product(s) and/or the program(s) described in this publication at any time without notice.

The products described in this document are not intended for use in implantation or other life support applications where malfunction may result in injury or death to persons. The information contained in this document does not affect or change Lenovo product specifications or warranties.

Nothing in this document shall operate as an express or implied license or indemnity under the intellectual property rights of Lenovo or third parties. All information contained in this document was obtained in specific environments and is presented as an illustration. The result obtained in other operating environments may vary.

Lenovo may use or distribute any of the information you supply in any way it believes appropriate without incurring any obligation to you.

Any references in this publication to non-Lenovo Web sites are provided for convenience only and do not in any manner serve as an endorsement of those Web sites. The materials at those Web sites are not part of the materials for this Lenovo product, and use of those Web sites is at your own risk.

Any performance data contained herein was determined in a controlled environment. Therefore, the result obtained in other operating environments may vary significantly. Some measurements may have been made on development-level systems and there is no guarantee that these measurements will be the same on generally available systems. Furthermore, some measurements may have been estimated through extrapolation. Actual results may vary. Users of this document should verify the applicable data for their specific environment.

Trademarks

Lenovo, the Lenovo logo, Flex System, System x, NeXtScale System, and X-Architecture are trademarks of Lenovo in the United States, other countries, or both.

Intel and Intel Xeon are trademarks of Intel Corporation in the United States, other countries, or both.

Internet Explorer, Microsoft, and Windows are trademarks of the Microsoft group of companies.

Linux is a registered trademark of Linus Torvalds.

Other company, product, or service names may be trademarks or service marks of others.

Important Notes

Processor speed indicates the internal clock speed of the microprocessor; other factors also affect application performance.

CD or DVD drive speed is the variable read rate. Actual speeds vary and are often less than the possible maximum.

When referring to processor storage, real and virtual storage, or channel volume, KB stands for 1 024 bytes, MB stands for 1 048 576 bytes, and GB stands for 1 073 741 824 bytes.

When referring to hard disk drive capacity or communications volume, MB stands for 1 000 000 bytes, and GB stands for 1 000 000 000 bytes. Total user-accessible capacity can vary depending on operating environments.

Maximum internal hard disk drive capacities assume the replacement of any standard hard disk drives and population of all hard-disk-drive bays with the largest currently supported drives that are available from Lenovo.

Maximum memory might require replacement of the standard memory with an optional memory module.

Each solid-state memory cell has an intrinsic, finite number of write cycles that the cell can incur. Therefore, a solid-state device has a maximum number of write cycles that it can be subjected to, expressed as total bytes written (TBW). A device that has exceeded this limit might fail to respond to system-generated commands or might be incapable of being written to. Lenovo is not responsible for replacement of a device that has exceeded its maximum guaranteed number of program/erase cycles, as documented in the Official Published Specifications for the device.

Lenovo makes no representations or warranties with respect to non-Lenovo products. Support (if any) for the non-Lenovo products is provided by the third party, not Lenovo.

Some software might differ from its retail version (if available) and might not include user manuals or all program functionality.

Recycling Information

Lenovo encourages owners of information technology (IT) equipment to responsibly recycle their equipment when it is no longer needed. Lenovo offers a variety of programs and services to assist equipment owners in recycling their IT products. For information on recycling Lenovo products, go to:

http://www.lenovo.com/recycling

Particulate Contamination

Attention: Airborne particulates (including metal flakes or particles) and reactive gases acting alone or in combination with other environmental factors such as humidity or temperature might pose a risk to the device that is described in this document.

Risks that are posed by the presence of excessive particulate levels or concentrations of harmful gases include damage that might cause the device to malfunction or cease functioning altogether. This specification sets forth limits for particulates and gases that are intended to avoid such damage. The limits must not be viewed or used as definitive limits, because numerous other factors, such as temperature or moisture content of the air, can influence the impact of particulates or environmental corrosives and gaseous contaminant transfer. In the absence of specific limits that are set forth in this document, you must implement practices that maintain particulate and gas levels that are consistent with the protection of human health and safety. If Lenovo determines that the levels of particulates or gases in your environment have caused damage to the device, Lenovo may condition provision of repair or replacement of devices or parts on implementation of appropriate remedial measures to mitigate such environmental contamination. Implementation of such remedial measures is a customer responsibility..

Contaminant	Limits
Particulate	• The room air must be continuously filtered with 40% atmospheric dust spot efficiency (MERV 9) according to ASHRAE Standard 52.2[1]. • Air that enters a data center must be filtered to 99.97% efficiency or greater, using high-efficiency particulate air (HEPA) filters that meet MIL-STD-282. • The deliquescent relative humidity of the particulate contamination must be more than 60%[2]. • The room must be free of conductive contamination such as zinc whiskers.
Gaseous	• Copper: Class G1 as per ANSI/ISA 71.04-1985[3] • Silver: Corrosion rate of less than 300 Å in 30 days

[1] ASHRAE 52.2-2008 - *Method of Testing General Ventilation Air-Cleaning Devices for Removal Efficiency by Particle Size.* Atlanta: American Society of Heating, Refrigerating and Air-Conditioning Engineers, Inc.

[2] The deliquescent relative humidity of particulate contamination is the relative humidity at which the dust absorbs enough water to become wet and promote ionic conduction.

[3] ANSI/ISA-71.04-1985. *Environmental conditions for process measurement and control systems: Airborne contaminants.* Instrument Society of America, Research Triangle Park, North Carolina, U.S.A.

Telecommunication Regulatory Statement

This product may not be certified in your country for connection by any means whatsoever to interfaces of public telecommunications networks. Further certification may be required by law prior to making any such connection. Contact a Lenovo representative or reseller for any questions.

Electronic Emission Notices

When you attach a monitor to the equipment, you must use the designated monitor cable and any interference suppression devices that are supplied with the monitor.

Federal Communications Commission (FCC) Statement

Note: This equipment has been tested and found to comply with the limits for a Class A digital device, pursuant to Part 15 of the FCC Rules. These limits are designed to provide reasonable protection against harmful interference when the equipment is operated in a commercial environment. This equipment generates, uses, and can radiate radio frequency energy and, if not installed and used in accordance with the instruction manual, may cause harmful interference to radio communications. Operation of this equipment in a residential area is likely to cause harmful interference, in which case the user will be required to correct the interference at his own expense.

Properly shielded and grounded cables and connectors must be used to meet FCC emission limits. Lenovo is not responsible for any radio or television interference caused by using other than recommended cables and connectors or by unauthorized changes or modifications to this equipment. Unauthorized changes or modifications could void the user's authority to operate the equipment.

This device complies with Part 15 of the FCC Rules. Operation is subject to the following two conditions: (1) this device may not cause harmful interference, and (2) this device must accept any interference received, including interference that might cause undesired operation.

Industry Canada Class A Emission Compliance Statement

This Class A digital apparatus complies with Canadian ICES-003.

Avis de Conformité à la Réglementation d'Industrie Canada

Cet appareil numérique de la classe A est conforme à la norme NMB-003 du Canada.

Australia and New Zealand Class A Statement

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

European Union EMC Directive Conformance Statement

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

This product has been tested and found to comply with the limits for Class A Information Technology Equipment according to European Standard EN 55022. The limits for Class A equipment were derived for commercial and industrial environments to provide reasonable protection against interference with licensed communication equipment.

Lenovo, Einsteinova 21, 851 01 Bratislava, Slovakia

Germany Class A Statement

Zulassungsbescheinigung laut dem Deutschen Gesetz über die elektromagnetische Verträglichkeit von Betriebsmitteln, EMVG vom 20. Juli 2007 (früher Gesetz über die elektromagnetische Verträglichkeit von Geräten), bzw. der EMV EG Richtlinie 2004/108/EC (früher 89/336/EWG), für Geräte der Klasse A.

Dieses Gerät ist berechtigt, in übereinstimmung mit dem Deutschen EMVG das EG-Konformitätszeichen - CE - zu führen. Verantwortlich für die Konformitätserklärung nach Paragraf 5 des EMVG ist die Lenovo (Deutschland) GmbH, Gropiusplatz 10, D-70563 Stuttgart.

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

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

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

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

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

Deutschland:

Einhaltung des Gesetzes über die elektromagnetische Verträglichkeit von Betriebsmittein
Dieses Produkt entspricht dem “Gesetz über die elektromagnetische Verträglichkeit von Betriebsmitteln” EMVG (früher “Gesetz über die elektromagnetische Verträglichkeit von Geräten"). Dies ist die Umsetzung der EU-Richtlinie 2004/108/EG (früher 89/336/EWG) in der Bundesrepublik Deutschland.

Zulassungsbescheinigung laut dem Deutschen Gesetz über die elektromagnetische Verträglichkeit von Betriebsmitteln, EMVG vom 20. Juli 2007 (früher Gesetz über die elektromagnetische Verträglichkeit von Geräten), bzw. der EMV EG Richtlinie 2004/108/EC (früher 89/336/EWG), für Geräte der Klasse A.

Dieses Gerät ist berechtigt, in übereinstimmung mit dem Deutschen EMVG das EG-Konformitätszeichen - CE - zu führen. Verantwortlich für die Konformitätserklärung nach Paragraf 5 des EMVG ist die Lenovo (Deutschland) GmbH, Gropiusplatz 10, D-70563 Stuttgart.

Informationen in Hinsicht EMVG Paragraf 4 Abs. (1) 4:
Das Gerät erfüllt die Schutzanforderungen nach EN 55024 und EN 55022 Klasse A.

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

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

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

Japan VCCI Class A Statement

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

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

Japan Electronics and Information Technology Industries Association (JEITA) Statement

高調波ガイドライン適合品

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

高調波ガイドライン準用品

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

Korea Communications Commission (KCC) Statement

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

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

Russia Electromagnetic Interference (EMI) Class A statement

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

People’s Republic of China Class A electronic emission statement

中华人民共和国“A类”警告声明

声 明

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

Taiwan Class A compliance statement

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