Development of PyNSM2 module

Mikhail Remnev

DAQ meeting, 2021.06.11

Motivation

One of frequent use cases for NSM2 is reading multiple process variables with nsmvget. However, each call of nsmvget:

1. Initializes NSM context.
2. Registers callback functions.
3. Actually reads the variable.

It's convenient to use Python as scripting language to do (1), (2) only once and then read as many variables as necessary.

PyNSM2 module is available in PR #388 and can be used in three ways:

a. Client scripts to read/write NSM2 variables in a synchronous way.
   Fully implemented, used with ELK and probably for auto mode.
b. Client scripts to read/write NSM2 variables in an event-driven way.
   Fully implemented, used in some PXD code and in SALSAgent.
c. Fully featured NSM2 nodes.
   Fully implemented, not yet used anywhere, I think.

Client scripts

- Synchronous requests.

```
import nsm2
#          nodename   port
nsm2.init('MY_NODE', 9020)
rcstate = nsm2.vget('runcontrol', 'rcstate')
print(rcstate)
```

- Event-driven application.

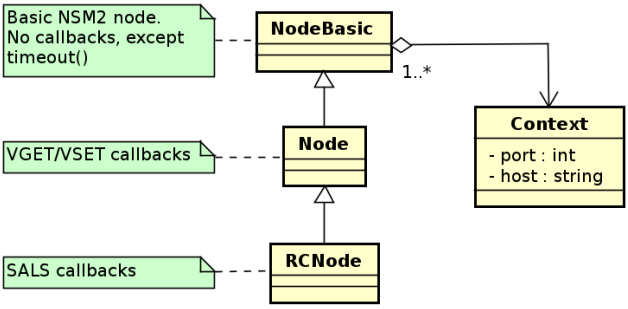
```
import nsm2
#                nodename   port
node = nsm2.Node('MY_NODE', 9020)
def vset_callback(self, msg):
    data = nsm2.unpackVSETData(msg)
    if not data.valid: return
    pv = data.varname
    val = data.value
    print(pv, val)
# Call vset_callback on each rcstate update
node.add_callback('VSET', vset_callback)
# Subscribe to RUNCONTROL.rcstate updates
node.get_context().vget('RUNCONTROL', 'rcstate')
# Run infinite loop, handling VSET requests
node.wait()
```

- There is also one interesting bug/feature in pyNSM2 handling of callbacks: if exception happens inside the callback, the application is not terminated, only the callback processing is stopped.
- This is very useful in some cases.
- However, this might be somewhat confusing. Should I fix this bug?

NSM2 node support

For event-driven operation, pyNSM2 provides three classes:

1. nsm2.NodeBasic, has no callbacks by default. New callbacks can be easily added by add_callback(..) method.
2. nsm2.Node, supports VGET and VSET callbacks by default.
3. nsm2.RCNode, supports STOP/ABORT/LOAD/START callbacks, has an rcstate.




```
#                nodename   port
node = nsm2.Node('example', 9999)
node.add('ival', 11)
node.add('tval', 'Some text')
def time_get(self, varname): # <-- custom callback
    return str(datetime.now())
node.add('time', '', getter=time_get, setter=None)
#== Run an infinite loop, responding to the requests
node.wait()
```

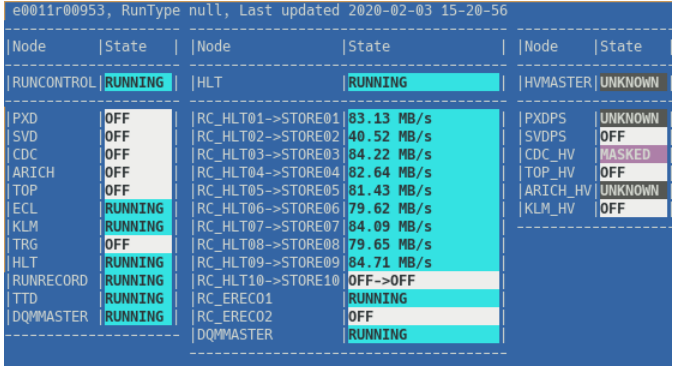
```
> nsmvlistget example
 ival   : int get set
 time   : text get
 tval   : text get set
> nsmvget example time
 time : 2021-06-10 12:45:36
```

Usage

Clone forked version of the repository:

1. Clone daq_slc repository.
   `git clone -recursive ssh://git@stash.desy.de:7999/b2daq/daq_slc.git`
2. Switch to the correct branch.
   `cd daq_slc; git checkout feature/add-pynsm2`
3. Build daq_slc:
   `source setenv; ./install.sh`
4. Use pynsm2:
   `python2 # or python3`
   `import nsm2`

* Run one of the examples:
`python python/examples/rcview.py std`



Documentation

- Most of module functions are documented.
- I'm using Doxygen format, so in theory it is very easy to export them into HTML file or PDF.

```
vget(target, varname)
    Send VGET request to get PV of specified NSM node
    Have to once call nsm2.register('vset') before that.

    @param target      Name of the target node
    @param varname     Name of the variable
    @return            Value of the specified variable

    Only int, float and string types are supported
```

- There is also a README.md file but it is a bit outdated.

Folder structure

PR #388 includes two additional packages besides nsm2:

- daq_slc/python/examples — usage examples.
- daq_slc/python/tests — automated tests.
- daq_slc/python/update_nsm_mappings — make automated updates to pyNSM2 if there are changes in the original NSM2.
- daq_slc/python/packages — three Python packages for slow control.
  - nsm2 — Python functions for NSM2 (based on ctypes).
  - daqdb — reading configs from DAQ DB.
  - b2slc — logging, reading daq_slc/data/config, running as daemon.

b2slc package

```
import b2slc
import nsm2
import logging

b2slc.add_logfile(__file__, 'example')
b2slc.daemon()
#                nodename   port
node = nsm2.Node('example', 9999)
node.add('ival', 11)
node.add('tval', 'Some text')
try:
    #== Run an infinite loop, handle requests
    node.wait()
except:
    logging.exception('Terminated with an exception:')
```

Logging to ~/log/test/example/ (doesn't support logging to ELK at the moment)

Starting as daemon

Log all crashes (except SIGTERM)

- See daq_slc/python/examples/daemon.py for detailed example.

Semi-automated tests

- There is a script that automatically starts nsmd2 and runs through all 4 possible combinations of Python versions (py2 server ↔ py2 client, py3↔py2, py2↔py3, py3↔py3)

```
[remnev@remnev-pc tests]$ ./00_test_all.sh
[2021-06-10 21:53:12] [INFO] Starting NSMD2 at port 9999
NSM2 at port 9999 is already started
[2021-06-10 21:53:12] [INFO] Running tests with nodes TEST_SERVER|TEST_CLIENT at nsmd2:9999
Traceback (most recent call last):
  File "/home/remnev/daq_slc_pynsm_2020.12.07/python/tests/pynsm2_client.py", line 20, in <module>
    ival = nsm2.vget('test_server', 'ival')
  File "/home/remnev/daq_slc_pynsm_2020.12.07/lib/python/site-packages/nsm2/__init__.py", line 64, in vget
    return node.get_context().vget(target, varname, wait_reply=True, timeout=timeout)
  File "/home/remnev/daq_slc_pynsm_2020.12.07/lib/python/site-packages/nsm2/nsmcontext.py", line 133, in vget
    target = target.upper().decode('ascii')
AttributeError: 'str' object has no attribute 'decode'
[2021-06-10 21:53:15] [ERROR] py2 server <=> py3 client test failed
```

- Of course, the problem with such tests is that sometimes I forget to run them.
- Not sure if it's a good idea to run these tests for each build of daq_slc.

Current status

A lot of features are now available in PR #388 to daq_slc:

- Python 2/3 compatibility (SL5 is not supported but can be done as well).
- Sending different requests (e.g. vget/vset).
- Shared memory allocation and access.

- NSM2 callbacks as python functions.
- NSM2 nodes implementation.
- Subscription to variable updates via VGET requests.

- Logging.
- Access to daq_slc configuration.
- Access to DAQ DB.

- Many usage examples provided in daq_slc/python/examples directory.

Summary and further steps

- As mentioned on the previous slide, there are now a lot of features provided by PyNSM2.
- The code has been tested in several applications both on Python 2 and Python 3.

- Would it be possible to merge PR #388?
  I think it might be very useful to have pyNSM2 available on CVMFS.
- Where should we keep PyNSM2 apps?
  daq_slc/apps/ ?
  daq_slc/python/apps ?
  daq_slc/python/packages ?
- Should I fix the bug with exception handling? (slide 3)

Backup slides

Performance

Python implementation is slower by $\sim$ 0.08 seconds.

```
time nsmvlistget → 0.02 seconds

time nsmvlistget.py → 0.1 seconds
```

- This is likely caused by longer startup time.
- This might be ctypes-only issue, further study is necessary.