Overview

This chapter provides an overview and installation information needed to use the Python application programming interface (API) support on Cisco Nexus 3000 Series switches.

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 distributes and points 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 allows you to perform the following tasks:

- Running a script to verify configuration on switch bootup.
- Backing 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 in the Cisco NX-OS software.You can invoke Python by entering the **python** command, and you can write scripts to access Cisco NX-OS APIs by importing the cisco.py module using the **import cisco** 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 future releases.

Using Python

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

Entering the 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 abc xyz 1 2
['/bootflash/test.py', 'abc', 'xyz', '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:

```
import re
import sys
from cisco import *

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
i = Interface (intf)
```

```
print ('---------show run int %s---------\n%s\n' % (intf, i.show().raw_output))
print ('----------------------\n%s\n' % o)
print 'Restoring interface %s' % intf
i.set_state ('up')
i.set_description ('++dont shut++')
print ('----------------------\n%s\n' % o)
print ('---------show run int %s---------\n%s\n' % (intf, i.show().raw_output))

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

Calculating MD5 Checksum

Every time you make a change to the configuration script, ensure that you recalculate the MD5 checksum. The following example shows the script to be run in a bash shell to recalculate the MD5 checksum:

```
#!/bin/env python
#md5sum="13f71ffebe16cc6143a6af2186b40948"
# If any changes to this script file are made, please run the below command
# in bash after modifications.
# The above is the (embedded) md5sum of this file taken without this line,
# can be # created this way if using a bash shell:
# f=poap_fabric.py ; cat $f | sed '/^#md5sum/d' > $f.md5 ; sed -i
"s/^#md5sum=.*/#md5sum=\"$(md5sum $f.md5 | sed 's/ .*//')\"/" $f
# This way this script's integrity can be checked in case you do not trust
# tftp's ip checksum. This integrity check is done by /isan/bin/poap.bin).
# The integrity of the files downloaded later (images, config) is checked
# by downloading the corresponding file with the .md5 extension and is
# done by this script itself.
```