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

Managing WAN Links Using LANforge JSON API

Goal: Create and modify WAN Links Using LANforge JSON API. This cookbook provides examples in Python. (See Querying the LANforge GUI for JSON Data) The provided Python scripts allow you the same API scope as the Perl scripts.
This chapter steps through using Python scripts to create and manage WAN Links on a LANforge. Scripts require Python 3. Requires LANforge 5.4.1 or later. Examples require CT910, CT521a, or better.

Creating a WAN Link

We will start by creating a WAN Link between ports `eth2` and `eth5` on our ct522. Two Ethernet ports will be involved in this example.

The `create_wanlink.py` Script

This script is located in `/home/lanforge/scripts/py-json` or on the lanforge-scripts github page. You can copy this script to a new name and edit it to fit your enviroment. Remember, these JSON scripts will be querying a LANforge Client (GUI or headless). The URL you will see being queried is going to be `http://localhost:8080/` for these exaples, assuming you are running the LANforge client on the same machine you are running your script.

This script performs the basic tasks you might use to manage WANlink connections:

- Listing existing WANlinks
- Removing a WANlink if it exists.
- Creating WANlink endpoints. You want to create the endpoints before creating the connection.
- Joining WANlink endpoints into a WANlink connection (CX)
- Starting the WANlink
- Modifying the WANlink. You can set endpoint tx rates and lossiness parameters while the endpoints are running.
- Stopping the WANlink
- Listing WANlink endpoint stats

Script Sections Explained

1. **Listing Wanlinks**
   Notice that when we get a listing response, we are looking for items in the response that are dictionarys with a `_links` key/value pair. There are other key/values used for diagnostics, such as *uri, handler, warnings and errors*.

```
base_url = "http://localhost:8080"
json_post = ""
json_response = ""
num_wanlinks = -1
# see if there are old wanlinks to remove
lf_r = LFRequest.LFRequest(base_url+"/wl/list")
try:
   json_response = lf_r.getAsJson()

   # For debugging, this makes json response more legible:
   LFUtils.debug_printer.pprint(json_response)
   for key,value in json_response.items():
      if (isinstance(value, dict) and "_links" in value):
         num_wanlinks = 1
except urllib.error.HTTPError as error:
```

```
   num_wanlinks = 0;
```

JSON output

```
{
  "wl_eg1" : {
      "_links" : "/wl/42.6",
      "name" : "wl_eg1",
      "entity id" : "Cross-Connect cx_id: 42, type: 6, idx: 0"
   },
   "uri" : "wl/:wl_id",
   "handler" : "candela.lanforge.GenericJsonResponder"
}
```

If there are no wanlinks, you will only see a warnings block telling you there are connections found that don't apply as WANlinks:

```
{
"warnings" : [
   "HttpWl::selectColumnsFromRow: eid not in table: Cross-Connect cx_id: 17, type: 1, idx: -1",
   "HttpWl::myEvaluateGet: EidCx type 1 (LANforge / UDP) unavailable in WL table: 17.1",
   "HttpWl::selectColumnsFromRow: eid not in table: Cross-Connect cx_id: 18, type: 1, idx: -1",
   "HttpWl::myEvaluateGet: EidCx type 1 (LANforge / UDP) unavailable in WL table: 18.1"
],
"handler" : "candela.lanforge.GenericJsonResponder",
"uri" : "wl/:wl_id"
}
```

2. ## Removing a WANlink

If we found WANlinks, we can remove them by posting the data to the corresponding CLI command URIs: `/cli-json/rm_cx` and `/cli-json/rm_endp`.

> i Remember the naming convention: Layer-3 and WANlink endpoints end with `-A` and `-B`. A WANlink named *westin500* has endpoints named *westin500-A* and *westin500-B*.

```
if (num_wanlinks > 0):
   lf_r = LFRequest.LFRequest(base_url+"/cli-json/rm_cx")
   lf_r.addPostData({
      'test_mgr': 'all', # could be 'default-tm', too
      'cx_name': 'wl_eg1'
   })
   lf_r.jsonPost()
   sleep(0.05)
```

The parameters for each command can be found via the help page: *http://localhost:8080/help/rm_cx*. Notice that slight pause between commands: 50ms is a good idea between deletion commands.

```
   lf_r = LFRequest.LFRequest(base_url+"/cli-json/rm_endp")
   lf_r.addPostData({
      'endp_name': 'wl_eg1-A'
   })
   lf_r.jsonPost()
   sleep(0.05)

   lf_r = LFRequest.LFRequest(base_url+"/cli-json/rm_endp")
   lf_r.addPostData({
      'endp_name': 'wl_eg1-B'
   })
   lf_r.jsonPost()
   sleep(0.05)
```

3. ## Creating WANLink Endpoints

Create the two endpoints first. Each side of a WANlink has its own transmission rate, buffer size and

corruption parameters. Each WANlink requires an ethernet port. Side A will be 128,000bps with 75ms latency:

```
lf_r = LFRequest.LFRequest(base_url+"/cli-json/add_wl_endp")
lf_r.addPostData({
   'alias': 'wl_eg1-A',
   'shelf': 1,
   'resource': '1',
   'port': 'eth3',
   'latency': '75',
   'max_rate': '128000',
   'description': 'cookbook-example'
})
lf_r.jsonPost()
sleep(0.05)
```

Side B will be 256,000bps with 95ms latency:

```
lf_r = LFRequest.LFRequest(base_url+"/cli-json/add_wl_endp")
lf_r.addPostData({
   'alias': 'wl_eg1-B',
   'shelf': 1,
   'resource': '1',
   'port': 'eth5',
   'latency': '95',
   'max_rate': '256000',
   'description': 'cookbook-example'
})
lf_r.jsonPost()
sleep(0.05)
```

4. **Create the WANlink**
   Creating the WANlink is simple, we will add it to the default test manager *default-tm*:

```
lf_r = LFRequest.LFRequest(base_url+"/cli-json/add_cx")
lf_r.addPostData({
   'alias': 'wl_eg1',
   'test_mgr': 'default_tm',
   'tx_endp': 'wl_eg1-A',
   'rx_endp': 'wl_eg1-B',
})
lf_r.jsonPost()
sleep(0.05)
```

5. **Start the WANLink**
   The LANforge server is very asynchronous. Before immediately changing the state on a connection or endpoint, test to see that it exists.

   Polling for the WANlink

```
seen = 0
while (seen < 1):
   sleep(1)
   lf_r = LFRequest.LFRequest(base_url+"/wl/wl_eg1?fields=name,state,_links")
```

Note how we can request fields by name, in this case *name, state, and _links*.

```
try:
      json_response = lf_r.getAsJson()
      if (json_response is None):
         continue
```

If there is no response, or we get a 400 error, the wanlink has probably not finished creating. Our response will be **None**. In the reponse below, we're testing for *dict* entries that have the key `_links` in them. If the name value matches, our WANlink has been created:

```
      for key,value in json_response.items():
         if (isinstance(value, dict)):
            if ("_links" in value):
               if (value["name"] == "wl_eg1"):
                  seen = 1
```

It might be helpful to use these *else* clauses when getting started:

```
               #else:
               #   print(" name was not wl_eg1")
            #else:
            #   print("value lacks _links")
         #else:
         #   print("value not a dict")

   except urllib.error.HTTPError as error:
      print("Error code "+error.code)
      continue
```

Change the WANLink State

Starting and stopping connections is done by changing the state:

```
lf_r = LFRequest.LFRequest(base_url+"/cli-json/set_cx_state")
lf_r.addPostData({
   'test_mgr': 'all',
   'cx_name': 'wl_eg1',
   'cx_state': 'RUNNING'
})
lf_r.jsonPost()
```

Polling the WANlink State

The connection might take a second to start. You can poll it similar to to how we polled it above:

```
running = 0
while (running < 1):
   sleep(1)
   lf_r = LFRequest.LFRequest(base_url+"/wl/wl_eg1?fields=name,state,_links")
   try:
      json_response = lf_r.getAsJson()
      if (json_response is None):
         continue
      for key,value in json_response.items():
         if (isinstance(value, dict)):
            if ("_links" in value):
               if (value["name"] == "wl_eg1"):
                  if (value["state"].startswith("Run")):
                     running = 1

   except urllib.error.HTTPError as error:
      print("Error code "+error.code)
      continue
```

6. **Modifying the WANlink**

   The frequency fields below are in occurrance per million. Speeds are set in bits per second (bps). Latencies are in milliseconds.

```
lf_r = LFRequest.LFRequest(base_url+"/cli-json/set_wanlink_info")
lf_r.addPostData({
   'name': 'wl_eg1-A',
   'speed': 265333,         # bps
   'latency': 30,           # 30 ms
   'reorder_freq': 3200,    #  3200/1000000
   'drop_freq': 2000,       #  2000/1000000
   'dup_freq': 1325,        #  1325/1000000
   'jitter_freq': 25125,    # 25125/1000000
```

```
})
lf_r.jsonPost()
```

7. **Stopping the WANlink**

Choose your Stop State
Stopping a WANlink is again, changing its state to either STOPPED or QUIESCE. The QUIESCE state stops transmission on both endpoints but does not close the connection so that in-flight packets can arrive. Choose QUIESCE if you want to make your accounting of packets the most accurate.

```
lf_r = LFRequest.LFRequest(base_url+"/cli-json/set_cx_state")
lf_r.addPostData({
   'test_mgr': 'all',
   'cx_name': 'wl_eg1',
   'cx_state': 'STOPPED'
})
lf_r.jsonPost()
```

Poll Untill Stopped
There might be a milliseconds to seconds of delay depending on how your connection is stopped. You might have to wait for slightly longer than QUIESCE-TIME before the connections are closed when using a QUIESCE stop. Polling the state of the connection is relatively simple:

```
running = 1
while (running > 0):
   sleep(1)
   lf_r = LFRequest.LFRequest(base_url+"/wl/wl_eg1?fields=name,state,_links")
   # LFUtils.debug_printer.pprint(json_response)
```

You might want to watch that debug output at first.

```
   try:
      json_response = lf_r.getAsJson()
      if (json_response is None):
         continue
      for key,value in json_response.items():
         if (isinstance(value, dict)):
            if ("_links" in value):
               if (value["name"] == "wl_eg1"):
                  if (value["state"].startswith("Stop")):
                     LFUtils.debug_printer.pprint(json_response)
                     running = 0
   except urllib.error.HTTPError as error:
      print("Error code "+error.code)
      continue
```

JSON Output

```
{ 'handler': 'candela.lanforge.GenericJsonResponder',
  'uri': 'wl/:wl_id',
  'wl_eg1': { '_links': '/wl/wl_eg1',
              'name': 'wl_eg1',
              'state': 'Run'}}
{ 'handler': 'candela.lanforge.GenericJsonResponder',
  'uri': 'wl/:wl_id',
  'wl_eg1': { '_links': '/wl/wl_eg1',
              'name': 'wl_eg1',
              'state': 'Stopped'}}
```

8. **Listing WANlink Endpoint Stats**
Each of the endpoints will show the amount of packets transmitted:

```
   lf_r = LFRequest.LFRequest(base_url+"/wl_ep/wl_eg1-A")
   json_response = lf_r.getAsJson()
```

```
LFUtils.debug_printer.pprint(json_response)
```

JSON Output

The key/value pairs are grouped for this example, but attribute order is not normally ordered.


```
{
   "endpoint" : {
      "name" : "wl_eg1-A",
      "buffer"       : 19936,    # bytes
      "corrupt 1"    : 0,     # these corruptions are per-wanpath
      "corrupt 2"    : 0,
      "corrupt 3"    : 0,
      "corrupt 4"    : 0,
      "corrupt 5"    : 0,
      "corrupt 6"    : 0,
      "delay"        : 30000,
      "dropped"      : 0,
      "dropfreq %"   : 0.200000002980232,
      "dup pkts"     : 0,
      "dupfreq %"    : 0.132499992847443,
      "eid"          : "1.1.3.82",
      "elapsed"      : 7,
      "extrabuf"     : 17408,
      "failed-late"  : 0,
      "jitfreq %"    : 2.51250004768372,
      "maxjitter"    : 0,
      "maxlate"      : 606,
      "max rate"     : 265333,
      "ooo pkts"     : 0,
      "qdisc"        : "FIFO",
      "reordfrq %"   : 0.319999992847443,
      "run"          : false,
      "rx bytes"     : 0,
      "rx pkts"      : 0,
      "script"       : "None",      # applicable if 2544 script has be applied
      "serdelay"     : 45648.296875,
      "tx bytes"     : 0,
      "tx drop %"    : 0,
      "tx pkts"      : 0,
      "tx rate"      : 0,
      "tx-failed"    : 0,
      # below is a to-string debug value
      "wps" : "TblJButton, eid: Shelf: 1  Resource: 1  Port: 3  Endpoint: 82  Type: WanLink",
   },
   "uri" : "wl_ep/:wl_ep_id",
   "candela.lanforge.HttpWlEndp" : {
      "duration" : "0"
   },
   "handler" : "candela.lanforge.HttpWlEndp$JsonResponse"
}
```