Send document comments to nexus3k-docfeedback@cisco.com.

Cisco Nexus 3000 Series NX-OS Python API Reference Guide, Release 5.0(3)U3(2)

May 01, 2012

Americas Headquarters
Cisco Systems, Inc.
170 West Tasman Drive
San Jose, CA 95134-1706
USA
http://www.cisco.com
Tel: 408 526-4000
800 553-NETS (6387)
Fax: 408 527-0883

Text Part Number: OL-26601-02

Cisco Nexus 3000 Series NX-OS Python API Reference Guide, Release 5.0(3)U3(2)

CONTENTS

Preface

This document lists and describes the Python Application Programming Interface (API) for Cisco Nexus 3000 series switches.

This chapter includes the following sections:

Audience

This publication is for system administrators who are responsible for installing and managing Cisco Nexus 3000 series switches.

Document Conventions

Command descriptions use these conventions:

Conventions	Description
boldface font	Commands and keywords are in boldface.
italic font	Arguments for which you supply values are in italics.
{ }	Elements in braces are required choices.
[ ]	Elements in square brackets are optional.
x | y | z	Alternative, mutually exclusive elements are separated by vertical bars.
string	A nonquoted set of characters. Do not use quotation marks around the string or the string will include the quotation marks.

Screen examples use these conventions:

`screen font`	Terminal sessions and information the device displays are in screen font.
`boldface screen font`	Information you must enter is in boldface screen font.

italic screen font	Arguments for which you supply values are in italic screen font.
< >	Nonprinting characters, such as passwords, are in angle brackets.
[ ]	Default responses to system prompts are in square brackets.
!, #	An exclamation point (!) or a pound sign (#) at the beginning of a line of code indicates a comment line.

This document uses the following conventions:

Note Means reader *take note*. Notes contain helpful suggestions or references to material not covered in the manual.

Caution Means *reader be careful*. In this situation, you might do something that could result in equipment damage or loss of data.

Related Documentation

Documentation for the Cisco Nexus 3000 Series Switch is available at the following URL:

http://www.cisco.com/en/US/products/ps11541/tsd_products_support_series_home.html

The documentation set is divided into the following categories:

Release Notes

The release notes are available at the follwing URL:

http://www.cisco.com/en/US/products/ps11541/prod_release_notes_list.html

Installation and Upgrade Guides

The installation and upgrade guides are available at the following URL:

http://www.cisco.com/en/US/products/ps11541/prod_installation_guides_list.html

Command References

The command references are available at the following URL:

http://www.cisco.com/en/US/products/ps11541/prod_command_reference_list.html

Technical References

The technical references are available at the following URL:

http://www.cisco.com/en/US/products/ps11541/prod_technical_reference_list.html

Configuration Guides

The configuration guides are available at the following URL:

http://www.cisco.com/en/US/products/ps11541/products_installation_and_configuration_guides_list.html

Error and System Messages

The system message reference guide is available at the following URL:

http://www.cisco.com/en/US/products/ps11541/products_system_message_guides_list.html

Obtaining Documentation and Submitting a Service Request

For information on obtaining documentation, submitting a service request, and gathering additional information, see the monthly *What's New in Cisco Product Documentation*, which also lists all new and revised Cisco technical documentation, at:

http://www.cisco.com/en/US/docs/general/whatsnew/whatsnew.html

Subscribe to the *What's New in Cisco Product Documentation* as a Really Simple Syndication (RSS) feed and set content to be delivered directly to your desktop using a reader application. The RSS feeds are a free service and Cisco currently supports RSS version 2.0.

Send document comments to nexus3k-docfeedback@cisco.com.

CHAPTER 1

Overview

This chapter provides the overview and installation information needed to use the Python Application Programming Interface (API) support on Cisco Nexus 3000 Series switches.

This chapter includes the following sections:

Information About the Python API

Python is an easy to learn, powerful programming language. It has efficient high-level data structures and a simple but effective approach to object-oriented programming. Python's elegant 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 extensive 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 3000 series switches support all the features available in Python v2.7.2.

The Python scripting capability on the Cisco Nexus 3000 series switches enables you to perform the following tasks:

- Run a script to verify configuration on switch bootup.
- Back up a configuration.
- Proactive congestion management by monitoring and responding to buffer utilization characteristics.
- Integration with the Power-On Auto Provisioning or EEM modules.
- Ability to perform a job at a specific time interval (such as Port Auto Description).
- Programmatic access to the switch command line interface (CLI) to perform various tasks.

Installing Python

The Python interpreter is available by default on the Cisco NX-OS software.You can invoke Python by entering the **python** command, and write scripts to access Cisco NX-OS APIs by importing the cisco.py module using the **import cisco.py** command.

Installing Third Party Pure Python Packages

You can install the third party pure Python package by copying mypkg.tgz on your server. Perform the following steps to extract and install the third party package:

- Secure copy the tar file by executing the **copy scp://user@server/path/to/mypkg.tgz bootflash:mypkg.tgz vrf management** command
- Untar the mypkg.tgz file by using the **tar extract bootflash:mypkg.tgz** command.
- Move the extracted file to bootflash by using the **move bootflash:mypkg-1.2/* bootflash:** command.
- You can install the package by using the **python setup.py install** command.
- Remove the copied file from bootflash.
- You can use the third party package in scripts or in the Python shell.

```
switch# python
>>> import mypkg
```

Note You will be able to install the third party packages using the **easy_install** command, in the future releases.

Using Python

Note To comply with the PEP8 coding guidelines, the names of a few Cisco Python APIs have been changed. You can use the following as example to convert your existing scripts to new APIs:

```
% cat sed.in
s/showArpTable/show_arp_table/g
s/showVshRoutes/show_vsh_routes/g
s/showHwRoutes/show_hw_routes/g
s/verifyRoutes/verify_routes/g
s/verifyArpTable/verify_arp_table/g
s/getTotalInstantUsage/get_total_instant_usage/g
s/GetRemainingInstantUsage/get_remaining_instant_usage/g
s/getMaxCellUsage/get_max_cell_usage/g
s/getSwitchCellCount/get_switch_cell_count/g

% sed -f sed.in < python_script_with_old_API_names.py > python_script_with_old_API_names.py
```

This section describes how to write and execute Python scripts by passing parameters and includes the following topics:

Entering Python Shell

You can enter the Python shell by using the **python** command without any parameters.

```
switch# python
Python 2.7.2 (default, Oct 11 2011, 13:55:49)
[GCC 3.4.3 (MontaVista 3.4.3-25.0.143.0800417 2008-02-22)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
Loaded cisco NX-OS lib!
>>> print 'helo world!'
helo world!
>>>exit()
switch#
```

Executing Scripts

You can execute a Python script by using the **python** <*filename*> command.

```
switch# python test.py
['/bootflash/test.py']
doing 0/1
doing 0/2
doing 1/2
switch#
```

Passing Parameters to the Script

You can execute a Python script by using the **python** <*filename*> [*arg1, arg2, arg3,.....*] command.

```
switch# python test.py foo bar 1 2
['/bootflash/test.py', 'foo', 'bar', '1', '2']
doing 0/1
doing 0/2
doing 1/2
switch#
```

Embedded Event Manager Support

Embedded Event Manager (EEM) supports invocation of Python scripts based on events. Syslog for events can be passed to a Python script by using the **variable $** command.

The following example shows EEM invoking a python script for an IF_DOWN event.

EEM configuration:

```
switch(config)# event manager applet if-mon
switch(config-applet)# event syslog pattern "*IF_DOWN.*"
Configuration accepted successfully
switch(config-applet)# action 1.0 cli python if-mon.py eth1/1 $command
switch(config-applet)# end
```

Python script:

If-mon.py

```
import re
import sys

def findIf ():
    x = re.compile ('[Ee]thernet\d+\/\d+')
    for a in sys.argv[1:]:
        if x.match (a):
            print a
            return a
    return None

print 'Starting my script.. args:'
print sys.argv
intf = findIf ()
if not intf:
    intf = 'eth1/1'

print 'Detected shut on interface %s' % intf

from cisco import *
s, o = cli ('show run int %s' % intf)
print ('----------------------\n%s\n' % o)
print 'Restoring interface %s' % intf
s, o = cfg_if (intf, desc='++dont shut++', state='up')
print ('----------------------\n%s\n' % o)
s, o = cli ('show run int %s' % intf)
print ('sh ver\n----------------------\n%s\n' % o)

print ('\nbye\n')
```

CHAPTER 2

Application Programming Interface (API) Functions

This chapter provides information about the following Python Application Programming Interface (API) functions. This chapter includes the following sections:

Routes()

Synopsis

```
Routes() - Class Object
```

Syntax

```
Routes()
```

Description

Instantiates an object of the Routes class.

Parameters

None.

Returns

An object of Routes class.

Example

```
rObj = Routes()
```

show_arp_table()

Synopsis

```
show_arp_table()
```

Syntax

```
Routes.show_arp_table()
```

Description

Executes the **show ip arp** command and returns the output.

Parameters

None.

Returns

Returns the ARP table entries on the switch.

Example

```
routeObj = Routes()
data = routeObj.show_arp_table().get_output()
```

Sample Output

```
Flags: D - Static Adjacencies attached to down interface

IP ARP Table for context default
Total number of entries: 4
Address         Age       MAC Address     Interface
50.1.201.2      00:02:10  547f.ee40.5a7c  Vlan201
50.1.1.10       00:07:53  547f.ee62.f801  Ethernet1/34
50.1.2.10       00:08:31  547f.ee62.f801  Ethernet1/35
50.1.3.10       00:08:31  547f.ee62.f801  Ethernet1/35.1

<cisco.CLI object at 0xb7c1462c>
```

show_vsh_routes()

Synopsis

```
show_vsh_routes()
```

Syntax

```
Routes.show_vsh_routes()
```

Description

Executes the show ip fib route and returns the output.

Parameters

None.

Returns

Returns the software route entries.

Example

```
routeObj = Routes()
data = routeObj.show_vsh_routes().get_output()
```

Sample Output

```
IPv4 routes for table default/base

------------------+------------------+----------------------
Prefix            | Next-hop         | Interface
------------------+------------------+----------------------
0.0.0.0/32          Drop               Null0
50.1.1.0/24         Attached           Ethernet1/34
50.1.1.0/32         Drop               Null0
50.1.1.10/32        50.1.1.10          Ethernet1/34
50.1.1.100/32       Receive            sup-eth1
50.1.1.255/32       Attached           Ethernet1/34
50.1.2.0/24         Attached           Ethernet1/35
50.1.2.0/32         Drop               Null0
50.1.2.10/32        50.1.2.10          Ethernet1/35
50.1.2.100/32       Receive            sup-eth1
50.1.2.255/32       Attached           Ethernet1/35
50.1.3.0/24         Attached           Ethernet1/35.1
50.1.3.0/32         Drop               Null0
50.1.3.10/32        50.1.3.10          Ethernet1/35.1
50.1.3.100/32       Receive            sup-eth1
50.1.3.255/32       Attached           Ethernet1/35.1

<cisco.CLI object at 0xb7b0a6ac>
```

show_hw_routes()

Synopsis

```
show_hw_routes()
```

Syntax

```
Routes.show_hw_routes()
```

Description

Computes the hardware routes and returns the output.

Parameters

None.

Returns

Returns the hardware route entries.

Example

```
routeObj = Routes()
data = routeObj.show_hw_routes()
```

Sample Output

```
------------------+------------------+---------------------
Prefix            | Next-hop         | Interface
------------------+------------------+---------------------
50.1.1.100/32       Receive            sup-eth1
50.1.2.100/32       Receive            sup-eth1
50.1.201.1/32       Receive            sup-eth1
0.0.0.0/32          Drop               Null0
50.1.3.0/32         Drop               Null0
50.1.201.0/32       Drop               Null0
50.1.2.255/32       Attached           sup-hi
50.1.1.255/32       Attached           sup-hi
60.1.1.0/32         Drop               Null0
50.1.3.255/32       Attached           sup-hi
50.1.201.255/32     Attached           sup-hi
255.255.255.255/32  Receive            sup-eth1
```

verify_routes()

Synopsis

```
verify_routes()
```

Syntax

```
Routes.verify_routes()
```

Description

Verifies the software and hardware routes.

Parameters

None.

Returns

Returns the number of routes matched and unmatched between hardware and software.

Example

```
routeObj = Routes()
found,nfound = routeObj.verify_routes()
```

Sample Output

```
Routes verified and found:  26

Routes not found:
50.1.205.0/24       3
51.1.1.0/24         3
51.1.2.0/24         4
51.1.3.0/24         6
100.1.1.0/24        7
100.1.2.0/24        7
100.1.3.0/24        7
101.1.1.0/24        7
101.1.2.0/24        7
101.1.3.0/24        7
120.1.1.0/24        7
```

verify_arp_table()

Synopsis

```
verify_arp_table()
```

Syntax

```
Routes.verify_arp_table()
```

Description

Verifies the software and hardware ARP table entries.

Parameters

None.

Returns

Returns the number of ARP table entries matched and unmatched between hardware and software.

Example

```
routeObj = Routes()
found,notfound = routeObj.verify_arp_table()
```

Sample Output

```
Flags: D - Static Adjacencies attached to down interface

IP ARP Table for context default
Total number of entries: 4
Address         Age       MAC Address     Interface
50.1.201.2      00:02:31  547f.ee40.5a7c  Vlan201
50.1.1.10       00:08:15  547f.ee62.f801  Ethernet1/34
50.1.2.10       00:08:53  547f.ee62.f801  Ethernet1/35
50.1.3.10       00:08:53  547f.ee62.f801  Ethernet1/35.1

mac address:54:7f:ee:40:5a:7c
Arp entry for 50.1.201.2 547f.ee40.5a7c Vlan201 found in HW
mac address:54:7f:ee:62:f8:01
Arp entry for 50.1.1.10 547f.ee62.f801 Ethernet1/34 found in HW
mac address:54:7f:ee:62:f8:01
Arp entry for 50.1.2.10 547f.ee62.f801 Ethernet1/35 found in HW
mac address:54:7f:ee:62:f8:01
```

```
Arp entry for 50.1.3.10 547f.ee62.f801 Ethernet1/35.1 found in HW
```

CheckPortDiscards()

Synopsis

```
CheckPortDiscards(<port>)
```

Syntax

```
CheckPortDiscards('ethernet1/1')
```

Description

Check the input discards for given port. If discard is more than 0, query and print the discard reason from broadcom.

Parameters

port

Returns

None.

Example

```
c = CheckPortDiscards('eth1/1')
```

Sample Output

```
Ethernet1/1 is up
  Hardware: 100/1000/10000 Ethernet, address: 547f.ee57.dd28 (bia 547f.ee57.dd28)
  MTU 1500 bytes, BW 10000000 Kbit, DLY 10 usec,
     reliability 255/255, txload 1/255, rxload 1/255
  Encapsulation ARPA
  Port mode is trunk
  full-duplex, 10 Gb/s, media type is 10G
  Beacon is turned off
  Input flow-control is off, output flow-control is off
  Rate mode is dedicated
  Switchport monitor is off
  EtherType is 0x8100
  Last link flapped 00:42:16
  Last clearing of "show interface" counters never
  30 seconds input rate 5016 bits/sec, 627 bytes/sec, 6 packets/sec
  30 seconds output rate 3232 bits/sec, 404 bytes/sec, 5 packets/sec
  Load-Interval #2: 5 minute (300 seconds)
    input rate 4.69 Kbps, 7 pps; output rate 2.82 Kbps, 4 pps
  RX
    297 unicast packets  20588 multicast packets  5 broadcast packets
    20890 input packets  1848701 bytes
    0 jumbo packets  0 storm suppression packets
    0 giants  0 input error  0 short frame  0 overrun   0 underrun
    0 watchdog  0 if down drop
    0 input with dribble  0 input discard(includes ACL drops)
    0 Rx pause
  TX
    262 unicast packets  16151 multicast packets  5 broadcast packets
    16418 output packets  1407200 bytes
    0 jumbo packets
    0 output errors  0 collision  0 deferred  0 late collision
```

```
    0 lost carrier  0 no carrier  0 babble
    0 Tx pause
  2 interface resets

zero discards
```

class BufferDepthMonitor(CLI)

Synopsis

```
BufferDepthMonitor()
```

Syntax

```
monitorObj = BufferDepthMonitor()
```

Description

Class parses **show hardware internal buffer info pkt-stats** command output and provides access method which returns buffer usage.

Parameters

None.

Returns

Object instance for this class.

Example

```
b = BufferDepthMonitor()
```

Sample Output

```
|-----------------------------------------------------------------------------------------------
|

                          Total Instant Usage             0
                          Remaining Instant Usage     46080
                          Max Cell Usage                 19
                          Switch Cell Count           46080
|-----------------------------------------------------------------------------------------------
|
<cisco.BufferDepthMonitor object at 0xb7c1462c>
```

get_total_instant_usage()

Synopsis

```
get_total_instant_usage()
```

Syntax

```
monitorObj = BufferDepthMonitor()
totUsage = monitorObj.get_total_instant_usage()
```

Description

Method which returns total instant usage from **show hardware internal buffer info pkt-stats** command output.

Parameters

None.

Returns

Returns total instant usage.

Example

```
b = BufferDepthMonitor()
b.get_total_instant_usage()
```

Sample Output

```
0
```

get_remaining_instant_usage()

Synopsis

```
get_remaining_instant_usage()
```

Syntax

```
monitorObj = BufferDepthMonitor()
remUsage = monitorObj.get_remaining_instant_usage()
```

Description

Method which returns remaining instant usage from **show hardware internal buffer info pkt-stats** command output.

Parameters

None.

Returns

Returns total instant usage.

Example

```
b = BufferDepthMonitor()
b.get_remaining_instant_usage()
```

Sample Output

```
46080
```

get_max_cell_usage()

Synopsis

```
get_max_cell_usage()
```

Syntax

```
monitorObj = BufferDepthMonitor()
cellUsage = monitorObj.get_max_cell_usage()
```

Description

Method which returns cell usage from **show hardware internal buffer info pkt-stats** command output.

Parameters

None.

Returns

Returns total instant usage.

Example

```
b = BufferDepthMonitor()
b.get_max_cell_usage()
```

Sample Output

```
19
```

get_switch_cell_count()

Synopsis

```
get_switch_cell_count()
```

Syntax

```
monitorObj = BufferDepthMonitor()
cellCount = monitorObj.get_switch_cell_count()
```

Description

Method which returns cell count usage from **show hardware internal buffer info pkt-stats** command output.

Parameters

None.

Returns

Returns total instant usage.

Example

```
b = BufferDepthMonitor()
b.get_switch_cell_count()
```

Sample Output

```
46080
```

transfer()

Synopsis

```
transfer()
```

Syntax

```
transfer (<protocol>, <host>, <source>, <dest>, <vrf>, <login_timeout>, <user>,
<password>)
```

Description

API to transfer file specified in <source> from <host> to the path mentioned in <dest> using <protocol>. **Protocol can be scp, tftp, ftp or sftp**.

Parameters

protocol, host, source, dest, vrf, login_timeout, user, password.

Returns

Returns True if transfer was successful.

Example

Transfer using scp:

```
c = transfer("scp", "10.193.190.100", "/tftpboot/transfer_test_image",
"transfer_test_image", user="scpUser", password="scpPasswd")
```

Transfer using sftp:

```
c = transfer("sftp", "10.193.190.100", "/tftpboot/transfer_test_image",
"transfer_test_image", user="sftpUser", password="sftpPasswd")
```

Transfer using tftp:

```
c = transfer("tftp", "10.193.190.100", "/transfer_test_image", "transfer_test_image",
user="", password="")
```

Transfer using ftp:

```
c = transfer("ftp", "10.193.190.51", "golden/home/su-ash/transfer_test_image",
"transfer_test_image", user="ftpUser", password="ftpPasswd")
```

CLI()

Synopsis

```
CLI() - Class Object
```

Syntax

```
CLI (<command>, <do_print>)
```

Description

Instantiates an object of the CLI class with the CLI command specified in <command>. <do_print> when set to False does not print the output of the command and prints the output when set to True, which is the default.

Parameters

command, do_print

Returns

An object of CLI class.

Example

```
c = CLI ('show runn inter eth1/1')
```

Sample Output

```
!Command: show running-config interface Ethernet1/1
!Time: Mon Feb 27 14:33:24 2012

version 5.0(3)U3(1)

interface Ethernet1/1
  switchport mode trunk
  udld enable
  channel-group 12

<cisco.CLI object at 0xb7ae948c>
```

get_output()

Synopsis

```
get_output()
```

Syntax

```
CLI.get_output()
```

Description

Returns the output of the CLI command.

Parameters

None.

Returns

Output of the CLI command.

Example

```
c = CLI ('show runn inter eth1/1')
c.get_output()
```

Sample Output

```
['', '!Command: show running-config interface Ethernet1/1', '!Time: Mon Feb 27 14:36:10
2012', '', 'version 5.0(3)U3(1)', '', 'interface Ethernet1/1', '  switchport mode trunk',
'  udld enable', '  channel-group 12', '', '']
```

rerun()

Synopsis

```
rerun()
```

Syntax

```
CLI.rerun()
```

Description

Reruns the command.

Parameters

None.

Returns

None.

Example

```
c = CLI ('show runn inter eth1/1')
c.rerun()
```

Sample Output

```
!Command: show running-config interface Ethernet1/1
!Time: Mon Feb 27 14:37:05 2012

version 5.0(3)U3(1)

interface Ethernet1/1
  switchport mode trunk
  udld enable
  channel-group 12
```

History()

Synopsis

History() - Class Object

Syntax

History()

Description

Instantiates an object of the History class.

Parameters

None.

Returns

An object of History class.

Example

```
a = History()
```

get_history()

Synopsis

```
get_history()
```

Syntax

```
History.get_history()
```

Description

Gets the history of CLI commands executed so far.

Parameters

None.

Returns

Returns the history of commands executed.

Example

```
a = History()
a.get_history()
```

clear_history()

Synopsis

```
clear_history()
```

Syntax

```
History.clear_history()
```

Description

Clears history.

Parameters

None.

Returns

None.

Example

```
a = History()
a.clear_history()
```