ophyd Devices: Imposing Hierarchy on the Flat EPICS V3 Namespace

Photon Controls & Data Systems Department
Kenneth Lauer

Abstract
EPICS V3 provides simple data types accessible over the network through Channel Access identified by a flat process variable (PV) name. This flexibility is often regarded as a strength of EPICS, as the user can easily pick and choose the information they require. However, such data is almost always inter-related in some manner, pushing the burden of reconstructing that relationship to the end-user/client.

ophyd devices make imposing this hierarchy simple, readable, and descriptive. ophyd represents hardware in Python as hierarchical classes, grouping together related signals from the underlying control system. This structure allows ophyd to provide a consistent interface across a wide-range of devices, which can then be used by higher-level software for any number of tasks: from command-line inspection, to scanning/data collection (Bluesky), or even automatic GUI generation (typhon, adviwer). ophyd contains a number of pre-built devices for common hardware (and IOC's) as well as the tools to build custom devices.

Take advantage of the built-in Devices
- Aggregate and classify components
- Signals (e.g., EpicsSignal / PVs)
- Devices
- Human-friendly naming, readable code
- Computer-friendly: consistent programming interface
- Metadata included: timestamps, control limits, units, ...
- Built-in abstractions for many common devices
- Motor record, scalers, DXP, ...
- AreaDetector cameras, plugins
- Simple simulated motors, detectors, etc.
- Framework for building your own custom devices

Or write your own... a simple example

```
from ophyd import Device, Component, EpicsSignalWithRBV

device_1 = MyDevice('AreaDetector1', name='device_1')
device_2 = MyDevice('AreaDetector2', name='device_2')

det = ophyd.SimpleDetector('131M1', name='det')
det.cam.manufacturer.get() # -> 'Simulated detector'

motor = ophyd.EpicsMotor('sinctrl1', name='motor')
st = motor.set(1.2) # -> status object
motor.read()
```

Use the device on the Python command line:

```
In [1]: device = device_1
In [2]: device['ItemA']
Out[2]: Component('ItemA', dtype=EpicsSignalWithRBV)
```

Make a Typhon/PyDM GUI automatically:

```
suite = typhon.TyphonSuite()
suite.add_device(device_1)
suite.add_device(device_2)
```

Build full applications using the interface... (*)

Use the device in a bluesky scan:

```
In [1]: RS(sample_scan([device_1, device_2], device_1.ItemA, [0, 1, 0.1]), ...
   ...:      reset=forwards)  # ...
```

This isn’t noise...

All of the 5,500 AreaDetector PVs made available by default in the R3-2 simulated detector IOC in a size 1 font.

Tips
- Name your PVs sensibly
- Follow a convention – convince others
- Use a common prefix for grouping
- Think about how it’ll map onto a Device while writing records

(*) For more information, see my “adviewer” poster

https://blueskyproject.io/

Acknowledgements
Special thanks to Thomas Caswell, Daniel B. Allan (Brookhaven National Laboratory), and Daron Chabot (FRIB, Michigan State University). ophyd would not be where it is today without you.

SLAC National Accelerator Laboratory is operated by Stanford University for the U.S. Department of Energy Office of Science. Work supported by U.S. D.O.E. Contract DE-AC02-76SF00515