

Querying the LANforge JSON API using Python

Goal: Use Python scripts to query the LANforge Client JSON API. (See Querying the LANforge GUI for JSON Data) The provided Python scripts allow you the same API scope as the Perl scripts.

LANforge now provides Python scripts that query the REST API that the LANforge Client now exposes by default. This chapter steps through using each of the scripts. At the end we show an example of how to write a Python script. Scripts encourage Python 3. Requires LANforge 5.4.1 or later.

Client Settings

On your LANforge Server, the scripts directory is located at `/home/lanforge/scripts`. Under that directory, the `py-json` directory contains Python scripts.

Script	Purpose
<code>__init__.py</code>	Defines the py-json module
<code>show_ports.py</code>	Simplest example of querying all ports. This is typical of querying any of the APIs you browse from <code>/</code> or <code>/help</code>
<code>create_sta.py</code>	Script creates a virtual station
<code>LANforge/__init__.py</code>	Defines the py-json/LANforge module
<code>LANforge/LFRequest.py</code>	Use the LFRequest module to help make GET and POST requests to the LANforge Client
<code>LANforge/LFUfils.py</code>	Use the LFUfils module to help process JSON results

Getting Started

First, start the LANforge Client (LANforge GUI) and connect it to your LANforge Server. If you want to start the client in headless mode, open a terminal, and from the `LANforgeGUL_5.4.1` directory, start the script with the `-daemon` argument:

```
$. ./lfclient -daemon -s localhost
```

Querying Ports

Running the script

In the `/home/lanforge/scripts/py-json` directory:

```
$ python3 ./show_ports.py
{ '1.1.eth0': { '_links': '/port/1/1/0',
               'alias': 'eth0',
               'phantom': False,
               'port': '1.1.00'}}
{ '1.1.eth1': { '_links': '/port/1/1/1',
               'alias': 'eth1',
               'phantom': False,
               'port': '1.1.01'}}
{ '1.1.sta00500': { '_links': '/port/1/1/9',
                  'alias': 'sta00500',
                  'phantom': False,
                  'port': '1.1.09'}}
{ '1.1.sta00501': { '_links': '/port/1/1/10',
                  'alias': 'sta00501',
                  'phantom': False,
                  'port': '1.1.10'}}
{ '1.1.sta00502': { '_links': '/port/1/1/11',
                  'alias': 'sta00502',
                  'phantom': False,
                  'port': '1.1.11'}}
```

Looking inside the script

This script is a way of pretty-printing the results of `GET http://localhost:8080/port/1/1/list`

```
lf_r = LfRequest.LfRequest("http://localhost:8080/port/1/1/list")
json_response = lf_r.getAsJson()
j_printer = pprint.PrettyPrinter(indent=2)
for record in json_response['interfaces']:
    j_printer.pprint(record)
```

Other variations of this you can try are:

/port/list

This is an abbreviation

/port/1/2/list

If you have a second LANforge resource

/port/1/2/eth0

Show a specific port

Example of Creating a Station

Running the script

```
[lanforge@et524-debbie py-json]$ python3 ./create_sta.py
Example 1: will create stations sta0200,sta0201,sta0202
Ex 1: Checking for station : http://localhost:8080/port/1/1/sta0200
...
Ex 1: Next we create stations...
Ex 1: Next we create station sta0200...
Ex 1: station up sta0200...

Example 2: using port list to find stations
Ex 2: checking for station : sta0220
Ex 2: create station sta0220
Ex 2: set port sta0220

Example 3: bring ports up and down
Ex 3: setting ports up...
Ex 3: setting ports down...
...ports are down
Example 4: Modify stations to mode /a
using add_sta to set sta0200 mode

Example 5: change station encryption from wpa2 to wpa3...
using add_sta to set sta0200 wpa3

Example 7: alter TX power on wiphy0...
```

Looking inside the script

Create a station

```
lf_r = LfRequest.LfRequest(base_url+"/cli-form/add_sta")
```

Flags are a decimal equivalent of a hexadecimal bitfield you can submit as either 0x(hex) or (dec) a helper page is available at http://localhost:8080/help/add_sta

You can watch console output of the LANforge GUI client when you get errors to this command, and you can also watch the websocket output for a response to this command at <ws://localhost:8081>. Use `$ wsdump ws://localhost:8081/` to follow those messages.

Modes are listed at http://localhost/LANforgeDocs-5.4.1/lfcli_ug.html or at https://www.candelatech.com/lfcli_ug.html

The MAC address field is a pattern for creation: entirely random mac addresses do not take advantage of address mask matchin in Ath10k hardware, so we developed this pattern to randomize a section of octets:

XX

keep parent

randomize

chars [0-9a-f]

use this digit

If you get errors like "X is invalid hex character", this indicates a previous `rm_vlan` call has not removed your station yet: you cannot rewrite mac addresses with this call, just create **new** stations.

The `staNewDownStaRequest()` creates a station in the Admin-Down state. This is a good way to efficiently create batches of stations because it defers all the PHY layer activity which takes significant time when you do it in a loop.

```
lf_r.addPostData( LfUtils.staNewDownStaRequest(sta_name, resource_id=resource_id, radio=radio, ssid=ssid, passphr
lf_r.formPost()
sleep(0.05)
```

Sleeping for 50ms is not sufficient to interact with the station, but is a functional minimum to allow the LANforge

to start processing the command; this is a good value to use in a loop that creates stations. Follow with:

```
LFUtils.waitForPortsAppear(resource_id, desired_stations)
```

Set station up

The LANforge API separates STA creation and Ethernet port settings. We need to revisit the stations we create and amend flags to add things like DHCP or ip+gateway, admin-{up,down} for sta_name in desired_stations:

```
lf_r = LRequest.LRequest(base_url+"/cli-json/set_port")
data = LFUtils.portDhcpUpRequest(resource_id, sta_name)
lf_r.addPostData(data)
lf_r.jsonPost()
sleep(0.05)
```

Set station down

```
for sta_name in desired_stations:
    lf_r = LRequest.LRequest(base_url+"/cli-json/set_port")
    lf_r.addPostData(LFUtils.portDownRequest(resource_id, sta_name))
    lf_r.jsonPost()
LFUtils.waitForPortsAdminDown(resource_id, desired_stations)
```

Change station mode

There is not a `set_sta` command. Many LANforge CLI commands do a default **modify** if the entity already exists. This is how we can modify attributes of existing stations. For the mode values, see http://www.candelatech.com/lfcli_ug.php#add_sta

```
for sta_name in desired_stations:
    lf_r = LRequest.LRequest(base_url+"/cli-json/add_sta")
    lf_r.addPostData({
        "shelf": 1,
        "resource": resource_id,
        "radio": radio,
        "sta_name": sta_name,
        "mode": 1, # 802.11a
    })
    lf_r.jsonPost()
    sleep(0.5)
```

Change station protocol

Flags for `add_sta` and `set_port` are actually 64-bit values. When the values in the command below are read by the `/help/add_sta` page, Javascript cannot deal with integers greater than 32-bits long.

```
lf_r = LRequest.LRequest(base_url+"/cli-json/add_sta")
lf_r.addPostData({
    "shelf": 1,
    "resource": resource_id,
    "radio": radio,
    "sta_name": sta_name,
    "mode": 0, # mode AUTO

    # sets use-wpa3
    "flags": 1099511627776,

    # sets interest in use-wpa3, wpa2_enable (becomes zero)
    "flags_mask": 1099511628800
})
print("using add_sta to set %s wpa3"%sta_name)
lf_r.jsonPost()
```

Change radio power on radio wiphy0

Virtual stations do not have individual tx power states. You can set the radio transmit power. See http://www.candelatech.com/lfcli_ug.php#set_wifi_radio. The txpower is set through `iwconfig`, so see **man 8 iwconfig**. Power is in dBm, **auto** or **off**.

Not all flags in a JSON request are actually LANforge CLI parameters. The `suppress_preexec_method` parameter is a meta-flag tells the LANforge client to not check that the port exists before issuing the command. You would use this to expedite a script, because a check-port command is synchronous, not intended to be used in a loop.

```
lf_r = LRequest.LRequest(base_url+"/cli-json/set_wifi_radio")
lf_r.addPostData({
    "shelf": 1,
    "resource": resource_id,
    "radio": radio,
    "mode": NA,
    "txpower": "auto",
    "suppress_preexec_method": "true"
})
lf_r.jsonPost()
```


ct524-debbie:8080/help/set_port

Command Composer [set_port]

This is the curl command:

```
$ echo '' > /tmp/curl_data
$ curl -sqv -H 'Accept: application/json' -X POST -d '@/tmp/curl_data' http://lanforge-srv:8080/cli-form/set_port
```

This is the CLI command:

```
1 1 sta0200 NA NA NA NA 2147483649 NA NA NA NA 75513858 8000 NA NA NA NA NA NA
NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA NA
```

Parse Command

Fields for the command will update when you change them:	Flag Fields for command will be computed when you select them, but you might need to actually write modified values into some fields (when you see token values like [string] or [name]).
01: shelf <input type="text" value="1"/>	cmd_flags.abort_if_scripts
02: resource <input type="text" value="1"/>	cmd_flags.force_MII_probe
03: port <input type="text" value="sta0200"/>	cmd_flags.from_dhcp
04: ip_addr <input type="text" value="NA"/>	cmd_flags.from_user
05: netmask <input type="text" value="NA"/>	cmd_flags.new_gw_probe
06: gateway <input type="text" value="NA"/>	cmd_flags.new_gw_probe_dev
07: cmd_flags <input type="text" value="NA"/>	cmd_flags.no_hw_probe
08: current_flags <input type="text" value="2147483649"/>	cmd_flags.probe_wifi
09: mac <input type="text" value="NA"/>	cmd_flags.reset_transceiver
10: mtu <input type="text" value="NA"/>	cmd_flags.restart_link_neg
11: tx_queue_len <input type="text" value="NA"/>	cmd_flags.skip_port_bounce
12: alias <input type="text" value="NA"/>	cmd_flags.use_pre_ifdown
13: interest <input type="text" value="75513858"/>	current_flags.adv_100bt_fd
14: report_timer <input type="text" value="8000"/>	current_flags.adv_100bt_hd
15: flags2 <input type="text" value="NA"/>	current_flags.adv_10bt_fd
	current_flags.adv_10bt_hd
	current_flags.adv_10g_fd
	current_flags.adv_10g_hd
	current_flags.adv_flow_ctl
	current_flags.auto_neg
	current_flags.aux_mgt
	current_flags.fixed_100bt_fd
	current_flags.fixed_10bt_fd
	current_flags.fixed_10bt_hd
	current_flags.ftp_enabled
	current_flags.gro_enabled
	current_flags.gso_enabled
	current_flags.http_enabled
	current_flags.if_down
	current_flags.ignore_dhcp
	current_flags.ipsec_Client
	current_flags.ipsec_concentrator
	current_flags.lro_enabled
	current_flags.no_dhcp_rel

Using the Scripts on Your Laptop

You can copy the `py-json` directory to your laptop or workstation. You may also use `git` and clone the LANforge-scripts repository: <https://github.com/greearb/lanforge-scripts/>.

i If you make a script on a Windows laptop and copy it back to your LANforge, please run `dos2unix` on the script to change the line-ending characters: `$ dos2unix myscript.py`

Python Module Methods

LFRequest.py

Create a new LFRequest object to help create a request:

```
lf_r = LFRequest.LFRequest(base_url+"/port/1/1/wiphy0")
wiphy0_json = lf_r.getAsJson()
```

Your REST requests are discussed in the [Querying LANforge GUI for JSON Data](#) chapter.

formPost(show_error=true)

This method formats post data as `application/x-www-form-urlencoded` data. There should be no significant difference between this and the `jsonPost()` method.

jsonPost(show_error=true)

This method formats post data as `application/json` data. There should be no significant difference between this and the `formPost()` method.

get(show_error=true)

Use this method to do a GET request with 'Accept: application/json' headers. You get unformatted results.

getAsJson()

Formats the results of `get()` into Objects using `json.loads()`

LFUtils.py

```
def staNewDownStaRequest(sta_name, resource_id=1, radio="wiphy0", flags=ADD_STA_FLAGS_DOWN_WPA2, ssid="",
passphrase="", debug_on=False):
```

For use with `add_sta`. If you don't want to generate mac addresses via patterns (`xx:xx:xx:xx:81:*`) you can

generate octets using `random_hex.pop(0)[2:]` and `gen_mac(parent_radio_mac, octet)` See http://localhost:8080/help/add_sta

def portSetDhcpDownRequest(resource_id, port_name, debug_on=False):

Sets port admin down. See http://localhost:8080/help/set_port

def portDhcpUpRequest(resource_id, port_name, debug_on=False):

Sets port up and to use DHCP. See http://localhost:8080/help/set_port

def portUpRequest(resource_id, port_name, debug_on=False):

Sets port up. See http://localhost:8080/help/set_port

def portDownRequest(resource_id, port_name, debug_on=False):

Sets port down. Does not change the use_dhcp flag See http://localhost:8080/help/set_port

def generateMac(parent_mac, random_octet):

Helps generate a random mac address.

def portNameSeries(prefix="sta", start_id=0, end_id=1, padding_number=10000):

This produces a named series similar to "sta000, sta001, sta002...sta0(end_id)" The padding_number is added to the start and end numbers and the resulting sum has the first digit trimmed, so f(0, 1, 10000) => {0000, 0001}

def generateRandomHex():

Use in conjunction with generateMac()

def portAliasesInList(json_list):

Return reverse map of aliases to port records. Normally, you expect nested records, which is an artifact of some ORM that other customers expect:

```
[
  {
    "1.1.eth0": {
      "alias": "eth0"
    }
  },
  { ... }
]
```

Naturally, this is more difficult to digest. This method returns a more intuitive structure:

```
{
  "eth0" : {
    "1.1.eth0": {
      "alias": "eth0"
    },
  },
  "eth1": {},
  ...
}
```

def findPortEids(resource_id=1, port_names=(), base_url="http://localhost:8080"):

returns PortEID objects matching requested port_names. Use after `set_port`

def waitUntilPortsAdminDown(resource_id=1, port_list=()):

Sleep and query until all ports report admin down. Use after `set_port`

def waitUntilPortsAdminUp(resource_id=1, port_list=()):

Sleep and query until all ports report admin up. Use after `set_port`

def waitUntilPortsDisappear(resource_id=1, port_list=()):

Sleep and query until requested ports have entirely gone away. Use after `rm_vlan`

def waitUntilPortsAppear(resource_id=1, port_list=()):

Sleep and query until requested ports have appeared. Use after `add_sta`