CiscoAutomationFramework

Release 1.0.6

May 24, 2023

Contents:

The CiscoAutomation Framework is designed to be a library in Python that abstracts away CLI navigation and scraping so a network administrator can interact with a Cisco device and they need to worry much less about parsing output or navigating to the correct CLI section when issuing commands.

CHAPTER 1

How To Connect to A Network Device

Calling the CiscoAutomationFramework.connect_ssh function is how you connect to a device. it sets up the connection and all of the low level library and hands you an instantiated child of the CiscoFirmware object specific to the device type you are connected to.

You can connect by assigning the output directly to a variable:

```
from CiscoAutomationFramework import connect_ssh
ssh = connect_ssh('ip', 'username', 'password')
#other code here
ssh.close_connection()
```

However It is recommended to use a context manager so you dont have to remember to close the connection or if an exception if hit in your code it closes for you:

```
from CiscoAutomationFramework import connect_ssh
with connect_ssh('ip', 'username', 'password') as ssh:
    # Code here while logged into the device
# Code here while logged out of the device
```

CiscoAutomationFramework.**connect_ssh**(*ip*, *username*, *password*, *port=22*, *enable_password=None*, *timeout=10*) → CiscoAutomationFramework.FirmwareBase.CiscoFirmware

Connects to your cisco device, returns a firmware specific instance of CiscoFirmware object.

Parameters

- **ip** (*str*) – IP address or hostname of Cisco device
- **username** (*str*) – Username used to login
- **password** (*str*) – Password for user
- **port** (*int*) – Port to use (default 22)
- **enable_password** (*str*) – Enable password to use if the user does not have privileges directly to privilege exec
- **timeout** (*int*) – SSH timeout in seconds

Returns CiscoFirmware Object

Return type *CiscoFirmware*

CHAPTER 2

Interacting with a Network Device

After connecting to a device by running the CiscoAutomationFramework.connect_ssh you receive a child of this class that is specific to IOS/Nexus based on what is detected you are connected to. You will have all of the attributes documented here to interact with your Cisco device regardless of its firmware type.

class CiscoAutomationFramework.FirmwareBase.**CiscoFirmware**(*transport*)

add_local_user(*username*, *password*, *password_code=0*, **args*, ***kwargs*)
Method here should be generating a string that the network device accepts in the following format 'username USERNAME <args> <kwarg key> <kwarg value> password PASSWORD_CODE PASSWORD'

Parameters

- **username** – Username to use
- **password** – Password to set (cleartext or encrypted depending on password_code)
- **password_code** – Code for password you are providing
- **args** – Single word arguments in command string
- **kwargs** – Key word arguments in command string

Returns Nothing

arp_table
Returns the arp table in its raw form

Returns Arp Table

Return type str

cli_to_config_mode() → bool
Navigates the CLI into config mode regardless of where it is

Returns True/False

Return type bool

cli_to_privileged_exec_mode() → bool
Navigates the CLI into prvileged exec mode regardless of where it is. Will raise an EnablePasswordError if the transport engine does not have an enable password set and the network device asks for one.

Returns True/False

Return type bool

Raises CiscoAutomationFramework.Exceptions.EnablePasswordError

close_connection() → None
Closes connection to device

Returns Nothing

commands_sent
A list of all commands sent to the device in the order they are entered from first to last in the current session

Returns List of commands sent

Return type list

delete_local_user(*username*)
Deletes a locally defined user on device

Parameters **username** (*str*) – username to delete

Returns Nothing

get_output(*buffer_size=100*, *timeout=1*) → list
Gets output from the device until the prompt is returned or the timeout is reached. You should not need to run this directly often.

Parameters

- **buffer_size** – Size of buffer when getting output from device. You shouldnt have to modify this much
- **timeout** – Time to stop waiting for output if no output is received.

Returns

hostname
Hostname of the Cisco device.

Returns Hostname

Return type str

interfaces
List of interfaces on device

Returns Interfaces

Return type list

is_nexus
Returns True/False if the Cisco device is a Nexus

Returns True/False

Return type bool

mac_address_table
Returns the MAC address table in its raw form

Returns MAC Address Table

Return type str

prompt
Returns the latest prompt that was returned from the device

Returns Latest prompt returned from device

Return type str

running_config
Returns running config in its raw form

Returns Running Configuration

Return type str

save_config()
Saves running config to startup config

Returns Nothing

send_command(*command*, *end='\n'*) → None

Sends command to device, does not get any output. You should not need to run this directly often

param command Command to send

param end End character (default

)

return Nothing

send_command_get_output(*command*, *end='\n'*, *buffer_size=100*, *timeout=1*, *delay=0.5*) → list

Sends a command to the device and returns the output from the device. If there were multiple commands sent this will gather the output from all the commands at once. This command is a combination of send_command and get_output.

param command Command to send

param end End character (default

)

param buffer_size Size of buffer when getting output from device. You shouldnt have to modify this much

param timeout Time to stop waiting for output if no output is received.

param delay Time to wait before gathering output from device

return Output from device starting with command, ending with prompt in a list split by line

rtype list

send_question_get_output(*command*) → list
Special method to get the output from sending a command with a question mark. This will send the command with a trailing '?' get the output from that and then erase the command that is returned on the ending prompt. You should NOT put a question mark in the command this function will handle that for you but if you do, it will be removed so the CLI doesnt do something that is not expected

Parameters **command** – Command to inspect

Returns list of possible command completions, or next possible words in command

Return type list

startup_config
Returns startup config in its raw form

Returns Startup Config

Return type str

terminal_length(*n='0'*)
Sets terminal length of shell

Parameters **n** (*str*) – length

Returns Nothing

terminal_width(*n='0'*)
Sets terminal width of shell

Parameters **n** (*str*) – width

Returns

uptime
Returns uptime of device

Returns Uptime

Return type str

CHAPTER 3

Interacting with Multiple Network Devices at the same time

CiscoAutomationFramework provides a way of running the same code on multiple devices concurrently. It takes into account situations where you will have a mix of IOS and Nexus and when you dont care, providing object to inherit from and override in both circumstances.

class CiscoAutomationFramework.ThreadLib.**SSH**(*ip*, *username*, *password*, *enable_password=None*, *perform_secondary_action=False*, ***kwargs*)

Base SSH object that you can inherit from when building a threaded script that will run across the network.

To build a script you must subclass this object and override the methods that are documented.

Method Execution Order The object logs into the device during_login is executed if perform_secondary_action is set to True secondary_action is executed and then post_secondary_action

during_login(*ssh*)

Every time this object runs against a device this method will be run. Override this method and add logic that needs to be run every time your object runs against a device.

I find I often add logic here that gathers data and does not modify configuration.

Parameters **ssh** (CiscoFirmware) – SSH object is provided to this method for interacting with the end device

Returns Nothing

post_secondary_action(*ssh*)

This method will run only if perform_secondary_action is set to True and AFTER secondary_action has completed.

I often find this is useful for containing logic that re gathers data or saves configuration.

Parameters **ssh** (CiscoFirmware) – SSH object is provided to this method for interacting with the end device

Returns Nothing

run() → None

Method representing the thread's activity.

You may override this method in a subclass. The standard run() method invokes the callable object passed to the object's constructor as the target argument, if any, with sequential and keyword arguments taken from the args and kwargs arguments, respectively.

secondary_action(*ssh*)
This method will run only if perform_secondary_action is set to True.

I often find this is useful for containing logic that modifies configuration that only needs to be run if a certain condition is met.

Parameters **ssh** (CiscoFirmware) – SSH object is provided to this method for interacting with the end device

Returns Nothing

class CiscoAutomationFramework.ThreadLib.**SSHSplitDeviceType**(*ip*, *username*, *password*, *enable_password=None*, *perform_secondary_action=False*, ***kwargs*)
Create a script by inheriting from this object if you are running the same script against both Nexus and IOS devices but the command are different on each platform or the sequence is different. Out of the box it will run the functions prepended with ios on ios and iosxe devices and functions prepended with nexus on nxos devices. Because it is implied that this script will be run against a mix of nexus and ios you must define the functions regardless if they will be used or not when inheriting from this object

during_login(*ssh*)
Every time this object runs against a device this method will be run. Override this method and add logic that needs to be run every time your object runs against a device.

I find I often add logic here that gathers data and does not modify configuration.

Parameters **ssh** (CiscoFirmware) – SSH object is provided to this method for interacting with the end device

Returns Nothing

ios_during_login(*ssh*)
Every time this object runs against a device that is detected to be NOT Nexus this method will be run. Override this method and add logic that needs to be run every time your object runs against a device.

Must be overridden when inheriting from this object regardless if your script will run against this device type

I find I often add logic here that gathers data and does not modify configuration.

Parameters **ssh** (CiscoFirmware) – SSH object is provided to this method for interacting with the end device

Returns Nothing

ios_post_secondary_action(*ssh*)
This method will run only if perform_secondary_action is set to True and AFTER secondary_action has completed and the device is detected to be NOT Nexus.

Must be overridden when inheriting from this object regardless if your script will run against this device type.

I often find this is useful for containing logic that re gathers data or saves configuration.

Parameters **ssh** (CiscoFirmware) – SSH object is provided to this method for interacting with the end device

Returns Nothing

ios_secondary_action(*ssh*)

This method will run only if perform_secondary_action is set to True and the device is detected as not Nexus.

Must be overridden when inheriting from this object regardless if your script will run against this device type

I often find this is useful for containing logic that modifies configuration that only needs to be run if a certain condition is met.

Parameters **ssh** (CiscoFirmware) – SSH object is provided to this method for interacting with the end device

Returns Nothing

nexus_during_login(*ssh*)

Every time this object runs against a device that is detected to be running Nexus this method will be run. Override this method and add logic that needs to be run every time your object runs against a device.

Must be overridden when inheriting from this object regardless if your script will run against this device type

I find I often add logic here that gathers data and does not modify configuration.

Parameters **ssh** (CiscoFirmware) – SSH object is provided to this method for interacting with the end device

Returns Nothing

nexus_post_secondary_action(*ssh*)

This method will run only if perform_secondary_action is set to True and AFTER secondary_action has completed and the device is detected to be Nexus.

Must be overridden when inheriting from this object regardless if your script will run against this device type.

I often find this is useful for containing logic that re gathers data or saves configuration.

Parameters **ssh** (CiscoFirmware) – SSH object is provided to this method for interacting with the end device

Returns Nothing

nexus_secondary_action(*ssh*)

This method will run only if perform_secondary_action is set to True and the device is detected Nexus.

Must be overridden when inheriting from this object regardless if your script will run against this device type.

I often find this is useful for containing logic that modifies configuration that only needs to be run if a certain condition is met.

Parameters **ssh** (CiscoFirmware) – SSH object is provided to this method for interacting with the end device

Returns Nothing

post_secondary_action(*ssh*)

This method will run only if perform_secondary_action is set to True and AFTER secondary_action has completed.

I often find this is useful for containing logic that re gathers data or saves configuration.

Parameters **ssh** (CiscoFirmware) – SSH object is provided to this method for interacting with the end device

Returns Nothing

secondary_action(*ssh*)
This method will run only if perform_secondary_action is set to True.

I often find this is useful for containing logic that modifies configuration that only needs to be run if a certain condition is met.

Parameters **ssh** (CiscoFirmware) – SSH object is provided to this method for interacting with the end device

Returns Nothing

CiscoAutomationFramework.ThreadLib.**start_threads**(*object*, *ips*, *username*, *password*, *enable_password=None*, *perform_secondary_action=False*, *wait_for_threads=False*, ***kwargs*)
This helper function is a quick and easy way to start your threads. Gives you an option for waiting for threads to complete also. It will return the thread objects from the function for further processing by the main thread.

Parameters

- **object** (*[SSH, SSHSplitDeviceType]*) – Object you have written to perform your task. Must be inherted from SSH or SSHSplitDeviceType
- **ips** (*list[str]*) – List of IP addresses or hostnames to run your worker object against
- **username** (*str*) – Username to login with
- **password** (*str*) – Password for user
- **enable_password** (*str*) – Enable password if user does not have rights to privilege exec mode without enable password
- **perform_secondary_action** (*bool*) – True if you want to run the secondary_action method against device
- **wait_for_threads** (*bool*) – Wait for all threads to complete before returning
- **kwargs** – Keyword arguments to be passed to your object. If your child class accepts additional arguments, pass them to the object via keyword arguments here.

Returns List of threads either running or completed depending on if wait_for_threads is True/False

Return type list[*SSH*, *SSHSplitDeviceType*, type(object)]

3.1 Example Scripts

Below is an example script that will run concurrently across multiple network devices:

```
from CiscoAutomationFramework.ThreadLib import SSH, start_threads


class MyDevice(SSH):

    def __init__(self, *args, **kwargs):
        super().__init__(*args, **kwargs)
        self.interfaces_connected = []
```

(continues on next page)

(continued from previous page)

```
    def during_login(self, ssh):
        # Code here will run every time no matter what
        for line in ssh.send_command_get_output('show interface status'):
            if 'connected' in line and not 'notconn' in line:
                self.interfaces_connected.append(line.split()[0])

    def secondary_action(self, ssh):
        """
        If there is some secondary action needed to be performed, code it here
        If there is nothing you need to do you DO NOT need to define this method

        To execute this, when starting threads via the start_threads method set
        perform_secondary_action=True
        """
        pass

    def post_secondary_action(self, ssh):
        """
        If you need to something post the secondary action taking place, code it here.
        an example would be re gathering data or saving config

        If there is nothing you need to do you DO NOT need to define this method

        To execute this, when starting threads via the start_threads method set
        perform_secondary_action=True
        """
        pass


if __name__ == '__main__':

    username = 'myusername'
    password = 'mypassword'

    # replace with IP addresses of network devices on your network
    ips = ['192.168.1.1', '192.168.2.1', '192.168.3.1', '192.168.4.1', '192.168.5.1',
↪'192.168.6.1']
    threads = start_threads(MyDevice, ips, username, password, wait_for_threads=True)
    for thread in threads:
        print(thread.hostname, len(thread.interfaces_connected))
```

CHAPTER 4

Command Output Parsers

CiscoAutomationFramework provides integrated parsers for various command outputs. These parsers allow you to pass in the output from the network device and be able to iterate over output and/or extract data from the output in a programatic way versus needing to build your own parsers

4.1 Show Interface Status Parser

Pass in the raw output from "show interface status" command to this parser and you will be able to iterate over the table and interact with the entries individually

class CiscoAutomationFramework.Parsers.InterfaceStatusParser.**InterfaceStatusOutputParser**(*ra*

Provide the output directly from the device after issuing the command 'show interface status' to this object to parse it

interfaces

Parses the raw output from the command and provides a list of all entries from the table

Returns LineParser

Return type list[*LineParser*]

class CiscoAutomationFramework.Parsers.InterfaceStatusParser.**LineParser**(*raw_line*)

description

Description set on interface

Returns Confiured Description

Return type str

duplex

Interface duplex settings ex. auto, a-full a-half

Returns Duplex Setting

Return type str

name
Name of the interface ex. Gi1/0/1

Returns Interface Name abbriviated

Return type str

not_present
Returns True/False if the text "not present" in the lowercase line

Returns True/False

Return type bool

speed
Interface Operating Speed ex. auto, a-100, a-1000

Returns Interface Speed

Return type str

status
Status of interface ex. connected, notconnected, disabled, err-disable, etc

Returns interface status

Return type str

type
Interface Type ex. 10/100/1000BaseTX, Not Present

Returns Interface Type

Return type str

vlan
Vlan configuration of interface ex. 1, 100, trunk, routed

Returns vlan configuration

Return type str

4.2 IP Device Tracking Parser

Pass in the raw output from "show ip device tracking all" and this parser allows you to iterate over the contents of the table one at a time.

class CiscoAutomationFramework.Parsers.IpDeviceTrackingParser.**DeviceTrackingOutputParser**(*ou*

entries
Extracts the entires from the table and returns them as a list

Returns List of table entries as EntryParser instances

Return type list[*EntryParser*]

class CiscoAutomationFramework.Parsers.IpDeviceTrackingParser.**EntryParser**(*entry_data*)

interface
Extracts interface of the ip device tracking entry

Returns Switch Interface

Return type str

ip_address
IP address of the IP device tracking entry

Returns IP address

Return type str

is_apipa
True/False if the IP address in the entry is an APIPA address.

Returns True/False

Return type bool

mac_address
MAC address of the ip device tracking entry

Returns MAC Address

Return type str

probe_timeout
Extracts probe timeout of the ip device tracking entry

Returns Probe Timeout

Return type str

source
Extracts source of the ip device tracking entry

Returns Source

Return type str

state
Extracts state of the ip device tracking entry

Returns State

Return type str

vlan
Extracts vlan of the ip device tracking entry

Returns Vlan

Return type str

4.3 MAC Address Table Parser

Pass in the raw output from "show mac address-table" and this parser allows you to iterate over the contents of the table one at a time and also analyze the table in other ways.:

```
from CiscoAutomationFramework import connect_ssh
from CiscoAutomationFramework.Parsers.MacAddressTableParser import␣
↪MacAddressTableParser

with connect_ssh('ip', 'username', 'password') as ssh:
   mac_table = MacAddressTableParser(ssh.mac_address_table)
```

(continues on next page)

(continued from previous page)

```
for entry in mac_table:
    print(f'{entry.interface} - {entry.vlan} - {entry.mac_address}')
```

class CiscoAutomationFramework.Parsers.MacAddressTableParser.**MacAddressTableParser**(*raw_table*)

is_nexus
Checks the raw table for specific key words to determine if it is a Nexus platform or not returning True if it is False if it is not

Returns True/False

Return type bool

table_entries
Parses the mac address table and returns a list of the entries

Returns List of entries from the MAC address table

Return type list[*MacEntryParser*]

class CiscoAutomationFramework.Parsers.MacAddressTableParser.**MacEntryParser**(*raw_entry*)

interface
Interface on device that the MAC address is on ex. Gi1/0/8

Returns Interface on Device

Return type str

is_nexus
Searches the entire mac address table, checking if any of a set of keywords are in it

mac_address
MAC address from the row

Returns MAC Address

Return type str

type
Address type from the row ex. dynamic, static

Returns Address Type

Return type str

vlan
Vlan from the row

Returns Vlan

Return type str

4.4 Power Inline Parser

Pass the raw output from 'show power inline' to this parser and you will be able to iterate over the entries in the power inline table:

```
from CiscoAutomationFramework import connect_ssh
from CiscoAutomationFramework.Parsers.PowerInlineParser import PowerInlineParser

with connect_ssh('ip', 'username', 'password') as ssh:
    parser = PowerInlineParser(ssh.send_command_get_output('show power inline'))

for entry in parser:
    print(f'{entry.name} - {entry.watts} - {entry.detected_device}')
```

class CiscoAutomationFramework.Parsers.PowerInlineParser.**PowerInlineParser**(*sh_power_inline*)

interfaces
Returns list of interfaces in the power inline table to extract the values from the entry

Returns List of entries in the power inline table

Return type list[*PowerInlineInterface*]

class CiscoAutomationFramework.Parsers.PowerInlineParser.**PowerInlineInterface**(*interface_data*)

admin
Admin column of the entry ex. auto

Returns Admin column

Return type str

detected_device
Detected device type of connected device

Returns Detected device

Return type str

max_watts
Max watts capable of being drawn on port

Returns Max watts supported

Return type float

name
Interface name ex. Gi1/0/5

Returns Interface Name

Return type str

oper
Oper column of the entry, off/on weather it is providing power or not

Returns Operation status

Return type str

poe_class
POE class of connected device ex. 0, 1, 2, 3, n/a

Returns POE Class

Return type str

watts
Watts currently being drawn on port by connected device

Returns Watts currently drawn

Return type float

CHAPTER 5

Utilities

CiscoAutomationFramework also provides some integrated utility functions for doing various things with data from switches such as abbreviating interfaces, checking if IP addresses are valid, and printing output in a nice format.

CiscoAutomationFramework.util.**abbreviate_interface**(*interface_string*, *max_chars=2*)

Takes an Interface name ex. GigabitEthernet1/0/4 and abbreviates it ex Gi1/0/4

Parameters **interface_string** (*str*) – Interface String ex. GigabitEthernet1/0/4

:param :return: Abbreviated interface string :rtype: str

CiscoAutomationFramework.util.**column_print**(*data*, *spaces_between_columns=2*, *separator_char=None*)

Print in nice even columns where each column is only the width it needs to be, not the width of the widest column in the list. This will give output in a table format similar to much Windows/Linux CLI output. Based off of code from https://stackoverflow.com/questions/9989334/create-nice-column-output-in-python

Parameters

- **data** (*list[Union[list, tuple]]*) – List of List/Tuples of equal length, the first list being the header, all the rest being the data.
- **spaces_between_columns** (*int*) – Number of spaces to be between each column (default 2)
- **separator_char** (*str*) – Character to place in a row between the header and data, default to nothing

Returns Nothing. This function executes the print

CiscoAutomationFramework.util.**is_ipv4**(*addr*)

Checks if the IP address provided is an IPv4 Address.

Parameters **addr** (*str*) – IP address to check

Returns True/False

Return type bool

CHAPTER 6

Example Usage

Basic connection to a device:

```
from CiscoAutomationFramework import connect_ssh
ssh = connect_ssh('ipaddress', 'username', 'password')
print(ssh.running_config)
print(ssh.startup_config)
print(ssh.arp_table)
print(ssh.mac_address_table)
ssh.close_connection()
```

You can also use a context manager (recommended) so you dont have to worry about closing the connection:

```
from CiscoAutomationFramework import connect_ssh

with connect_ssh('ip', 'username', 'password') as ssh:
   mac_table = ssh.mac_address_table

print(mac_table)
```

There are integrated output parsers so you can have an easy way of interacting tables that are complex to parse:

```
from CiscoAutomationFramework import connect_ssh
from CiscoAutomationFramework.Parsers.InterfaceStatusParser import␣
↪InterfaceStatusOutputParser

with connect_ssh('ip', 'username', 'password') as ssh:
   output = ssh.send_command_get_output('sh int status')

parser = InterfaceStatusOutputParser(output)
for entry in parser.interfaces:
   print(f'{entry.name} - {entry.vlan} - {entry.status} - {entry.description}')
```

CHAPTER 7

Indices and tables

- genindex
- modindex
- search

Python Module Index

C

Index

L

M

N

O

P