



Python API

Python programmability supports Python APIs.

- [Feature Information for Python API, on page 1](#)
- [About Python , on page 1](#)
- [Additional References for Python API, on page 6](#)
- [Additional References for Python API, on page 6](#)

Feature Information for Python API

The following table provides release information about the feature or features described in this module. This table lists only the software release that introduced support for a given feature in a given software release train. Unless noted otherwise, subsequent releases of that software release train also support that feature.

Use Cisco Feature Navigator to find information about platform support and Cisco software image support. To access Cisco Feature Navigator, go to www.cisco.com/go/cfn. An account on Cisco.com is not required.

Table 1: Feature Information for the CLI Python Module

Feature Name	Release	Feature Information
CLI Python Module	Cisco IOS XE 17.18.1	Python programmability provides a Python module that allows users to interact with IOS using CLIs. In Cisco IOS XE 17.18.1, this feature was implemented on the following platforms: <ul style="list-style-type: none">• Cisco Catalyst 9350 Series Switches

About Python

The Cisco IOS XE devices support Python Version 2.7 in both interactive and non-interactive (script) modes within the Guest Shell. The Python scripting capability gives programmatic access to a device's CLI to perform various tasks and Zero Touch Provisioning or Embedded Event Manager (EEM) actions.

Cisco Python Module

Cisco provides a Python module that provides access to run EXEC and configuration commands. You can display the details of the Cisco Python module by entering the `help()` command. The `help()` command displays the properties of the Cisco CLI module.

The following example displays information about the Cisco Python module:

```
Device# guestshell run python

Python 2.7.5 (default, Jun 17 2014, 18:11:42)
[GCC 4.8.2 20140120 (Red Hat 4.8.2-16)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>> >>> from cli import cli,clip,configure,configurep, execute, executep
>>> help(configure)
Help on function configure in module cli:

configure(configuration)
Apply a configuration (set of Cisco IOS CLI config-mode commands) to the device
and return a list of results.

configuration = '''interface gigabitEthernet 0/0
no shutdown'''

# push it through the Cisco IOS CLI.
try:
results = cli.configure(configuration)
print "Success!"
except CLIConfigurationError as e:
print "Failed configurations:"
for failure in e.failed:
print failure

Args:
configuration (str or iterable): Configuration commands, separated by newlines.

Returns:
list(ConfigResult): A list of results, one for each line.

Raises:
CLISyntaxError: If there is a syntax error in the configuration.

>>> help(configurep)
Help on function configurep in module cli:

configurep(configuration)
Apply a configuration (set of Cisco IOS CLI config-mode commands) to the device
and prints the result.

configuration = '''interface gigabitEthernet 0/0
no shutdown'''

# push it through the Cisco IOS CLI.
configurep(configuration)

Args:
configuration (str or iterable): Configuration commands, separated by newlines.
>>> help(execute)
Help on function execute in module cli:

execute(command)
```

Execute Cisco IOS CLI exec-mode command and return the result.

```
command_output = execute("show version")
```

Args:

command (str): The exec-mode command to run.

Returns:

str: The output of the command.

Raises:

CLISyntaxError: If there is a syntax error in the command.

```
>>> help(executep)
```

Help on function executep in module cli:

```
executep(command)
```

Execute Cisco IOS CLI exec-mode command and print the result.

```
executep("show version")
```

Args:

command (str): The exec-mode command to run.

```
>>> help(cli)
```

Help on function cli in module cli:

```
cli(command)
```

Execute Cisco IOS CLI command(s) and return the result.

A single command or a delimited batch of commands may be run. The delimiter is a space and a semicolon, " ;". Configuration commands must be in fully qualified form.

```
output = cli("show version")
```

```
output = cli("show version ; show ip interface brief")
```

```
output = cli("configure terminal ; interface gigabitEthernet 0/0 ; no shutdown")
```

Args:

command (str): The exec or config CLI command(s) to be run.

Returns:

string: CLI output for show commands and an empty string for configuration commands.

Raises:

errors.cli_syntax_error: if the command is not valid.

errors.cli_exec_error: if the execution of command is not successful.

```
>>> help(clip)
```

Help on function clip in module cli:

```
clip(command)
```

Execute Cisco IOS CLI command(s) and print the result.

A single command or a delimited batch of commands may be run. The delimiter is a space and a semicolon, " ;". Configuration commands must be in fully qualified form.

```
clip("show version")
```

```
clip("show version ; show ip interface brief")
```

```
clip("configure terminal ; interface gigabitEthernet 0/0 ; no shutdown")
```

Args:
 command (str): The exec or config CLI command(s) to be run.

Cisco Python Module to Execute IOS CLI Commands



Note Guest Shell must be enabled for Python to run. For more information, see the *Guest Shell* chapter.

The Python programming language uses six functions that can execute CLI commands. These functions are available from the Python CLI module. To use these functions, execute the **import cli** command.

Arguments for these functions 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 six functions:

- **cli.cli(command)**—This function takes an IOS command as an argument, runs the command through the IOS parser, and returns the resulting text. If this command is malformed, a Python exception is raised. The following is sample output from the **cli.cli(command)** function:

```
>>> import cli
>>> cli.cli('configure terminal; interface loopback 10; ip address
10.10.10.10 255.255.255.255')
*Mar 13 18:39:48.518: %LINEPROTO-5-UPDOWN: Line protocol on Interface Loopback10, changed
state to up
>>> cli.cli('show clock')
'\n*18:11:53.989 UTC Mon Mar 13 2017\n'
>>> output=cli.cli('show clock')
>>> print(output)
*18:12:04.705 UTC Mon Mar 13 2017
```

- **cli.clip(command)**—This function works exactly the same as the **cli.cli(command)** function, except that it prints the resulting text to *stdout* rather than returning it. The following is sample output from the **cli.clip(command)** function:

```
>>> cli
>>> cli.clip('configure terminal; interface loopback 11; ip address
10.11.11.11 255.255.255.255')
*Mar 13 18:42:35.954: %LINEPROTO-5-UPDOWN: Line protocol on Interface Loopback11, changed
state to up
*Mar 13 18:42:35.954: %LINK-3-UPDOWN: Interface Loopback11, changed state to up
>>> cli.clip('show clock')
*18:13:35.313 UTC Mon Mar 13 2017
>>> output=cli.clip('show clock')
*18:19:26.824 UTC Mon Mar 13 2017
>>> print(output)
None
```

- **cli.execute(command)**—This function executes a single EXEC command and returns the output; however, does not print the resulting text. No semicolons or newlines are allowed as part of this command. Use a Python list with a for-loop to execute this function more than once. The following is sample output from the **cli.execute(command)**

function:

```
>>> cli.execute("show clock")
'15:11:20.816 UTC Thu Jun 8 2017'
>>>
>>> cli.execute('show clock'; 'show ip interface brief')
File "<stdin>", line 1
    cli.execute('show clock'; 'show ip interface brief')
                        ^
SyntaxError: invalid syntax
>>>
```

- **cli.executep(command)**—This function executes a single command and prints the resulting text to *stdout* rather than returning it. The following is sample output from the **cli.executep(command)** function:

```
>>> cli.executep('show clock')
*18:46:28.796 UTC Mon Mar 13 2017
>>> output=cli.executep('show clock')
*18:46:36.399 UTC Mon Mar 13 2017
>>> print(output)
None
```

- **cli.configure(command)**—This function configures the device with the configuration available in commands. It returns a list of named tuples that contains the command and its result as shown below:

```
[Think: result = (bool(success), original_command, error_information)]
```

The command parameters can be in multiple lines and in the same format that is displayed in the output of the **show running-config** command. The following is sample output from the **cli.configure(command)** function:

```
>>>cli.configure(["interface GigabitEthernet1/0/7", "no shutdown",
"end"])
[ConfigResult(success=True, command='interface GigabitEthernet1/0/7',
line=1, output='', notes=None), ConfigResult(success=True, command='no shutdown',
line=2, output='', notes=None), ConfigResult(success=True, command='end',
line=3, output='', notes=None)]
```

- **cli.configurep(command)**—This function works exactly the same as the **cli.configure(command)** function, except that it prints the resulting text to *stdout* rather than returning it. The following is sample output from the **cli.configurep(command)** function:

```
>>> cli.configurep(["interface GigabitEthernet1/0/7", "no shutdown",
"end"])
Line 1 SUCCESS: interface GigabitEthernet1/0/7
Line 2 SUCCESS: no shut
Line 3 SUCCESS: end
```

Python Scripts Overview

Python run in a virtualized Linux-based environment, Guest Shell. For more information, see the *Guest Shell* chapter. Cisco provides a Python module that allows user's Python scripts to run IOS CLI commands on the host device.

Using Python

Additional References for Python API

Related Documents

Related Topic	Document Title
Guest Shell	Guest Shell
EEM Python Module	Python Scripting in EEM

Technical Assistance

Description	Link
<p>The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies.</p> <p>To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds.</p> <p>Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.</p>	http://www.cisco.com/support

Additional References for Python API

Related Documents

Related Topic	Document Title
Guest Shell	Guest Shell
EEM Python Module	Python Scripting in EEM

Technical Assistance

Description	Link
<p>The Cisco Support website provides extensive online resources, including documentation and tools for troubleshooting and resolving technical issues with Cisco products and technologies.</p> <p>To receive security and technical information about your products, you can subscribe to various services, such as the Product Alert Tool (accessed from Field Notices), the Cisco Technical Services Newsletter, and Really Simple Syndication (RSS) Feeds.</p> <p>Access to most tools on the Cisco Support website requires a Cisco.com user ID and password.</p>	http://www.cisco.com/support

