sales@candelatech.com
support@candelatech.com
+1 (360) 380-1618 [PST, GMT -8]

Querying the LANforge GUI for JSON Data

Goal: The LANforge GUI now has an embedded webserver and a headless mode of operation. Read on for how to configure and query the client for JSON formatted data.

Updated 2019-11-21: New features in 5.4.1.

- Some of the CLI API parameter names have changed. Notably: **nc_show_ports** `flags` changed to `probe_flags`. Be aware that older scripts might break on upgrade.

Updated 2018-07-24: New features in 5.3.8.

The LANforge GUI (as of release 5.3.6) can be configured to start an embedded web server that can provide data about ports and layer-3 connections. This service can be queried with with any browser or AJAX connection. We're going to increasingly refer to it as the LANforge **client**. This feature provides these benefits:

- More rapid polling: using CLI scripts to poll ports on the LANforge manager can add stress and contention to the LANforge manager; polling the GUI will not tax your test scenario.
- Expanded array of data: the views found in the GUI, like Port Mgr and Layer-3 tabs, contain synthesized data columns not available through the CLI scripting API. Most of these columns can be returned in JSON format.
- Reduced effort when integrating with third party test libraries: many other testing libraries expect JSON formatted input.
- Web socket delivery of event data allows real-time reporting of interface changes and station scan results. This is also a channel for querying additional diagnostic data.
- There is a /help web page that allows you to build POST commands.
- A headless `-daemon` mode that will run the client without any GUI windows. This requires much less memory and has been queried for weeks at a time without crashing or memory leaks.

Present and potential drawbacks of the JSON feature:

- Actively being developed: the JSON views/schema of the objects is at a demonstation state. URLs and JSON structures have changed in 5.3.8.
- Now no longer possible to create Groovy plugins to add JSON features if you want to use the headless mode. JSON Features are compiled into the LANforge GUI from Java sources.
- In 5.3.8 we have limited the view of ports, have added URLs to post direct CLI commands, and have applied HTML `application/x-www-form-urlencoded` form posting submissions in name/value pairs. There is no `multipart/form-data` JSON submission at this time.

Client Settings

The LANforge GUI is started using a script (`lfclient.bash` or `lfclient.bat`). From a terminal, we call that script with the `-httpd` switch. By default the GUI will listen on port 8080:

```
$ cd /home/lanforge
$ ./lfclient.bash -httpd
```

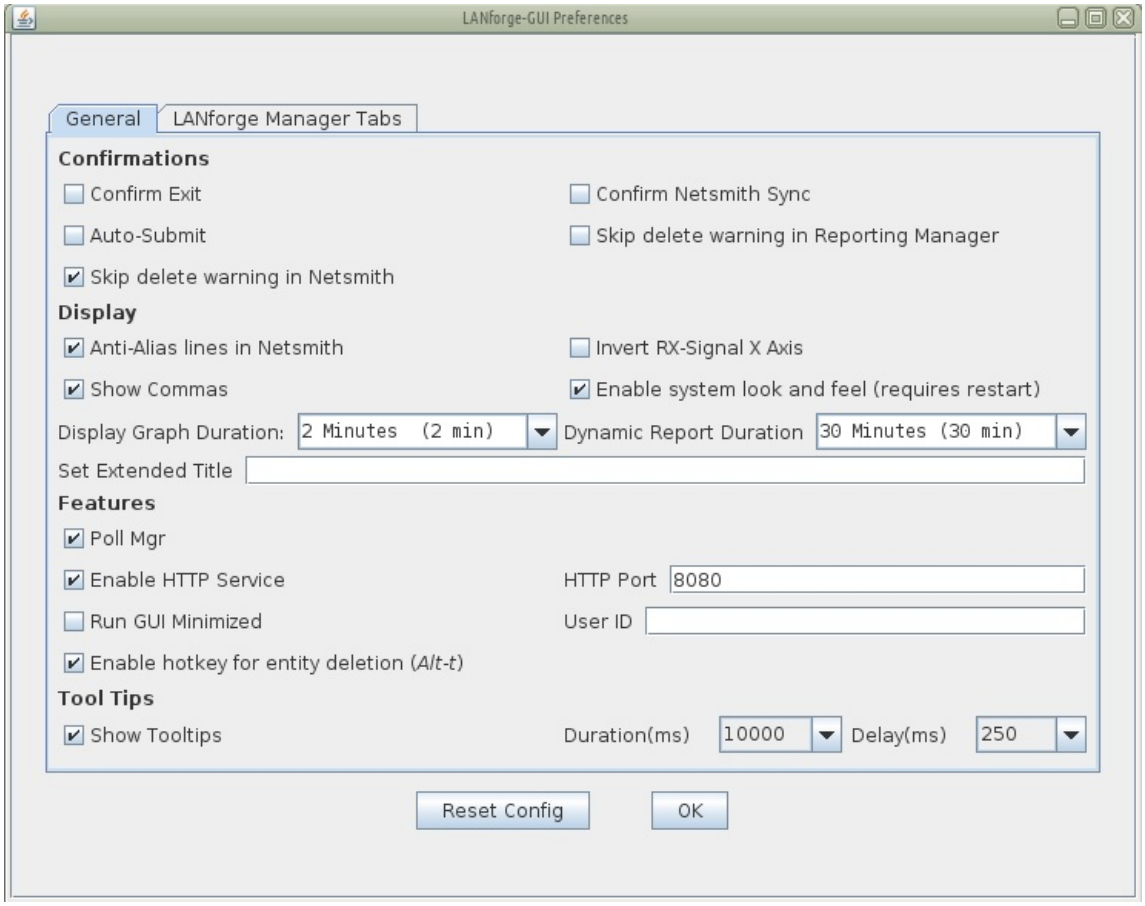
You can specify the port to listen on:

```
$ ./lfclient.bash -httpd 3210
```

You can run the client headless with the `-daemon` switch as well:

```
$ ./lfclient.bash -httpd -daemon
```

There is a setting in the 5.3.8 Control→Preferences menu for setting a minimized mode and the HTTP port number as well.

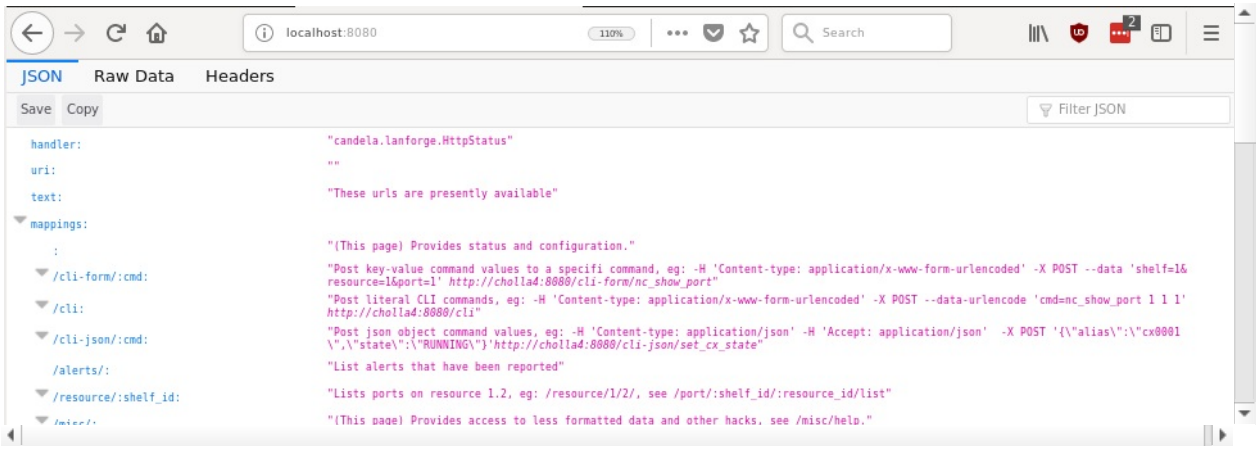


Making Queries

From the terminal we can query the port to find a basic message from the GUI:

```
$ curl -sq http://localhost:8080/
```

This first page (`/`) will give you a JSON list of the resource URLs available. Most URLs will provide JSON as their default content type. Notably, `/help` defaults to HTML.



By default, most URLs will treat a default `Accept: */*` header as `text/html`. Compare the two techniques below:

JSON Output

```
$ curl -sqv -H 'Accept: application/json' http://localhost:8080/resource/1/1
{"handler":"candela.lanforge.HttpResource$JsonResponse","uri":"resource","candela.lanforge.HttpR
```

Clearly, the JSON output is difficult to read. We cover formatting output below.

HTML Output

Most of the queries to the client will return JSON by default. The notable exception is the `/help` URL. To get HTML output in the terminal, you have to specify `Accept: text/html` to curl:

```
$ curl -sqv -H 'Accept: text/html' http://localhost:8080/port/1/1/1
<!DOCTYPE html>
<html>
<head><title>/port</title>
</head>
<body>
<table border='1'><thead><tr><th>EID</th><th>AP</th><th>Activity</th><th>Channel</th><th>Device<
<tbody>
<tr><td>1.1.1</td><td></td><td>0.0</td><td></td><td>eth1</td><td>false</td><td>0.0.0.0</td><td><
</table><hr />
</body>
</html>
```

Formatting Results

JSON formatted text is pretty difficult to read, there are a few different utilities that can help you look at it: `jq, json_pp, json_reformat, tidy, xmllint, yajl` and `jsonlint`.

Example of installing formatters

On Fedora, install:

```
$ sudo dnf install -y jq perl-JSON-PP tidy libxml2 yajl
```

On Ubuntu, install:

```
$ sudo apt install -y jq libjson-pp-perl perltidy xmllint libxml2-utils yajl-tools
```

Now we can perform a query:

```
$ curl -sq /port/1/1/1
{
  "candela.lanforge.HttpPort" : {
     "duration" : "1"
  },
  "handler" : "candela.lanforge.HttpPort$JsonResponse",
  "interface" : {
     "stuff":...
  },
  "uri" : "port/:shelf_id/:resource_id/:port_id"
}
```

Notice that the URI object list paths with colon-tagged positions in them, e.g.: `/cli-form/:cmd`. These are interpreted as URL parameters and not query string parameters, they cannot be moved into the query string.

Making your shell friendly

To save you typing, you might want to add this function to your `.bash_aliases` file:

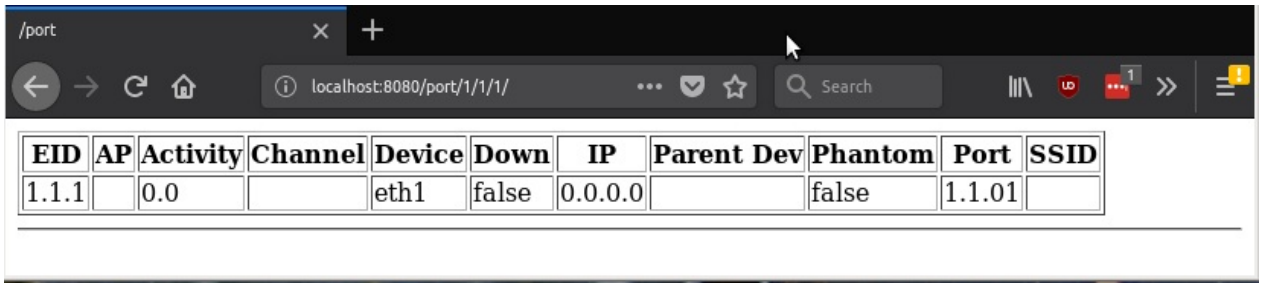
```
function Json() {
   curl -sqv -H 'Accept: application/json' "http://localhost:8080${@}" \
    | json_reformat | less
}
```

Then you can make your calls this way:

```
$ Json /port/1/1/1
```

Browsing results in table format

We can view a URL in a browser as well:



EID	AP	Activity	Channel	Device	Down	IP	Parent Dev	Phantom	Port	SSID
1.1.1		0.0		eth1	false	0.0.0.0		false	1.1.01	

Viewing Alerts and Events

You can both view and stream event data. Querying events and alerts are both quite similar:

```
$ Json /events
{
"handler" : "candela.lanforge.HttpEvents$FixedJsonResponder",
"events" : [
   {
      "2249259" : {
         "event" : "Connect",
         "_links" : "/events/2249259",
         "entity id" : "NA"
      }
   },
   ....
```

A busy LANforge system will generate hundreds of thousands of events. Only the last few thousand can be recalled.

You can inspect a singular event:

```
$ Json /events/2249259
{
    "handler": "candela.lanforge.HttpEvents$FixedJsonResponder",
    "uri": "events/:event_id",
    "candela.lanforge.HttpEvents": {
        "duration": "0"
    },
    "event": {
        "eid": "1.3.21",
        "entity id": "NA",
        "event": "Connect",
        "event description": "sta3106 (phy #1): connected to 00:0e:8e:d5:fa:e6",
        "id": "2249259",
        "name": "sta3106",
        "priority": "  Info",
        "time-stamp": "2018-07-24 14:39:33.776",
        "type": "Port"
    }
}
```

We can view `/alerts` similarly.

```
$ Json /alerts/92
{
   "handler" : "candela.lanforge.HttpEvents$FixedJsonResponder",
   "uri" : "alerts/:event_id",
   "alert" : {
      "name" : "wlan0",
      "time-stamp" : "2018-07-02 16:23:30.880",
      "entity id" : "NA",
      "id" : "92",
      "eid" : "1.1.5",
```

Streaming Events

Continually polling the `/events` URL is not as effective as streaming a websocket providing the same data. We need a web socket client. Websockets are built into modern browsers and there are python and perl utilities for the job as well. An easy to use python client is `wsdump`.

Installing wsdump

There is a useful python utility called `wsdump` (or `wsdump.py`). Try to install the `python-websocket` package to get it. There are many similar matches, but there is not one dedicated package that provides it. On Fedora:

```
root@fedora$ dnf whatprovides `which wsdump`
root@fedora$ dnf install -y python3-websocket-client
```

```
root@ubuntu$ ls -l /usr/bin/wsdump
  /usr/bin/wsdump → /etc/alternatives/wsdump
root@ubuntu$ ls -l /etc/alternatives/wsdump
 /etc/alternatives/wsdump → /usr/bin/python2-wsdump
root@ubuntu$ dpkg-query -S /usr/bin/python2-wsdump
python-websocket: /usr/bin/python2-wsdump

root@ubuntu$ sudo apt install python-websocket
```

You might need to install pip, and that might be in the python3-pip package. Then you can install via:

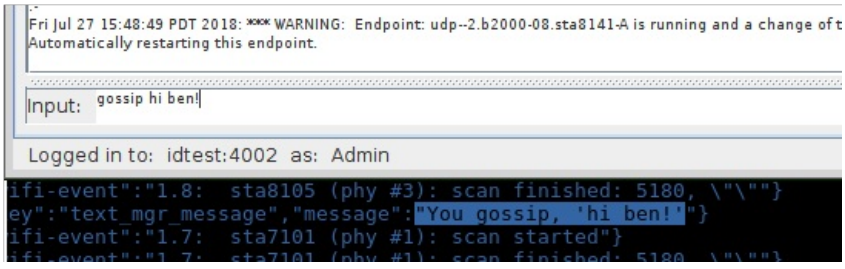
```
$ sudo apt install python-pip # or sudo dnf install python-pip
$ sudo pip install --upgrade pip
$ pip search websocket
$ sudo pip install websocket-client
```

Streaming Using `wsdump`

Here's an example of `wsdump` below. Don't forget you are now using h the `ws://` schema and not the `http://` schema!

```
$ /usr/bin/wsdump ws://localhost:8081/
```

It might take a few second to start showing results if your system is not very active. You should be able to prompt output by executing this message in the **Messages** tab: `gossip hi ben!`



Streaming Using `javascript`

You can also use a web page to follow events because websockets are built into modern browsers. This is a screenshot of the

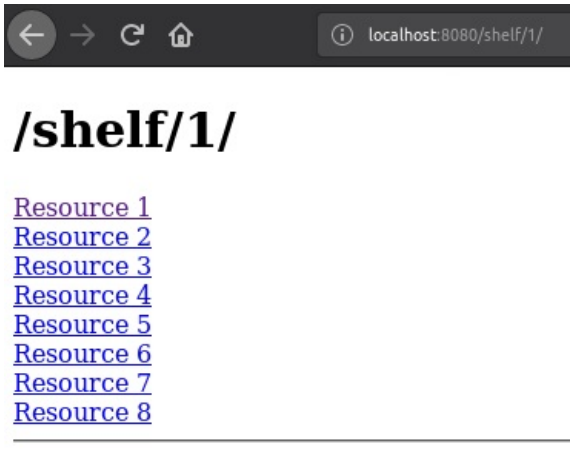
Data Views

URLs

/shelf

The `/shelf/1/` URL provides a list of resources in your realm:

```
$ Json /shelf/1
{
    "handler": "candela.lanforge.HttpResource$JsonResponse",
    "uri": "shelf/:shelf_id",
    "candela.lanforge.HttpResource": {
        "duration": "0"
    },
    "resources": [
        {
            "1.1": {
                "_links": "/resource/1/1",
                "hostname": "idtest.candelatech.com"
            }
        },
        {
            "1.2": {
                "_links": "/resource/1/2",
                "hostname": "hedtest"
            }
        }
   ]
}
```



/resource

The `/resource` URL provides a digest of ports available at the requested resource.

```
$ Json /resource/1/1
{
   "handler" : "candela.lanforge.HttpResource$JsonResponse",
   "resource" : {
      "free swap" : 526332,
      "free mem" : 4634228,
      "load" : 0.4,
      "bps-rx-3s" : 7850,
      "sw version" : "  5.3.8  64bit",
      "entity id" : "NA",
      "tx bytes" : 40533976395,
      "phantom" : false,
      "eid" : "1.1",
      "hostname" : "idtest.candelatech.com",
      "hw version" : "Linux/x86-64",
```

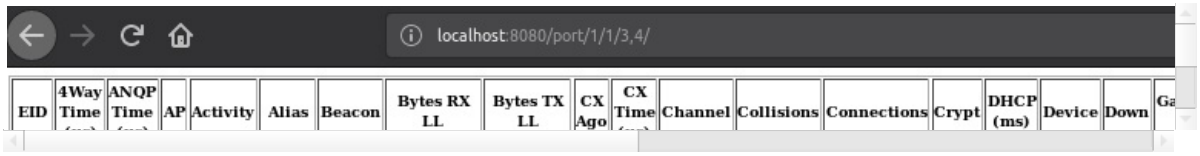
/port

The `/port` URL provides a digest of ports and their state. You can request multiple ports by ID on this resource by appending the port IDs with commas. You can list ports on a resource:

```
$ Json /port/1/5/list
{
   "handler" : "candela.lanforge.HttpPort$JsonResponse",
   "uri" : "port/:shelf_id/:resource_id/:port_id",
   "interfaces" : [
      {
         "1.5.b5000" : {
            "entity id" : "NA",
            "_links" : "/port/1/5/7",
            "alias" : "b5000"
         }
      },
      {
         "1.5.eth0" : {
            "alias" : "eth0",
```

We can query multiple ports at a time by their number or their name by placing a comma between the specifiers. Additionally, we can query for just the fields we desire. All field names are lower-case: `?fields=tx+crr,rx+fifo`.

```
$ Json '/port/1/5/wiphy0,wiphy1?fields=device,phantom,tx+bytes,mode'
{
  "interfaces" : [
     {
        "1.5.wiphy0" : {
           "tx bytes" : 401236186,
           "mode" : "802.11abgn",
           "device" : "wiphy0",
           "phantom" : false
        }
     },
     {
        "1.5.wiphy1" : {
           "phantom" : false,
           "device" : "wiphy1",
```



/cx

The `/cx` URL allows us to query Layer-3 connection information.

```
$ Json /cx
{
   "uri" : "cx",
   "handler" : "candela.lanforge.GenericJsonResponder",
   "connections" : [
      "41.1" : {
         "entity id" : "NA",
         "name" : "udp:r3r2:3000",
         "_links" : "/cx/41"
      },
      "50.1" : {
         "name" : "udp:r3r2:3009",
         "entity id" : "NA",
         "_links" : "/cx/50"
      }
```

And individual connections:

```
$ Json /cx/udp:r3r2:3000$ Json 'cx/udp:r3r2:3000'
{
   "uri" : "cx/:cx_id",
   "41.1" : {
      "drop pkts b" : 0,
      "type" : "LF/UDP",
      "rx drop % a" : 0,
      "rpt timer" : "1000",
      "pkt rx a" : 0,
      "avg rtt" : 0,
      "rx drop % b" : 0,
      "name" : "udp:r3r2:3000",
      "endpoints (a ↔ b)" : "udp:r3r2:3000-A <=> udp:r3r2:3000-B",
      "drop pkts a" : 0,
      "entity id" : "NA",
```

> Technically, colons in URLs need to be encoded as `%3A`, so the above URL should be `/cx/udp%3Ar3r2%3A3000`, but `curl` is pretty darned forgiving.

/endp

Endpoints may be listed and inspected:

```
$ Json /endp/
            {
   "uri" : "endp",
   "handler" : "candela.lanforge.HttpEndp$JsonResponse",
   "candela.lanforge.HttpEndp" : {
      "duration" : "4"
   },
   "endpoint" : [
      {
         "1.2.8.55.2" : {
            "_links" : "/endp/55",
            "entity id" : "NA",
            "name" : "sta3000-ep-B"
         }
      },
```

```
$ Json /endp/sta3000-ep-B
   {
      "candela.lanforge.HttpEndp" : {
         "duration" : "1"
      },
      "uri" : "endp/:endp_id",
      "endpoint" : {
         "rx rate ll" : 0,
         "pdu/s tx" : 0,
         "bursty" : false,
         "rx rate" : 0,
         "tx pkts ll" : 0,
         "rx bytes" : 0,
         "run" : false,
         "tcp rtx" : 0,
```

Creating Ports

It is possible to create ports and connections by using the CLI commands. Your LANforge test scenarios (located in the `/home/lanforge/DB/` directory) contain all the CLI commands that create your ports and connections. You can submit those commands over HTTP in two ways:

- `/cli-json/$command` An example of using the gossip command:

```
curl -X POST -H 'Content-type: application/json' \
               -d '{"message":"hello world"}' http://localhost:8080/cli-json/gossip
```

Then check your LANforge GUI messages.

- `/cli-form/$command` An example of using the gossip command:

  ```
  curl -X POST -d 'message=hello+world' http://localhost:8080/cli/gossip
  ```

  Then check your LANforge GUI messages.

- `/cli/`: use this method to submit a raw URL-encoded command. This might be useful if you are copying commands directly out of a database:

  ```
  curl -X POST -d 'cmd=gossip hello' http://localhost:8080/cli/
  ```

Except for `/cli-json`, these methods accept `application/x-www-form-urlencoded` content type submissions. This is default for the NanoHttp library and default for `curl`.

These CLI commands do not return data, only a result code. All data that the Perl scripts would collect from command line queries is sent directly to the GUI. Some CLI commands send data over the websocket, like the `diag` command.

Command help

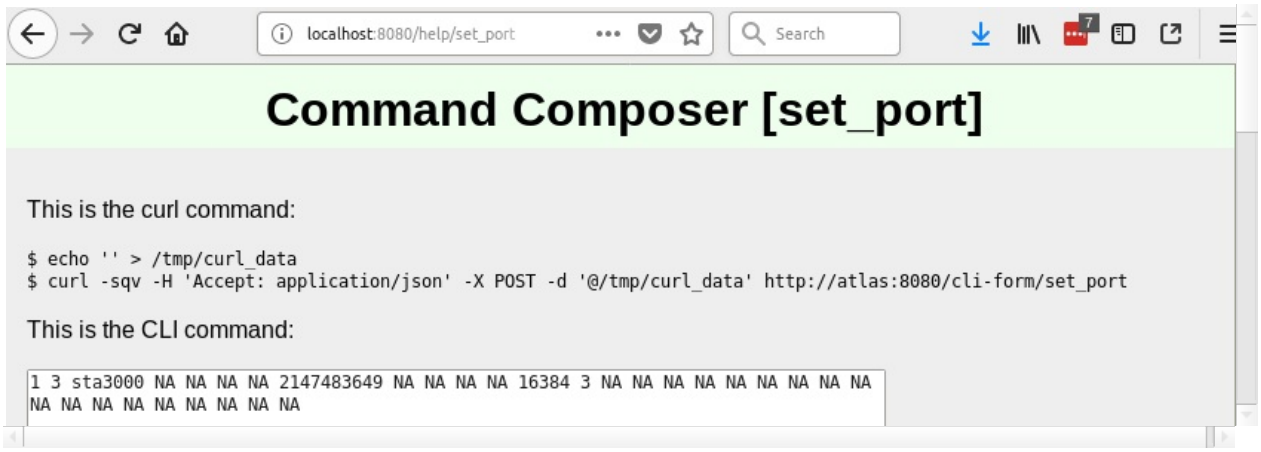
Commands are often complex and include a number of bitwise flags to set the state and features of ports. There is presently no tag-substitution for port flags, but there is a help utility that can help you compute them.

```
http://localhost:8080/help/
```

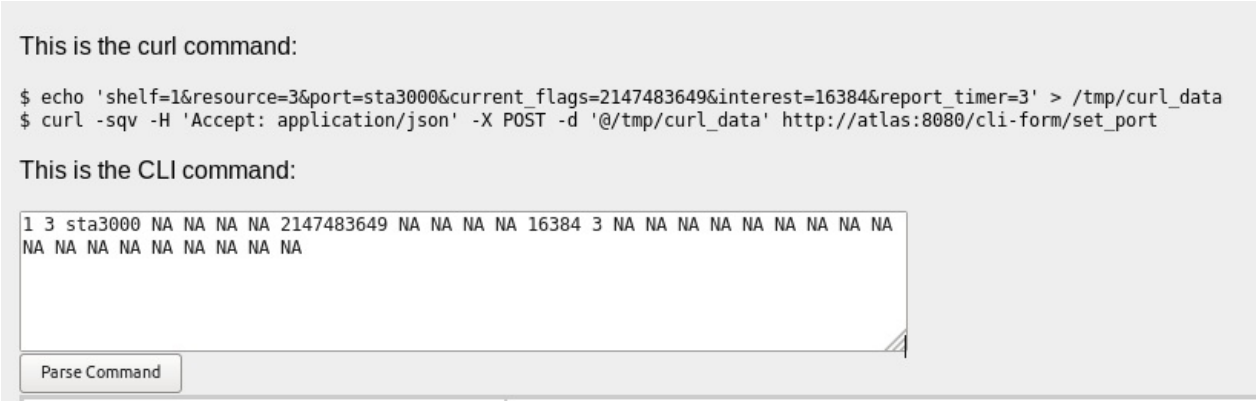
Select a command to see the field helper screen:

```
http://localhost:8080/help/set_port
```

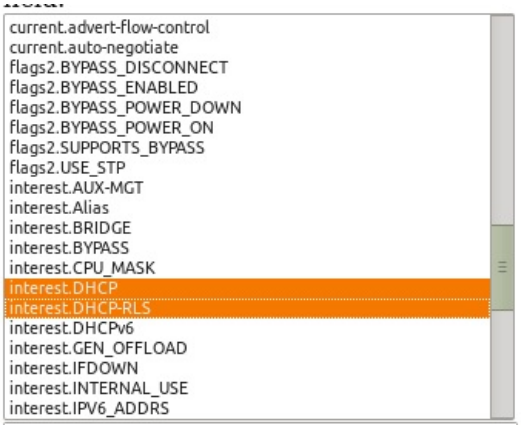
Type values into the field inputs and the CLI command will be refreshed:



Click the **Parse Command** button and the values in the command box will be displayed in the curl command and the field inputs. (Notice this form is doing a GET request.)



You may find a list of flag fields that are organized by field names. The text area below the selection list is the sum of the selected fields. Copy the flag values into the input field above to incorporate it into your command.



Creating a WiFi Station

Please refer to the scripts `lf_associate_ap.pl` and `lf_vue_mod.sh` for examples of how to produce lists of CLI commands involved in creating stations. Please refer to:

1. Learn CLI Commands used to operate WiFi stations
2. and Changing Station WiFi SSID with the CLI API

These will provide ways of collecting the CLI commands in log files for you to place into the command `/help/` page.

- Use ssh to log into your LANforge manager. Use the `lf_vue_mod.sh` script to create a station:

```
$ cd scripts
$ ./lf_vue_mod.sh --mgr localhost --resource 3 --create_sta --name sta3101 \
  --radio wiphy1 --ssid idtest-1000-open --passphrase '[BLANK]' \
  --log_cli /tmp/clilog.txt

$ cat /tmp/clilog.txt
set_wifi_radio 1 3 wiphy1 NA -1 NA NA NA NA NA NA NA NA 0x1 NA
add_sta 1 3 wiphy1 sta3101 1024 idtest-1000-open NA [BLANK] AUTO NA 00:0e:8e:c1:df:45 8 NA NA NA
set_port 1 3 sta3101 0.0.0.0 255.255.0.0 0.0.0.0 NA 2147483648 00:0e:8e:c1:df:45 NA NA NA 840503
```

1. Enter each command into the your browser toolbar by altering the command into a url:

```
http://localhost:8080/help/set_wifi_radio?cli=1 3 wiphy1 NA -1 NA NA NA NA NA NA I
```

Produces:

```
$ echo 'shelf=1&resource=3&radio=wiphy1&channel=-1&flags=0x1' > /tmp/curl_data
$ curl -sqv -H 'Accept: application/json' -X POST -d '@/tmp/curl_data' \
 http://localhost:8080/cli-form/set_wifi_radio
```

```
http://localhost:8080/help/add_sta?cli=1 3 wiphy1 sta3101 1024 idtest-1000-open NA
```

Produces:

```
$ echo 'shelf=1&resource=3&radio=wiphy1&sta_name=sta3101&flags=1024&ssid=idtest-1(
$ curl -sqv -H 'Accept: application/json' -X POST -d '@/tmp/curl_data' \
 http://localhost:8080/cli-form/add_sta
```

```
http://localhost:8080/help/set_port?cli=1 3 sta3101 0.0.0.0 255.255.0.0 0.0.0.0 NA
```

Produces:

```
$ echo 'shelf=1&resource=3&port=sta3101&ip_addr=0.0.0.0&netmask=255.255.0.0&gatewa
$ curl -sqv -H 'Accept: application/json' -X POST -d '@/tmp/curl_data' \
 http://localhost:8080/cli-form/set_port
```

2. Verify with the LANforge GUI the changes you wish to make.

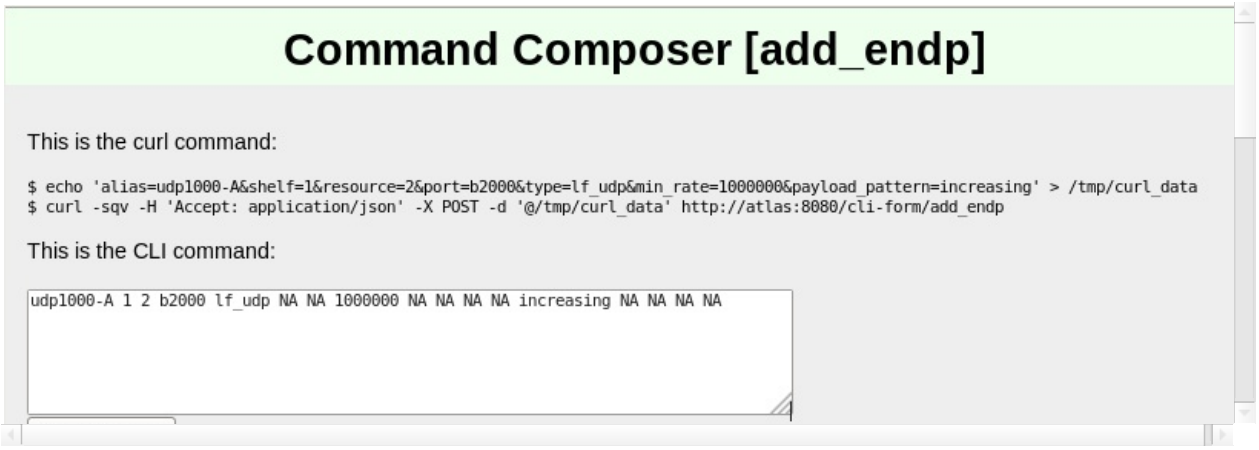
Creating Connections

Using the `/cli-json/add_endp` and `/cli-json/add_cx` URLs, it is possible to create Layer-3 connections. Create the Layer-3 endpoints first, of course.

Create L3 Endpoints

Construct your command using the `/help/add_endp` page. For an example, use these parameters:

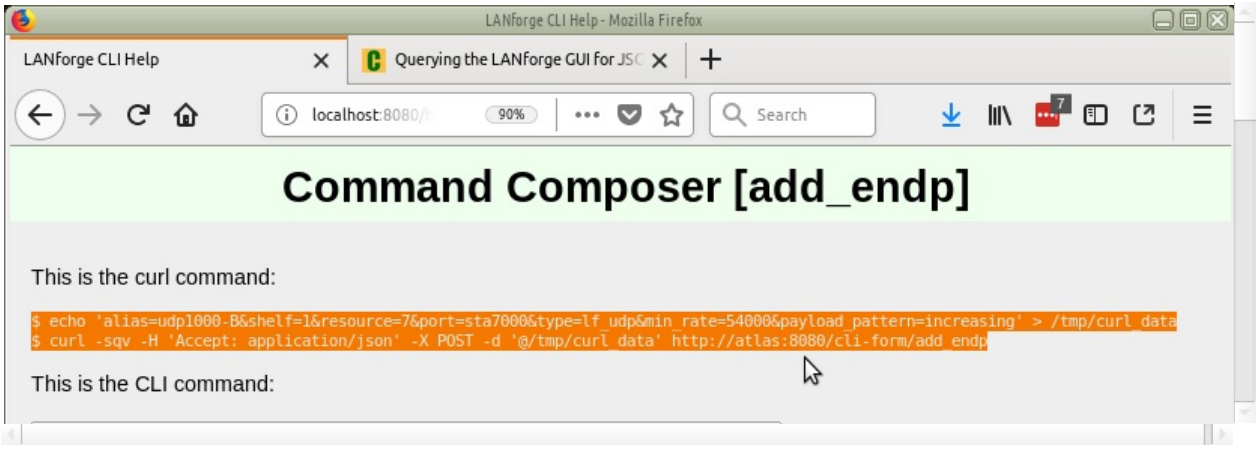
alias	enter udp1000-A
shelf	1
resource	2
port	b2000
type	select **type.lf_udp**
min_rate	1000000 (1 Mbps)
max_rate	SAME
payload_pattern	select **payload_pattern.increasing**



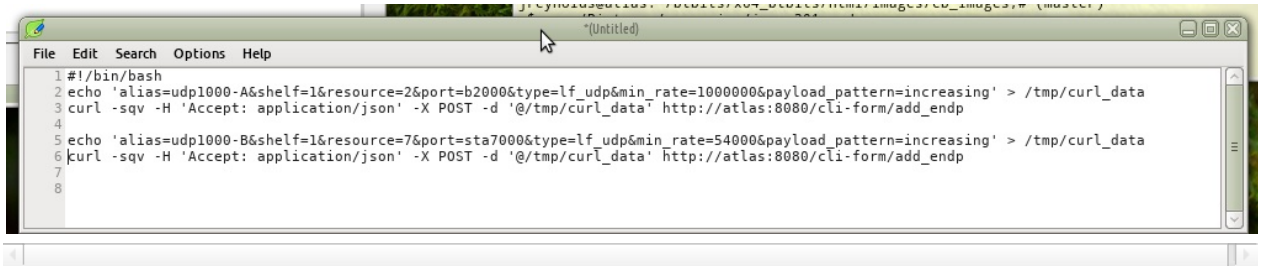
Click **Parse Command** and copy the resulting `curl` command into a text editor:

And for the **B** endpoint, choose a station:

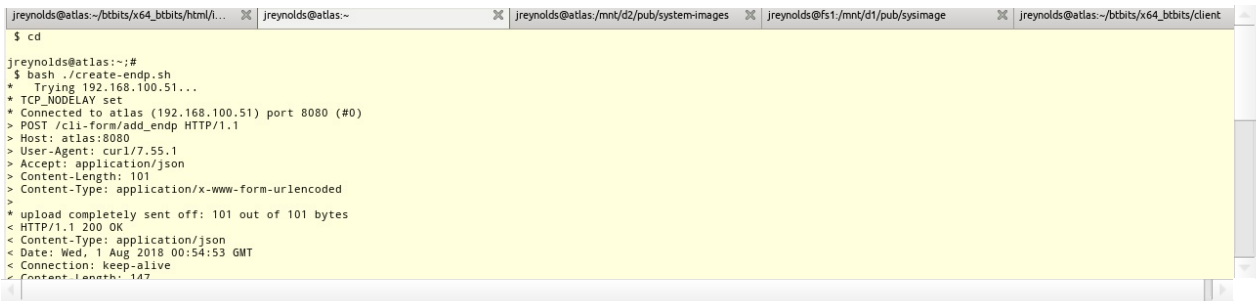
alias	enter udp1000-B
shelf	1
resource	7
port	sta7000
type	select **type.lf_udp**
min_rate	54000 (54 Kbps)
max_rate	SAME
payload_pattern	select **payload_pattern.increasing**



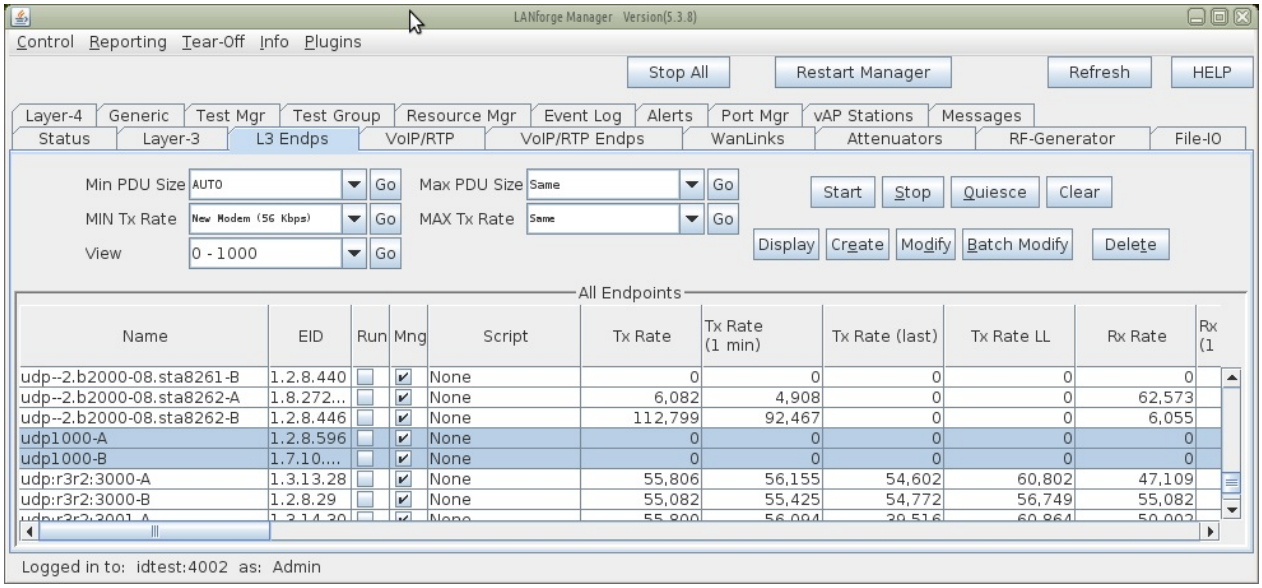
Click **Parse Command** and copy the resulting `curl` command into a text editor:



We'll save this file as a shell script: `~/create-endp.sh` We can then run it from our terminal like so: `bash -x create-endp.sh`

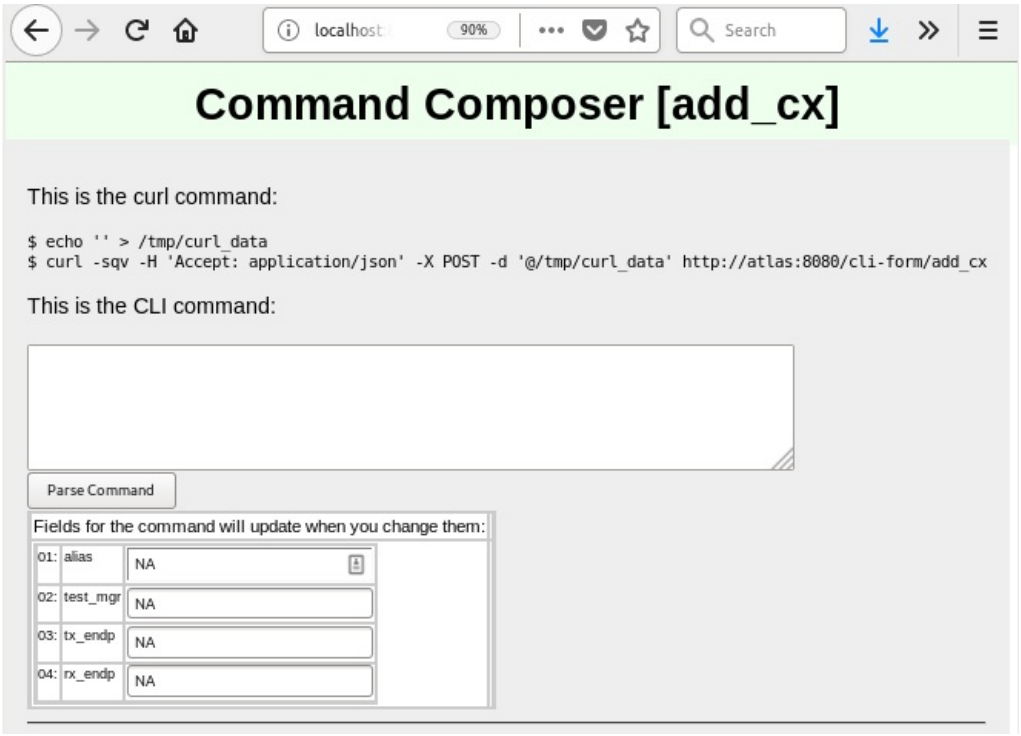


We should see the endpoints we've created in the LANforge GUI **Endps** tab:



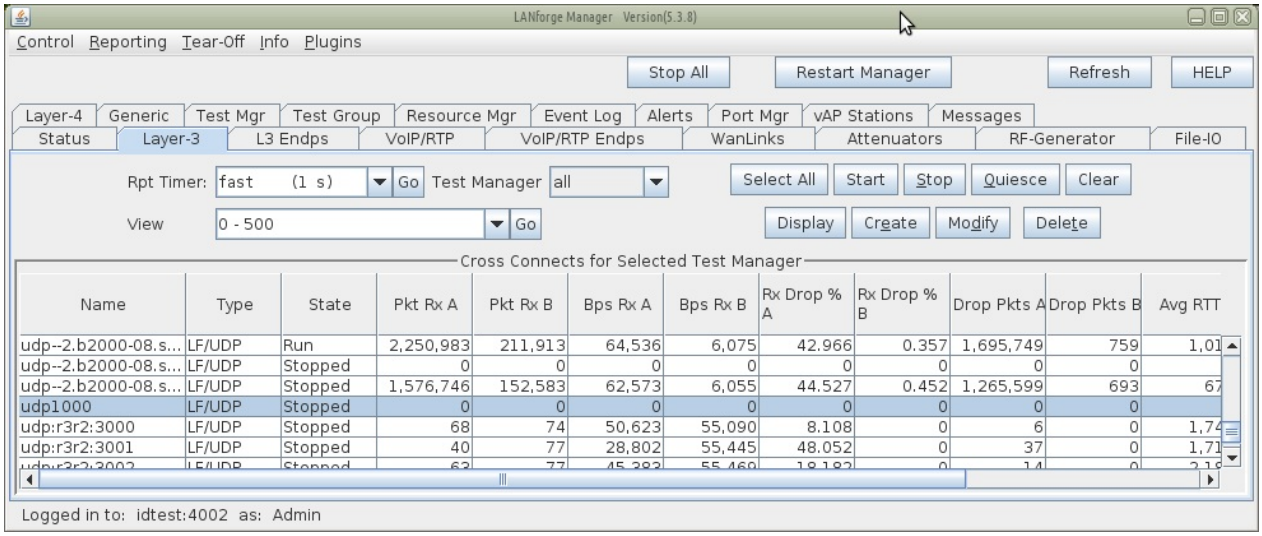
Create L3 Connection

With the creation of two endpoints, we can proceed with creating a Layer 3 cross-connect. This is much simpler, it really only takes the names of the two endpoints we created above. We'll choose `default_tm` for the test manager.



alias	udp1000
test_mgr	default_tm
tx_endp	udp1000-A
rx_endp	udp1000-B

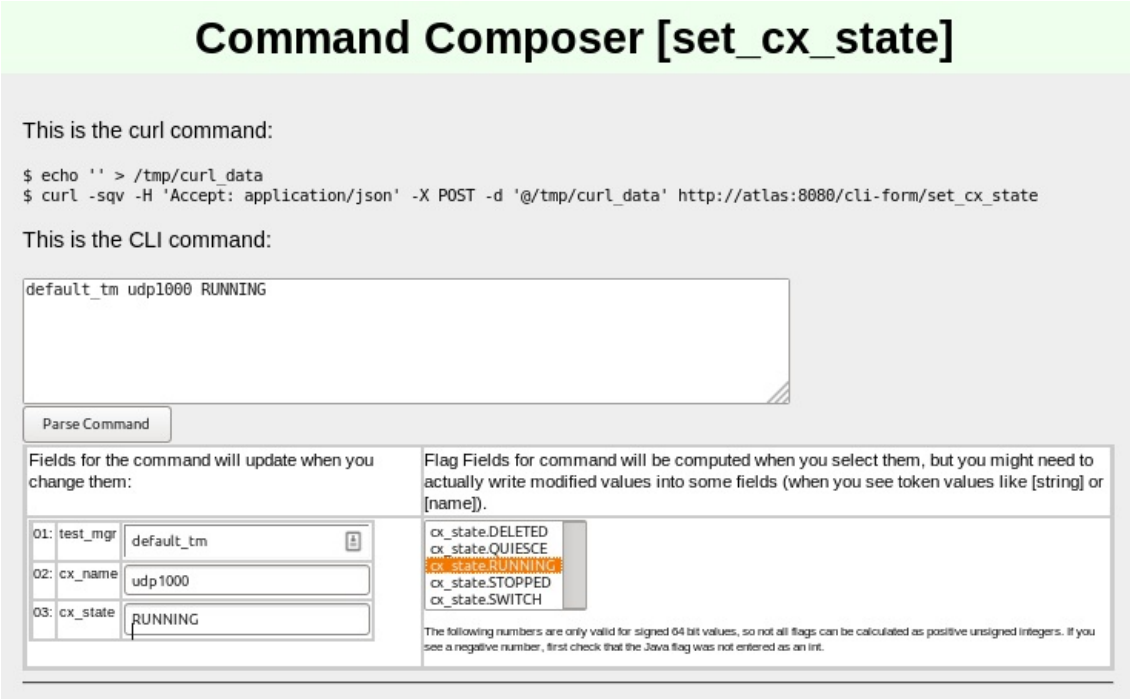
Click the **Parse Command** button and copy the resulting `curl` command into your editor with the shell script. Run the script again. It doesn't hurt to re-create the endpoints.



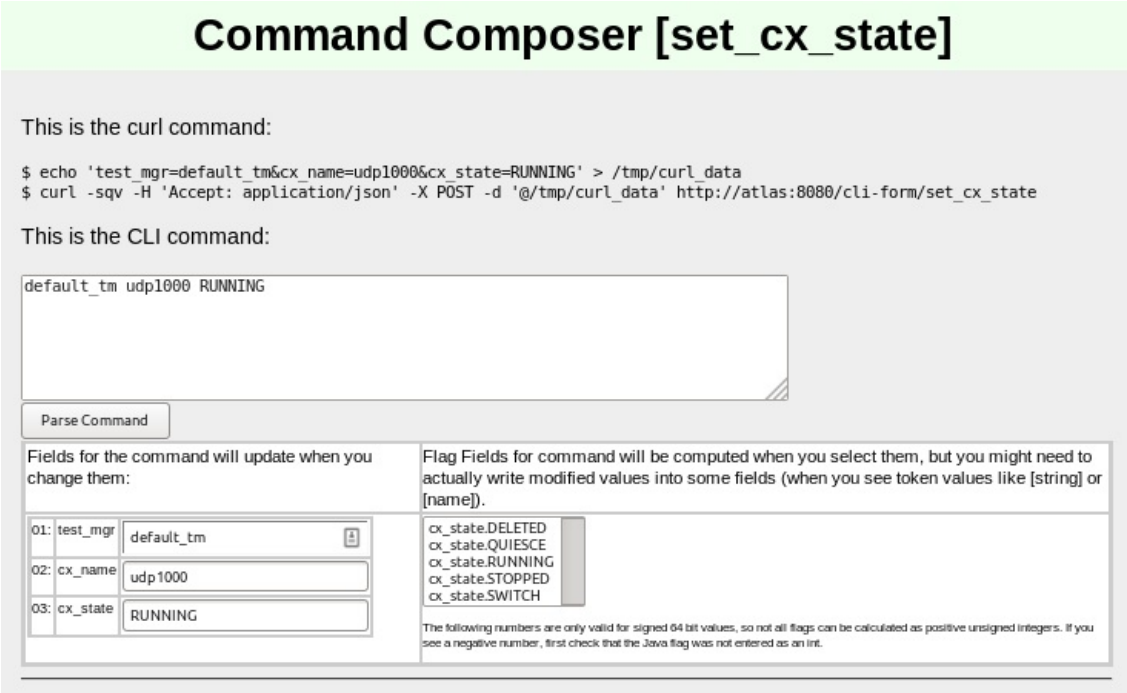
Toggling the Connection

Cross connects have three good state: STOPPED, RUNNING, and QUIESCE. The command to change them is `set_cx_state`. You will have no trouble creating the command:

test_mgr	default_tm
cx_name	udp1000
cx_state	RUNNING



Click **Parse Command** and then you can paste the resulting command into your editor.



Advanced Techniques

You can make JSON submissions and you can also submit Base64 encoded values in both form an and JSON submission URLs.

Submitting Base64

Field names that end in `-64` are interpreted as base64 encoded values. From a linux terminal, you can convert text to base64 encoded value using the `base64` command:

```
$ echo "RUNNING" | base 64
UlVOTklORwo=
```

Below is a CLI command example. **You typically would not care to spend the effort doing this unless the data you need to express is difficult to URL encode.**

```
$ echo 'test_mgr-64=YW55Cg==&cx_name-64=dWRwMTAwMAo=&cx_state-64=UlVOTklORwo=' > /tmp/curl_data
$ curl -A 'Accept: application/json' -X POST -d @/tmp/curl_data http://host/cli-form?set_cx_state
```

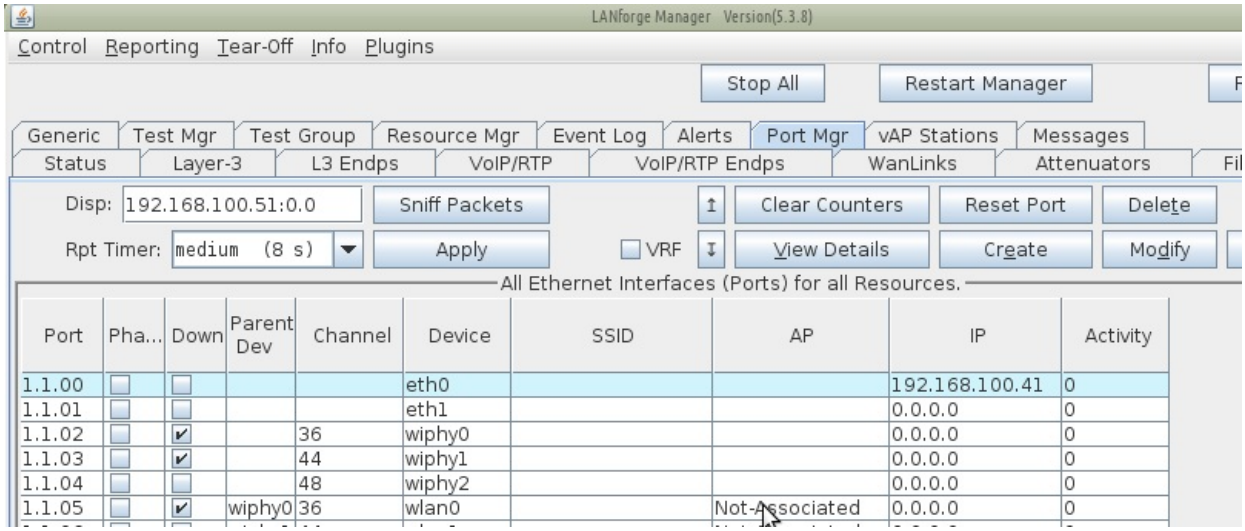
Submitting JSON

Instead of posting to `/cli-form`, you can post to `/cli-json` and your submission will be parsed as a json object. The parameter names stay the same. The base64 name extensions are also available! You **need** to specify that your `Content-type` in the POST is `application/json`.

```
$ echo '{"test_mgr":"default_tm","cx_name":"udp1000","cx_state":"RUNNING"}' > /tmp/curl_data
$ curl -sq -H 'Content-type: application-json' -H 'Accept: application/json' \
  -X POST -d@/tmp/curl_data http://localhost:8080/cli-json/set_cx_state
```

Handling Mismatched Column Errors

(This should be fixed as of 2018/08/14) When the LANforge cliet is in GUI mode, the **columns** of data that are returned match the GUI **table columns** displayed. You can use the Right-click→Add/Remove Table Columns menu item to change this. **We do not recommend doing this** for querying JSON data though, because the table columns definitions will not match up to the data the webserver expects to return.



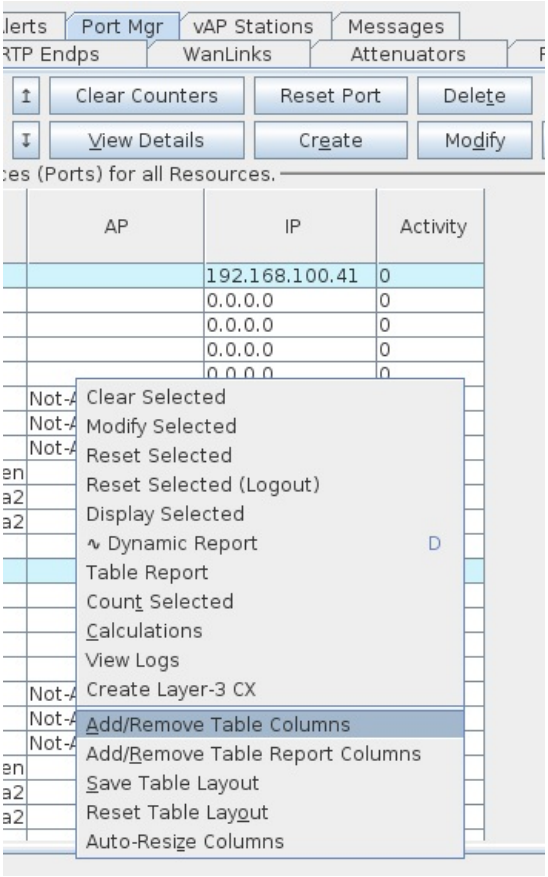

```
$ curl -sq http://localhost:8080/port | json_pp
{
   "error_list" : [
      "names_to_col_ids map is not going to work:\n[tx abort][25 > 4]\n[cx time (us)][48 > 4]\n[
   ],
   "status" : "INTERNAL_ERROR"
}
```

The terminal you started the LANforge client on will also give a similar error:

```
1532480073953:  names_to_col_ids size:71
java.lang.IllegalArgumentException: names_to_col_ids map is not going to work:
1532480073953:  lfj_table columns:10
```

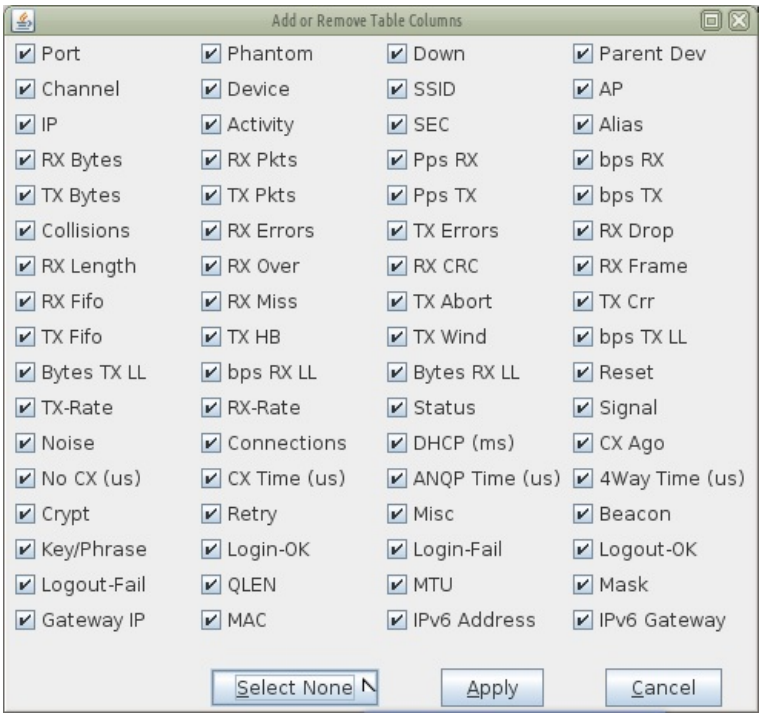
Reset the Table Layout

1. Right-clicking the Port Mgr and selecting **Add/Remove Table Columns** will allow you to change this.

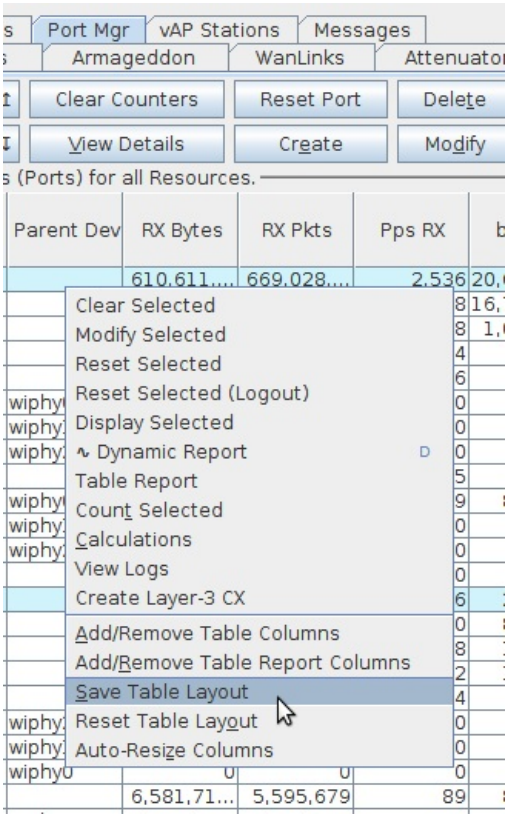


2. Clicking the **Select All/None** button and then **Apply** will get all the columns displayed, and returned in

your queries.



3. Make sure to **Right-Click → Save Table Layout** so that your next session will show all the data.



4. Restart the LANforge client