Python API

Python API

Python is a programming language that has high-level data structures and a simple 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 application development in many areas on most platforms.

The Python interpreter and the standard library are freely available in source or binary form for all major platforms from the Python website:

http://www.python.org/

The same site also contains distributions of and pointers to many free third-party Python modules, programs and tools, and additional documentation.

The Cisco Nexus Series devices support Python version 2.7.5 in both interactive and noninteractive (script) modes is available in the Guest Shell.

The Python scripting capability provides programmatic access to the device's CLI to perform various tasks and PowerOn Auto Provisioning (POAP) or Embedded Event Manager (EEM) actions. Python can also be accessed from the Bash shell.

The Python interpreter is available in the Cisco NX-OS software.

For information about using Python with Cisco Nexus devices, see the Cisco Nexus 9000 Series Python SDK User Guide and API Reference at this URL: http://opennxos.cisco.com/public/api/python/.

Using Python

This section describes how to write and execute Python scripts.

Python Package for Cisco

Cisco NX-OS provides a Python package for Cisco package that enables access to many core network device modules, such as interfaces, VLANs, VRFs, ACLs, and routes. You can display the details of the Python package for Cisco by entering the **help()** command. To obtain additional information about the classes and methods in a module, 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 Python package for Cisco:

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

NAME
    cisco

FILE
    /isan/python/scripts/cisco/__init__.py

PACKAGE CONTENTS
    acl
    bgp
    cisco_secret
    cisco_socket
    feature
    interface
    key
    line_parser
    md5sum
    nxcli
    ospf
    routemap
    routes
    section_parser
    ssh
    system
    tacacs
    vrf

CLASSES
     __builtin__.object
        cisco.cisco_secret.CiscoSecret
        cisco.interface.Interface
        cisco.key.Key
```

Using the CLI Command APIs

The Python programming language uses three APIs that can execute CLI commands. These 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, enter the CLI command as an argument string of one of the following APIs:

Table 1: CLI Command APIs

API	Description
cli() Example: `string = cli ("cli-command")`	Returns the raw output of CLI commands, including control and 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")`	Returns JSON output for **cli-command** if XML support exists for the command; otherwise, an exception is thrown. **Note** 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`

To get output using Cisco/CLI library and read output from slot, enclose the complete CLI in double quotes and the argument in single quotes.

```
>>> cli("slot 6 quoted 'show hardware internal dev-port-map'")
```

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 eth4/1")
```

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 eth4/1 ; shut")
```

Note Commands are separated with a semicolon (;), as shown in the example. The semicolon 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 that the Python interpreter is designated with the >>> or … prompt.

```
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.
>>> from cli import *
>>> import json
>>> cli('configure terminal ; interface loopback 5 ; no shut')
''
```

```
>>> intflist=json.loads(clid('show interface brief'))
>>> i=0
>>> while i < len(intflist['TABLE_interface']['ROW_interface']):
...   intf=intflist['TABLE_interface']['ROW_interface'][i]
...   i=i+1
...   if intf['state'] == 'up':
...     print intf['interface']
...
mgmt0
Ethernet2/7
Ethernet4/7
loopback0
loopback5
>>>
```

Display Formats

The following examples show various display formats using Python APIs:

Example 1

```
>>> from cli import *
>>> cli("conf ; interface loopback 1")
''
>>> clip('where detail')
mode:
username: admin
vdc: switch
routing-context vrf: default
```

Example 2

```
>>> from cli import *
>>> cli("conf ; interface loopback 1")
''
>>> cli('where detail')
' mode: \n username: admin\n vdc:
switch\n routing-context vrf: default\n'
>>>
```

Example 3

```
>>> from cli import *
>>> cli("conf ; interface loopback 1")
''
>>> r = cli('where detail') ; print r
mode:
username: admin
vdc: EOR-1
routing-context vrf: default
>>>
```

Example 4

```
>>> from cli import *
>>> import json
>>> out=json.loads(clid('show version'))
>>> for k in out.keys():
...   print "%30s = %s" % (k, out[k])
...
                kern_uptm_secs = 6
                kick_file_name = bootflash:///n9000-dk9.6.1.2.I1.1.bin
```

```
                    rr_service = None
                     module_id = Supervisor Module
                   kick_tmstmp = 10/21/2013 00:06:10
                bios_cmpl_time = 08/17/2013
                bootflash_size = 20971520
             kickstart_ver_str = 6.1(2)I1(2) [build 6.1(2)I1(2)] [gdb]
                kick_cmpl_time = 10/20/2013 4:00:00
                    chassis_id = Nexus9000 C9508 (8 Slot) Chassis
                 proc_board_id = SAL171211LX
                        memory = 16077872
                  manufacturer = Cisco Systems, Inc.
                kern_uptm_mins = 26
                  bios_ver_str = 06.14
                      cpu_name = Intel(R) Xeon(R) CPU E5-2403
                 kern_uptm_hrs = 2
                      rr_usecs = 816550
                   rr_sys_ver = None
                    rr_reason = Reset Requested by CLI command reload
                     rr_ctime = Mon Oct 21 00:10:24 2013
                   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) 2002-2013, Cisco Systems, Inc. All rights reserved.
The copyrights to certain works contained herein are owned by other third parties and are
used and distributed under license.
Some parts of this software are covered under the GNU Public License. A copy of the license
 is available at
http://www.gnu.org/licenses/gpl.html.
                    host_name = switch
                     mem_type = kB
                kern_uptm_days = 0
>>>
```

Python script in Noninteractive Mode

A Python script can run in noninteractive mode when the Python script name is provided 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.

Cisco Nexus 7000 Series devices also support the source CLI command for running Python scripts. The `bootflash:scripts` directory is the default script directory for the source CLI command.

You can launch and run a python process in the backgorund. However, when the associated SSH terminal is terminated, the child processes started from the terminal will also be terminated. Beginning from release 8.0(1), if you want to run the script after terminal exit, ignore the SIGHUP signal in the running script. e.g. signal.signal(signal.SIGHUP, signal.SIG_IGN)

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

```
switch# show file bootflash:deltaCounters.py
#!/isan/bin/python

from cli import *
import sys, time

ifName = sys.argv[1]
delay = float(sys.argv[2])
count = int(sys.argv[3])
cmd = 'show interface ' + ifName + ' counters'

out = json.loads(clid(cmd))
rxuc = int(out['TABLE_rx_counters']['ROW_rx_counters'][0]['eth_inucast'])
rxmc = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inmcast'])
```

```
rxbc = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inbcast'])
txuc = int(out['TABLE_tx_counters']['ROW_tx_counters'][0]['eth_outucast'])
txmc = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outmcast'])
txbc = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outbcast'])
print 'row rx_ucast rx_mcast rx_bcast tx_ucast tx_mcast tx_bcast'
print '==============================================================='
print '    %8d %8d %8d %8d %8d %8d' % (rxuc, rxmc, rxbc, txuc, txmc, txbc)
print '==============================================================='

i = 0
while (i < count):
  time.sleep(delay)
  out = json.loads(clid(cmd))
  rxucNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][0]['eth_inucast'])
  rxmcNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inmcast'])
  rxbcNew = int(out['TABLE_rx_counters']['ROW_rx_counters'][1]['eth_inbcast'])
  txucNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][0]['eth_outucast'])
  txmcNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outmcast'])
  txbcNew = int(out['TABLE_tx_counters']['ROW_tx_counters'][1]['eth_outbcast'])
  i += 1
  print '%-3d %8d %8d %8d %8d %8d %8d' % \
    (i, rxucNew - rxuc, rxmcNew - rxmc, rxbcNew - rxbc, txucNew - txuc, txmcNew - txmc,
txbcNew - txbc)
```

```
switch# python bootflash:deltaCounters.py Ethernet1/1 1 5
row rx_ucast rx_mcast rx_bcast tx_ucast tx_mcast tx_bcast
===============================================================
           0      791        1        0   212739        0
===============================================================
1          0        0        0        0       26        0
2          0        0        0        0       27        0
3          0        1        0        0       54        0
4          0        1        0        0       55        0
5          0        1        0        0       81        0
switch#
```

The following example shows how a **source** command specifies command-line arguments. In this example, *policy-map* is an argument to the **cgrep python** script. The example also shows that a source command can follow after the pipe operator (|).

```
switch# show running-config | source sys/cgrep policy-map

policy-map type network-qos nw-pfc
policy-map type network-qos no-drop-2
policy-map type network-qos wred-policy
policy-map type network-qos pause-policy
policy-map type qos foo
policy-map type qos classify
policy-map type qos cos-based
policy-map type qos no-drop-2
policy-map type qos pfc-tor-port
```

Running Scripts with the Embedded Event Manager

On Cisco Nexus 7000 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 an 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
```

Python Integration with Cisco NX-OS Network Interfaces

On Cisco Nexus 7000 Series devices, Python is integrated with the underlying Cisco NX-OS network interfaces. You can switch from one virtual routing context to another by setting up a context through the cisco.vrf.set_global_vrf() API.

The following example shows how to retrieve an HTML document over the management interface of a device. You can also establish a connection to an external entity over the inband interface by switching to a desired virtual routing context.

```
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 urllib2
>>> from cisco.vrf import *
>>> set_global_vrf('management')
>>> page=urllib2.urlopen('http://172.23.40.211:8000/welcome.html')
>>> print page.read()
Hello Cisco Nexus 9000

>>>
>>> import cisco
>>> help(cisco.vrf.set_global_vrf)
Help on function set global vrf in module cisco.vrf:

set global vrf(vrf)
    Sets the global vrf. Any new sockets that are created (using socket.socket)
    will automatically get set to this vrf (including sockets used by other
    python libraries).

    Arguments:
        vrf: VRF name (string) or the VRF ID (int).

    Returns: Nothing

>>>
```

Cisco NX-OS Security with Python

Cisco NX-OS resources are protected by the Cisco NX-OS sandbox layer of software and by CLI role-based access control (RBAC). Cisco NX-OS allows access to all resources, including file system, guest shell, and Bash commands for privileged users, which are limited to the network-admin and dev-ops roles.

For privileged users the sandbox is disabled. Python access for all other roles including custom are considered non-privileged and are contained by the sandbox. For more information on guidelines and limitations of VDC user roles, see Information About VDCs section in *Cisco Nexus 7000 Series Virtual Device Context Configuration Guide*.

- Examples of Security and User Authority, on page 8
- Example of Running Script with Scheduler, on page 9

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# configure terminal
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#
```