



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, page 1-1](#)
- [Installing Python, page 1-2](#)
- [Installing Third Party Pure Python Packages, page 1-2](#)
- [Using Python, page 1-2](#)

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_new_API_names.py
```

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

- [Entering Python Shell, page 1-3](#)
- [Executing Scripts, page 1-3](#)
- [Passing Parameters to the Script, page 1-3](#)
- [Embedded Event Manager Support, page 1-3](#)

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')
```