# Python configuration generation simplification

Philip Rodrigues

University of Oxford

December 1, 2021

## Summary

- Modifying configuration generation to be “what the user wants to do” rather than “how the underlying app framework implementation works”
- Since last time: got a limited minidaqapp working
- Aim for today: can we get this in 2.10? What are the necessary steps?
- Many more details in

  https://github.com/DUNE-DAQ/minidaqapp/tree/philiprodrigues/conf-shorter/python/minidaqapp/nanorc

## A simple example

```
# moo imports omitted

from .util import Module, ModuleGraph, Direction
from .util import Connection as Conn
from . import util

######## Create app1 ########
modules = {}

modules["mod1"] = Module(plugin = "MyModule1",
                         conf =  mod1.Conf(...) # configuration parameters omitted
                         connections = {"output": "mod2.input"})
modules["mod2"] = Module(plugin = "MyModule2",
                         conf =  mod2.Conf(...) # configuration parameters omitted
                         connections = {}) # No output connections

mgraph1 = ModuleGraph(modules)
mgraph1.add_endpoint("time_sync", "mod1.time_sync_source", Direction.IN)
mgraph1.add_endpoint("hsievent",  "mod2.hsievent_sink",    Direction.OUT)

######## Create system ########
the_system = util.System()
the_system["app1"] = util.App(modulegraph = mgraph1, host="localhost")
# You'll have to imagine the creation of the modulegraph for app2
the_system["app2"] = util.App(modulegraph = mgraph2, host="localhost")

the_system.app_connections = {"app1.hsievent": util.Sender(msg_type="dunedaq::dfmessages::HSIEvent",
                                                           msg_module_name="HSIEventNQ",
                                                           receiver="app2.hsievents_in")}

######## Do everything remaining to create the json files for nanorc ########
util.make_apps_json(the_system, "json/")
```

## minidaqapp tests so far

- Using a v2.8.2 base release, and minidaqapp branched from there, tested with this command:

```
python -m minidaqapp.nanorc.mdapp_multiru_gen --host-ru localhost -d ./frames.bin -o . -s 10 json \
    --enable-software-tpg --trigger-activity-config 'dict(prescale=1000)' --number-of-data-producers 2
```

- Produces HDF5 file with expected fragments in it
- Requires a small (and I think, sensible) change to `TriggerRecordBuilder`
- Some of the limitations: No DQM, no real HSI (only fake), probably doesn't work with real hardware, Kurt reports it doesn't work with multiple readout apps

## Next steps

- Definitely needed for 2.10, I think:
  1. Find a better place to put this code. In its own package? Needs discussion with sw coordination
  2. Check with sw coordination that the `networkx` python package dependency is OK
  3. Support NetworkManager
  4. Update the minidaqapp configuration to whatever is current, with full support for everything
     - Update to “current” pre-NetworkManager, or skip straight to config with NM?
- The steps above are probably enough to keep us busy. Other things we might want before we put this in a release:
  1. Work out how to support Conwex (Brett has suggestions)
  2. While we're changing things here, maybe try to fix the option explosion in minidaqapp? ($\sim$ 60 arguments and counting!)

## Lower-priority/smaller items

1. Rename `Module` to `DAQModule` and `System` to `Partition`, as suggested by Brett
2. Don't store items in dicts keyed by name, but use lists and put item names in the class of the item
3. Add a `validate()` method to each of the data objects. Checks would include things like making sure that all endpoints referred to in intra-/inter-app connections actually exist
4. Split up code into multiple files (Pierre has suggestions)
5. Make graphviz diagrams at various stages/levels for debugging. Eg:
   - Draw a `ModuleGraph` showing the internal modules, connections and endpoints
   - Draw a `System` object with the individual apps showing just their external endpoints, no internals
6. Move the `msg_type` and `msg_module_type` settings in N2Q/Q2N connections from the connection to the endpoints
7. Switch to using the `logging` package for logging, so that we have different levels. We can keep the output looking fancy via `rich.Console` with the technique described at `https://rich.readthedocs.io/en/stable/logging.html` and also used in `nanorc`
8. When a topological sort finds cycles, display the cycles to the user for debugging
9. Maybe make connection to outgoing external endpoint be in Module ctor, instead of separate add_endpoint
10. Brett suggests using python module cleverness to allow items to be accessed via dot-paths like `system.app.module.sink_or_source`

## Points for discussion

- Should we aim to get this in 2.10?
- If so, which items are must-haves for 2.10?
- How do we order the work?

Backup slides

## Supporting Conwex

- My understanding, based on discussions with Brett:
- Conwex needs to know about the "data" classes (`ModuleGraph`, `App`, `System`) and system options (ie, cmd line args) via JSON Schema, so that it can present the options in the web UI, and send modified versions back
- One option is to use moo schema written in jsonnet for the "data" classes. moo supports creation of JSON Schema
- Could also specify classes as pydantic "models", which can be converted to JSON Schema. I mildly favour this, as it seems simpler for me
- The arguments to the `generate()` functions would also need to be specified in a Conwex-accessible format. I think this will tie in with fixes to argument explosion, and involve being clearer about the "API" that app generators implement, but the details are still a bit beyond me