

# Create Python Scripts Utilizing the Realm Library

**Goal**: Create a python script to create stations and Layer-3 cross connects Using the **realm.py** library we will write a script that will allow us to automate the creation of stations and Layer-3 cross connects. We will also be able to start and stop traffic over the cross connects using the script. We will be referencing the script, test\_ipv4\_variable\_time.py, as an example throughout this cookbook. Requires LANforge 5.4.2.

```
1.
```

# Starting the script

- A. Setting up inheritance for our object
  - A. In order for our script to be platform independent we will need to import sys. Then use if 'py-json' not in sys.path:
    - sys.path.append(os.path.join(os.path.abspath('...'), 'py-json'))
  - B. When creating our object we will need to import the LFCliBase module from the LANforge module using from LANforge.lfcli\_base import LFCliBase
  - C. After importing LFCliBase we can create our Class and inherit from LFCliBase

B. Setting up the main method

- A. The main method will typically follow a pattern:
  - I. First, the creation of a list of stations. This can be done in many ways. Example:

```
station_list = LFUtils.port_name_series(prefix_="sta",
    start_id_=0,
    end_id_=4,
    padding_number_=10000)
```

II. Following the station list, we can initialize our object:

```
ip_var_test = IPV4VariableTime(lfjson_host, lfjson_port,
    number_template="00",
    sta_list=station_list,
    name_prefix="var_time",
    ssid="testNet",
    password="testPass",
    resource=1,
    security="wpa2",
    test_duration="5m",
    side_a_min_rate=256,
    side_b_min_rate=256)
```

- III. After our object has been initialized we can begin the testing process. The preferred order for running our tests is to:
  - i. Call **cleanup()** to prevent stations, cross-connects, and endpoints within our list from having creation issues if anything exists with the same name.
  - ii. Call the **build()** method in our class to setup the basic versions of the stations, cross-connects, and endpoints.
  - iii. Call the **start()** method that will start the test itself, as well as any bring up any stations and start traffic on cross-connects that need it.
  - iv. Call the stop() method to stop the traffic and bring down any stations that are up.
  - v. Verify that the tests passed using our inherited passes() method.
  - vi. After verifying a pass we can then call our cleanup function again to clean up everything we worked with.

C. Example Main Method

```
def main():
    lfjson_host = "localhost"
    lfjson_port = 8080
    station_list = LFUtils.portNameSeries(prefix_="sta", start_id_=0, end_id_=4, paddin
    ip_var_test = IPV4VariableTime(lfjson_host, lfjson_port, number_template="00", sta_
       name_prefix="var_time",
        ssid="testNet",
        password="testPass",
        resource=1,
        security="wpa2", test_duration="5m",
        side_a_min_rate=256, side_b_min_rate=256),
    ip_var_test.cleanup(station_list)
    ip_var_test.build()
    if not ip_var_test.passes():
        print(ip_var_test.get_fail_message())
        exit(1)
    ip_var_test.start(False, False)
    ip_var_test.stop()
    if not ip_var_test.passes():
        print(ip_var_test.get_fail_message())
        exit(1)
    time.sleep(30)
    ip_var_test.cleanup(station_list)
    if ip_var_test.passes():
        print("Full test passed, all connections increased rx bytes")
                                                                                       Þ
```

### 2.

### **Test Methods Available With Realm**

- A. Using lfcli\_base.\_pass() and lfcli\_base.\_fail()
  - A. Since our class is inheriting lfcli\_base.py, we have access to methods that will help us keep track of passes and fails during our tests. We can access them using self.\_pass() or self.\_fail(). They will take two parameters, a string message and an optional boolean print\_pass and print\_fail for \_pass() and \_fail() respectively. If print\_pass or print\_fail are set to True, they will write the message to stdout whenever the functions are called.
  - B. Ifcli\_base will add a "PASSED: message" or "FAILED: message" to a list when the tests pass or fail. This list can be accessed using the methods

```
get_result_list()
get_failed_result_list()
get_fail_message()
get_all_message()
```

- B. Using Ifcli\_base to check test success
  - A. passes() will return a boolean depending on whether or not there were any fails in the test. If it finds a fail message it will return False, if none are found it will return True.
     get\_result\_list() will return all logged pass/fail messages as a list.
     get\_failed\_result\_list() will return a list of only fail messages.
    - get\_fai1\_message() will return a list of string of fail messages separated by newlines

get\_message() will return a list of string of all messages separated by newlines

### **Building a Station**

#### A. Build Method

- A. We will need to do a number of things to setup our build method.
  - To begin we will set the security type of our stations using station\_profile.use\_security()
  - II. We will then use station\_profile.set\_number\_template() to name our stations
  - III. After this we can set our command flags and parameters using

```
self.station_profile.set_command_flag("add_sta","create_admin_down",1)
self.station_profile.set_command_param("set_port","report_timer",1500)
self.station_profile.set_command_flag("set_port","rpt_timer", 1)
```

IV. Once our parameters and flags are set, we can pass a list of stations to station\_profile.create() and cx\_profile.create(). Our build function could

```
look like this:
for station in range(len(self.sta_list)):
    temp_sta_list.append(str(self.resource)+"."+self.sta_list[station])
self.station_profile.create(resource=1, radio="wiphy0", sta_names_=self.sta_list, debug=Fal
self.cx_profile.create(endp_type="lf_udp", side_a=temp_sta_list, side_b="1.eth1", sleep_tim
self._pass("PASS: Station build finished")
```

The naming convention for the sides will look like **foo-A** for side\_a and **foo-B** for side\_b. *foo* will be set based on the names in the list of stations given.

#### B. StationProfile

- A. The preferred method for creating a station\_profile is to use the factory methoc new\_station\_profile() found in realm
  - I. We will need to assign some variables for the creation of our stations before we can call create().
    - i. self.station\_profile.use\_security(security\_type, ssid, passwd) is the preferred method to use when setting the security type, ssid, and password variables

Example:

self.station profile.use security("wpa2", "testNet", "testPass")

ii. self.station\_profile.number\_template\_ is the numerical prefix for stations. Using a number\_template of "00" will have stations look like sta01, sta02...sta10 Example: self.station profile.number template ="00"

beii.beacion\_pioirie.namber\_cempiace\_ 00

 iii. self.station\_profile.mode determines the wifi mode used by the stations. See here for available modes Example:

self.station\_profile.mode=0

#### 4.

### **Cross Connects**

- A. Starting and Stopping Traffic
  - A. In order for us to be able to start traffic, our stations will need to be admined up, associated, and with an IP. We can bring them up using station\_profile.admin\_up(). We can then use realm.wait\_for\_ip(resource, sta\_list) to wait for our stations, as well as eth1, to get an IP address.
  - B. Once we are sure all of our stations have ip addresses, we can use cx\_profile.start\_cx() to start the traffic for our cross-connects. When we decide to stop the traffic we can just as easily use cx\_profile.stop\_cx() to stop traffic.

### B. L3CXProfile

- A. self.local\_realm.create\_new\_13\_cx\_profile() is the preferred method for creating a new Layer 3 CX Profile.
  - I. We will need to assign some variables for the creation of our stations before we can call create().
    - i. self.cx\_profile.name\_prefix will be used to specify the name prefix for the cx. Assigning self.cx\_profile.name\_prefix to "test\_" would produce cross-connects named test\_sta00 with the numbers being dependent on station\_profile's number\_template.
       Example:

self.cx profile.name prefix="test "

 Set the \_min\_bps to the desired amount. \_max\_bps can be set but typically defaults to 0 which sets it to the same as the minimum bps. Example:

```
self.cx_profile.side_a_min_bps=56000
self.cx profile.side b min bps=56000
```

### 5.

### Using TTLS

- A. TTLS setup requires a few pieces of information to work correctly. StationProfile has a set\_wifi\_extra() method for setting the relevant variables. See here for the available options
- B. We will need a key management type (key\_mgmt), an EAP method (eap), an EAP identity string (identity), an EAP password string (passwd), an 802.11u realm (realm), an 802.11u domain (domain), and an 802.11u HESSID (hessid)

```
Example:

key_mgmt="WPA-EAP"

eap="TTLS"

identity="testuser"

passwd="testpasswd"

realm="localhost.localdomain"

domain="localhost.localdomain"

hessid="00:00:00:00:00:1"

We can then use these variables to call the set_wifi_extra() method

Example:
```

station\_profile.set\_wifi\_extra(key\_mgmt, eap, identity, passwd, realm, domain, hessid)

### 6.

# **Cleaning Up**

- A. Cleanup stations and cross connects
  - A. We have two options for cleaning up everything once we finish:
    - I. The preferred method to cleanup is to use the individual cleanup methods found in StationProfile and L3CXProfile. These are

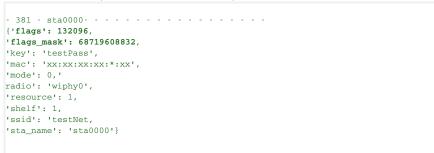
station\_profile.cleanup(resource, desired\_station\_list) and

**cx\_profile.cleanup()**. These methods are preferred because they will only delete stations, cross-connects, and endpoints created during the test while leaving others untouched. This is useful if you are running other scripts in the background.

II. The other method for cleanup is to use Realm's remove\_all\_stations(), remove\_all\_endps(), and remove\_all\_cxs() methods. These will remove all stations, cxs, and endpoints that exist. These are good for doing a full cleanup, and it is recommended to use them in the order of cx, endpoint, station to prevent potential issues or missed deletions.

## **Debugging Stations**

- A. Debug information for station creation can be output by setting debug on=True in
  - StationProfile.create()
    - A. There are a few important debug outputs to pay attention to:
      - I. This is the debug output that appears when using the add\_sta command. This is used frequently in StationProfile.create(). This debug output will allow you to troubleshoot any flags or other information that is being set when creating your stations. It will output the name at the top and the raw JSON data will follow.



II. The next bit of debugging output comes from using the set\_port command. We are able to see all of the JSON data that is posted, and can use this to check our flags and other info.



- B. There are a few steps we can take to make validating the information we get through debugging easier.
  - A. We can use the help page available on the address of the machine LANforge is running onhttp://127.0.0.1/help/ will take us to a page containing all of the commands we can get help with.

LANforge CLI Help - Mozil	la Firefox
LANforge CLI Help	× +
← → ♂ ✿ 0	i 127.0.0.1:8080/help/
<u>add_arm_endp</u>	<u>CLI Reference for /help/add_arm_endp</u>
<u>add_bgp_peer</u>	CLI Reference for /help/add_bgp_peer
add_bond	CLI Reference for /help/add_bond
add_br	<u>CLI Reference for /help/add_br</u>
add_cd	<u>CLI Reference for /help/add_cd</u>
<u>add_cd_endp</u>	<u>CLI Reference for /help/add_cd_endp</u>
<u>add_cd_vr</u>	<u>CLI Reference for /help/add_cd_vr</u>
<u>add_chamber</u>	<u>CLI Reference for /help/add_chamber</u>
<u>add_chamber_cx</u>	<u>CLI Reference for /help/add_chamber_cx</u>
<u>add_chamber_path</u>	<u>CLI Reference for /help/add_chamber_path</u>
<u>add_channel_group</u>	<u>CLI Reference for /help/add_channel_group</u>
add_cx	<u>CLI Reference for /help/add_cx</u>
<u>add_dut</u>	<u>CLI Reference for /help/add_dut</u>
<u>add_dut_notes</u>	<u>CLI Reference for /help/add_dut_notes</u>
<u>add_endp</u>	<u>CLI Reference for /help/add_endp</u>
add_event	<u>CLI Reference for /help/add_event</u>

B. Using http://127.0.0.1/help/add\_sta will bring us to a page specific to the add\_sta command.

LANforge CLI Help - Mo	zilla Fir	efox					•••
LANforge CLI Help	×	+					
← → ♂ ଢ	<b>0</b> ()	<b>127.0.0.1</b> :8080/help/add_sta			∭\ 🖸	9 🙂	<b>#</b> =
		<b>Command Compos</b>	er [add	_sta]			
These are the curl com	mands	:					
echo "" > /tmp/curl_dat curl -sqv -H "Accept: a		tion/json" -X POST -d '@/tmp/curl_data' htt	tp://ctlt2-logan	:8080/cli-form,	/add_sta		
This is the JSON version	on:						
echo "}" > /tmp/json_da curl -sqv -H "Accept: a		tion/json" -H "Content-type: application/js	son" -X POST -d	'@/tmp/json_da	ata' http:	//ctlt2-l	.ogan : 808
This is the CLI comman	nd:						
Parse Command							

C. Here we can enter all of the data we got from our debugging output into the correct areas.

Fields for the co	ommand will update when you change them:	Flag Fields for command will be computed when you select them, but you might need to values into some fields (when you see token values like [string] or [name]).
01: shelf	1	flags.80211r_pmska_cache flags.80211u_additional
02: resource	1	flags.80211u_auto flags.80211u_e911
03: radio	wiphy0	flags.80211u_e911_unauth flags.80211u_enable flags.80211u_ew
04: sta_name	sta0000	flags.so21r_radius flags.create admin down
05: flags	132096	flags.custom_conf flags.disable_fast_reauth
06: ssid	testNet	flags.disable_gdaf flags.disable_ht80
07: nickname	NA	flags.disable_roam flags.disable_sgi flags.hs20 enable
08: key	testPass	flags.ht40_enable flags.ht40_disable
09: ap	NA	flags.ibss_mode flags.lf_sta_migrate
10: wpa_cfg_file	NA	flags.mesh_mode flags.no-supp-op-class-ie
11: mac	xx:xx:xx:*:xx	flags.osen_enable flags.passive_scan flags.power save enable
12: mode	0	
13: rate	NA	flags use wead
14: max_amsdu	NA	flags.wds-mode flags.wep_enable
15: ampdu_factor	NA	flags.wpa2_enable flags.wpa2_enable
16: ampdu_densit	y NA	mode.802.11a mode.AUTO mode.abg
17: sta_br_ip	NA	mode.abgn mode.abgn
18: flags_mask	68719608832	mode.abgnAX mode.an
19: ieee80211w	NA	mode.anAC mode.anAX
20: x_coord	NA	mode.bg mode.bgn
21: y_coord	NA	mode.bgnAC mode.bgnAX
22: z_coord	NA	mode.g rate./a/g

D. Flag fields have a button next to them that will calculate and highlight relevant flags in the right hand column of the page. This can be useful for checking that the correct flags are being set.

ields for the cor	nmand will update when you change them:		Flag Fields for command will be computed when you select them, but you might need values into some fields (when you see token values like [string] or [name]).
01: shelf			flags.80211r_pmska_cache
	1		flags.80211u additional
02: resource			flags.80211u auto
2. 10300100	1		flags.80211u_e911
			flags.80211u e911 unauth
)3: radio			flags.80211u_enable
	wiphy0		- flags.80211u_gw
04: sta_name			
	sta0000		flags.8021x_radius
05.0			flags.create_admin_down
05: flags	132096	*	flags.custom_conf
		07	flags.disable_fast_reauth
06: ssid			flags.disable_gdaf
	testNet		flags.disable_ht80
7. sisters			flags.disable_roam
)7: nickname	NA		flags.disable_sgi
			flags.hs20_enable
08: key			
	testPass		flags.ht160_enable
20.			flags.ht40_disable
09: ap	NA		flags.ibss_mode
	175		flags.lf_sta_migrate
10: wpa_cfg_file			flags.mesh mode
	NA		flags.no-supp-op-class-ie
			flags.osen enable
11: mac	xx:xx:xx:*:xx		flags.passive scan
			flags.power save enable
12: mode			
	0	89	flags.scan_ssid
12. sets			flags.txo-enable
13: rate	NA	*	flags.use-wpa3
		00	flags.verbose
14: max_amsdu			flags.wds-mode
-	NA		flags.wep enable
15. america factor			flags.wpa2 enable
15: ampdu_factor	NA		flags.wpaz_enable
			mode.802.11a
16: ampdu_density			
	NA		mode.AUTO
17: sta br in			mode.abg
17: sta_br_ip	NA		mode.abgn
			mode.abgnAC
18: flags_mask			mode.abgnAX
	68719608832		mode.an
19: ieee80211w			mode.anAC
13. 166600511M	NA		mode.anAX
			mode.b
20: x_coord	[]		
	NA		mode.bg
11: v. operd			mode.bgn
21: y_coord	NA		mode.bgnAC
			mode.bgnAX
22: z_coord			mode.g
-	NA		rate./a/g
			rate./b

E. After we have done this, we can click the **parse command** button towards the top of the data inputs. We can then enter this command into LANforge's messages tab in the input box.

Command Composer [add_sta]
These are the curl commands:
echo "shelf=l&resource=l&radio=wiphy86sta_name=sta80006flags=1320066ssid=testNet&key=testPass&nac=xx:xx::xx::xx:=xx&node=06flags_mask=68719608832" > /tmp/curl_data curl -sqv -H "Accept: application/json" -X POST -d '@/tmp/curl_data' http://ctlt2-logan:8080/cli-form/add_sta
This is the JSON version:
echo "{'shelf':1,'resource':1,'radio':'wiphy0','sta_name':'sta0000','flags':122006,'ssid':'testNet','key':'testPass','mac':'xx:xx:xx:xx:','mode':0,'flags_mask':68719608832}" > /tmp/json_data curl -sqv -H "Accept: application/json" -H "Content-type: application/json" -X POST -d '@/tmp/json_data' http://ctlt2-logan:8080/cli-json/add_sta
This is the CLI command:
l 1 wiphyd staddod 132096 testNet Na testPass Na Na xx:xx:xx:*:xx 0 Na Na Na Na Na 60719608822 Na Na Na Na
Parse Command

Candela Technologies, Inc., 2417 Main Street, Suite 201, Ferndale, WA 98248, USA www.candelatech.com | sales@candelatech.com | +1.360.380.1618