Python API

About the Python API

Python is an easy to learn and powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's syntax and dynamic typing, together with its interpreted nature, make it an ideal language for scripting and rapid development for many applications. The Python website www.python.org contains distributions of and pointers to many free third-party Python modules, programs and tools, and additional documentation.

The Python interpreter is available in the Cisco MDS NX-OS command-line interface (CLI) along with standard and Cisco Python modules. Both interactive and non-interactive (script) modes are supported. This gives programmatic control of MDS devices to perform repetitive tasks. Some examples of where this can be leveraged are PowerOn Auto Provisioning (POAP) scripts and Embedded Event Manager (EEM) actions. It is also used in the Overlay CLI of the SAN Analytics feature to format analytics data.

Supported Versions

Table 1: Python Version History for MDS Platforms, on page 1 shows the milestones of Python on Cisco MDS switches. It shows when changes in the supported and default versions occurred for each platform.

Table 1: Python Version History for MDS Platforms

Release	Cisco NX-OS MDS Platform							
	9132T	**9148S**	**9148T**	**9220i**	**9250i**	**9396S**	**9396T**	**9700**
6.2(29)	—	—	—	—	—	—	—	2.7
8.3(1)	2.7.8	—	2.7.8	—	—	—	2.7.8	2.7.8
8.4(2)	2.7.8* 3.7.3	—	2.7.8* 3.7.3	—	—	—	2.7.8* 3.7.3	2.7.8* 3.7.3

Release	Cisco NX-OS MDS Platform							
	9132T	9148S	9148T	9220i	9250i	9396S	9396T	9700
8.5(1)	2.7.8* 3.7.3	—	2.7.8* 3.7.3	2.7.8* 3.7.3	—	—	2.7.8* 3.7.3	2.7.8* 3.7.3
9.2(1)	2.7.8* 3.7.3	—	2.7.8* 3.7.3	2.7.8* 3.7.3	—	—	2.7.8* 3.7.3	2.7.8* 3.7.3
9.2(2)	3.7.3*	—	3.7.3*	3.7.3*	—	—	3.7.3*	3.7.3*

* indicated the default Python version.

Note Python API is not supported on Cisco MDS 16 Gbps Fabric switches such as Cisco MDS 9148S, Cisco MDS 9250i, and Cisco MDS 9396S.

Using Python

This section describes how to write and execute Python scripts.

Cisco Python Package

Cisco MDS NX-OS provides a Cisco Python package that enables access to many core network device modules, such as interfaces, VLANs, and routes. You can display the details of the Cisco Python package by entering the **help()** command. To obtain additional information about the classes and methods in a module, you can run the help command for a specific module. For example, **help**(*cisco.interface*) displays the properties of the cisco.interface module.

The following is an example of how to display information about the Cisco python package:

```
>>> import cisco
>>> help(cisco)
Help on package cisco:

NAME
    cisco

DESCRIPTION
    ################################################################################
    #
    #       File:   cli.py
    #       Name:
    #
    #       Description:
    #
    # Copyright (c) 2015-2017, 2019-2020 by cisco Systems, Inc.
    # All rights reserved.
    #
    ################################################################################

PACKAGE CONTENTS
```

```
    acl
    bgp
    buffer_depth_monitor
    check_port_discards
    cisco_secret
    feature
    history
    interface
    ipaddress
    key
    line_parser
    mac_address_table
    nxcli
    ospf
    routemap
    routes
    section_parser
    ssh
    system
    tacacs
    transfer
    vlan
    vrf

CLASSES
    builtins.object
```

Using the CLI Command APIs

The Python programming language uses three APIs that can execute CLI commands. The APIs are available from the Python CLI module.

These APIs are listed in the following table. You need to enable the APIs with the **from cli import** * command. The arguments for these APIs are strings of CLI commands. To execute a CLI command through the Python interpreter, you enter the CLI command as an argument string of one of the following APIs:

Table 2: CLI Command APIs

API	Description
cli() Example: `string = cli ("cli-command")`	Returns the raw output of CLI commands, including control/special characters. **Note** The interactive Python interpreter prints control/special characters 'escaped'. A carriage return is printed as '\n' and gives results that might be difficult to read. The **clip()** API gives results that are more readable.

API	Description
clid() Example: `json_string = clid ("cli-command")`	For CLI commands that support XML, this API returns JSON output. **Note** An exception is thrown when XML is not used. This API can be useful when searching the output of show commands.

clip()
Example:
`clip ("cli-command")`
	Prints the output of the CLI command directly to stdout and returns nothing to Python.
	Note
	`clip ("cli-command")`
	is equivalent to
	`r=cli("cli-command")`
	`print r`

When two or more commands are run individually, the state is not persistent from one command to subsequent commands.

In the following example, the second command fails because the state from the first command does not persist for the second command:

```
>>> cli("conf t")
>>> cli("interface fc4/1")
```

When two or more commands are run together, the state is persistent from one command to subsequent commands.

In the following example, the second command is successful because the state persists for the second and third commands:

```
>>> cli("conf t ; interface fc4/1 ; shut")
```

Note Commands are separated with " ; " as shown in the example. (The ; must be surrounded with single blank characters.)

Invoking the Python Interpreter from the CLI

The following example shows how to invoke Python from the CLI:

Note

- The Python interpreter is designated with the ">>>" or "…" prompt.
- In Cisco MDS NX-OS Release 7.3(x) and later releases, the Python interpreter can be invoked only in Privileged EXEC mode on Cisco MDS 9700 Series Switches.
- From Cisco MDS NX-OS Release 9.2(2), the python command runs Python 3.0.

The following example shows how to invoke Python2 .7.5 from the CLI:

```
switch# python

Python 2.7.5 (default, Jun  3 2016, 03:57:06)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> from cli import *
>>> cli("show clock")
'Time source is NTP\n06:32:20.023 UTC Tue Jan 31 2017\n'
>>> exit()
```

The following example shows how to invoke Python3 from the CLI:

```
switch# python3

Python 3.7.3 (default, Aug 26 2019, 23:20:10)
[GCC 4.6.3] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>> from cli import *
>>> cli("show clock")
'Time source is NTP\n07:46:50.923 UTC Tue Apr 28 2020\n'
```

Display Formats

The following examples show various display formats using the Python APIs:

Example 1:

```
>>> from cli import *
>>> cli("conf ; interface fc1/1")
''
clip('where detail')
  mode:
  username:            admin

>>>
```

Example 2:

```
>>> from cli import *
>>> cli("conf ; interface fc1/1")
''
>>> cli('where detail')
'  mode:                \n  username:            admin\n'
>>>
```

Example 3:

```
>>> from cli import *
>>> cli("conf ; interface fc1/1")
''
>>> r = cli('where detail') ; print(r)
  mode:
  username:            admin

>>>
```

Example 4:

```
>>> from cli import *
>>> import json
```

```
>>> out=json.loads(clid('show version'))
>>> for k in out.keys():
...    print("%30s = %s" % (k, out[k]))
...
                    header_str = Cisco Nexus Operating System (NX-OS) Software
TAC support: http://www.cisco.com/tac
Documents: http://www.cisco.com/en/US/products/ps9372/tsd_products_support_series_home.html
Copyright (c) 2002-2022, Cisco Systems, Inc. All rights reserved.
The copyrights to certain works contained in this software are
owned by other third parties and used and distributed under
license. Certain components of this software are licensed under
the GNU General Public License (GPL) version 2.0 or the GNU
Lesser General Public License (LGPL) Version 2.1. A copy of each
such license is available at
http://www.opensource.org/licenses/gpl-2.0.php and
http://www.opensource.org/licenses/lgpl-2.1.php
                  bios_ver_str = 2.12.0
              kickstart_ver_str = 9.2(2)
                    sys_ver_str = 9.2(2)
                 bios_cmpl_time = 05/26/2021
                 kick_file_name = bootflash:///m9700-sf4ek9-kickstart-mz.9.2.2.bin
                 kick_cmpl_time = 1/1/2022 12:00:00
                    kick_tmstmp = 01/20/2022 22:42:00
                 isan_file_name = bootflash:///m9700-sf4ek9-mz.9.2.2.bin
                 isan_cmpl_time = 1/1/2022 12:00:00
                    isan_tmstmp = 01/21/2022 00:12:45
                     chassis_id = MDS 9706 (6 Slot) Chassis
                      module_id = Supervisor Module-4
                       cpu_name = Intel(R) Xeon(R) CPU D-1548 @ 2.00GHz
                         memory = 14146356
                       mem_type = kB
                   proc_board_id = JAE22320AXJ
                       host_name = sw184-9706
                 bootflash_size = 3932160
                     slot0_size = 0
                 kern_uptm_days = 5
                  kern_uptm_hrs = 21
                 kern_uptm_mins = 39
                 kern_uptm_secs = 27
                       rr_usecs = 699796
                       rr_ctime = Tue Jan 25 10:40:02 2022
                      rr_reason = Reset Requested by CLI command reload
                     rr_sys_ver = 9.2(2)
                     rr_service = None
                   manufacturer = Cisco Systems, Inc.
>>>
```

Non-interactive Python

A Python script can run in non-interactive mode by providing the Python script name as an argument to the Python CLI command. Python scripts must be placed under the bootflash or volatile scheme. A maximum of 32 command line arguments for the Python script are allowed with the Python CLI command.

The Cisco MDS 9000 Series device also supports the source CLI command for running Python scripts. The `bootflash:scripts` directory is the default script directory for the source CLI command.

Note To run Python scripts using the Python3 interpreter, ensure the first line of the script contains python3 string such as, #!/isan/bin/python3 OR #!/usr/bin/env python3.

The following example shows a script and how to run it:

```
switch# show file bootflash:flashCheck.py
#!/bin/env python
import re
import json
import cli
import syslog

threshold = 60
ignore_paths = ['bootflash']

allmodules = json.loads(cli.clid("show module"))['TABLE_modinfo']['ROW_modinfo']
if type(allmodules) is dict:
        allmodules = [allmodules]
for eachmodule in allmodules:
        mod = eachmodule['mod']
       modtype = eachmodule['modtype']
        cmd = "slot " + str(mod) + " show system internal flash"
        if 'Supervisor' in modtype:
                s = "Supervisor(Module " + str(mod) + ")"
                regex_to_match =
r'(?P<mnton>\S+)\s+(?P<onekblks>\d+)\s+(?P<used>\d+)\s+(?P<avail>\d+)\s+(?P<useper>\d+)\s+(?P<fs>\S+)'

        elif 'Sup' in modtype:
            cmd = " show system internal flash"
            s = "Sup and LC - Module 1"
            regex_to_match =
r'(?P<mnton>\S+)\s+(?P<onekblks>\d+)\s+(?P<used>\d+)\s+(?P<avail>\d+)\s+(?P<useper>\d+)\s+(?P<fs>\S+)'

        else:
                s = "Module " + str(mod)
                regex_to_match =
r'(?P<fs>\S+)\s+(?P<onekblks>\d+)\s+(?P<used>\d+)\s+(?P<avail>\d+)\s+(?P<useper>\d+)%\s+(?P<mnton>\S+)'


        out = cli.cli(cmd)
        alllines = out.splitlines()
        for eachline in alllines:
                match = re.search(regex_to_match,eachline)
                if match:
                        grps = match.groupdict()
                        #print(grps)
                        sysstr = "Flash usage exceeds threshold({}%) on {} - Filesystem:
{}, Mounted-On: {}, Usage:
{}%".format(str(threshold),s,grps['fs'],grps['mnton'],grps['useper'])
                        for each_ignore_path in ignore_paths:
                                if each_ignore_path in grps['mnton']:
                                        break
                                else:
                                        if int(grps['useper']) > threshold:
                                                syslog.syslog(2,sysstr)
```

Running Scripts with Embedded Event Manager

On Cisco MDS 9000 Series devices, embedded event manager (EEM) policies support Python scripts.

The following example shows how to run a Python script as an EEM action:

- An EEM applet can include a Python script with an action command.

```
switch# show running-config eem

!Command: show running-config eem
!Time: Sun May  1 14:40:07 2011

version 6.1(2)I2(1)
event manager applet a1
  event cli match "show clock"
  action 1 cli python bootflash:pydate.py
  action 2 event-default
```

- You can search for the action triggered by the event in the log file by running the **show file** *logflash*:*event_archive_1* command.

```
switch# show file logflash:event_archive_1 | last 33

eem_event_time:05/01/2011,19:40:28 event_type:cli event_id:8 slot:active(1)
vdc:1 severity:minor applets:a1
eem_param_info:command = "exshow clock"
Starting with policy a1
Python

2011-05-01 19:40:28.644891
Executing the following commands succeeded:
          python bootflash:pydate.py

PC_VSH_CMD_TLV(7679) with q
```

Cisco MDS NX-OS Security with Python

Cisco MDS NX-OS resources are protected by the Cisco MDS NX-OS Sandbox layer of software and by the CLI role-based access control (RBAC).

All users associated with a Cisco MDS NX-OS network-admin or dev-ops role are privileged users. Users who are granted access to Python with a custom role are regarded as non-privileged users. Non-privileged users have a limited access to Cisco MDS NX-OS resources, such as file system, guest shell, and Bash commands. Privileged users have greater access to all the resources of Cisco MDS NX-OS.

Examples of Security and User Authority

The following example shows how a privileged user runs commands:

```
switch# python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.system('whoami')
admin
0
>>> f=open('/tmp/test','w')
>>> f.write('hello from python')
>>> f.close()
>>> r=open('/tmp/test','r')
>>> print r.read()
hello from python
>>> r.close()
```

The following example shows a non-privileged user being denied access:

```
switch# python
Python 2.7.5 (default, Oct  8 2013, 23:59:43)
[GCC 4.6.3] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> import os
>>> os.system('whoami')
system(whoami): rejected!
-1
>>> f=open('/tmp/test','r')
Permission denied. Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
IOError: [Errno 13] Permission denied: '/tmp/test'
>>>
```

RBAC controls CLI access based on the login user privileges. A login user's identity is given to Python that is invoked from the CLI shell or from Bash. Python passes the login user's identity to any subprocess that is invoked from Python.

The following is an example for a privileged user:

```
>>> from cli import *
>>> cli('show clock')
'11:28:53.845 AM UTC Sun May 08 2011\n'
>>> cli('configure terminal ; vrf context myvrf')
''
>>> clip('show running-config l3vm')

!Command: show running-config l3vm
!Time: Sun May  8 11:29:40 2011

version 6.1(2)I2(1)

interface Ethernet1/48
  vrf member blue

interface mgmt0
  vrf member management
vrf context blue
vrf context management
vrf context myvrf
```

The following is an example for a non-privileged user:

```
>>> from cli import *
>>> cli('show clock')
'11:18:47.482 AM UTC Sun May 08 2011\n'
>>> cli('configure terminal ; vrf context myvrf2')
Traceback (most recent call last):
  File "<stdin>", line 1, in <module>
  File "/isan/python/scripts/cli.py", line 20, in cli
    raise cmd_exec_error(msg)
errors.cmd_exec_error: '% Permission denied for the role\n\nCmd exec error.\n'
```

The following example shows an RBAC configuration:

```
switch# show user-account
user:admin
        this user account has no expiry date
        roles:network-admin
user:pyuser
        this user account has no expiry date
        roles:network-operator python-role
```

```
switch# show role name python-role
```

Example of Running Script with Scheduler

The following example shows a Python script that is running the script with the scheduler feature:

```
#!/bin/env python
from cli import *
from nxos import *
import os

switchname = cli("show switchname")
try:
    user = os.environ['USER']
except:
    user = "No user"
    pass

msg = user + " ran " + __file__ + " on : " + switchname
print msg
py_syslog(1, msg)
# Save this script in bootflash:///scripts
```

```
switch# conf t
Enter configuration commands, one per line.  End with CNTL/Z.
switch(config)# feature scheduler
switch(config)# scheduler job name testplan
switch(config-job)# python bootflash:///scripts/testplan.py
switch(config-job)# exit
switch(config)# scheduler schedule name testplan
switch(config-schedule)# job name testplan
switch(config-schedule)# time start now repeat 0:0:4
Schedule starts from Mon Mar 14 16:40:03 2011
switch(config-schedule)# end
switch# term mon
2011 Mar 14 16:38:03 switch %VSHD-5-VSHD_SYSLOG_CONFIG_I: Configured from vty by admin on
10.19.68.246@pts/2
switch# show scheduler schedule
Schedule Name       : testplan
------------------------------
User Name           : admin
Schedule Type       : Run every 0 Days 0 Hrs 4 Mins
Start Time          : Mon Mar 14 16:40:03 2011
Last Execution Time : Yet to be executed
----------------------------------------------
     Job Name            Last Execution Status
----------------------------------------------
    testplan                              -NA-
==============================================================================
switch#
switch# 2011 Mar 14 16:40:04 switch  %USER-1-SYSTEM_MSG: No user ran
/bootflash/scripts/testplan.py on : switch   - nxpython
2011 Mar 14 16:44:04 switch last message repeated 1 time
switch#
```