

# Typhon

Automated Generation of PyDM Displays

Teddy Rendahl

- PyDM opens the door to a more modern UI
  - Integrated with experimental Python environment
  - Include information from multiple data sources
  - Programmatically generate screens using Python
- Thousands of existing EDM screens to convert or recreate
- Four new LCLS-II instruments with a focus on lowered operations budget
  - Less support from engineers to create and maintain screens
  - Less support from staff scientists to assist with user beamtimes

- **Engineering Goals**

- Avoid "drag and drop" widget creation wherever possible
- Decrease the maintenance required to keep screens updated by removing configuration information
- Provide a clear framework for adding intelligence to screens

- **Operational Improvements**

- Uniform methodology for displaying complex device structures
- Importance of control points is clear from screen design
- Consistency between interfaces lowers operator training time

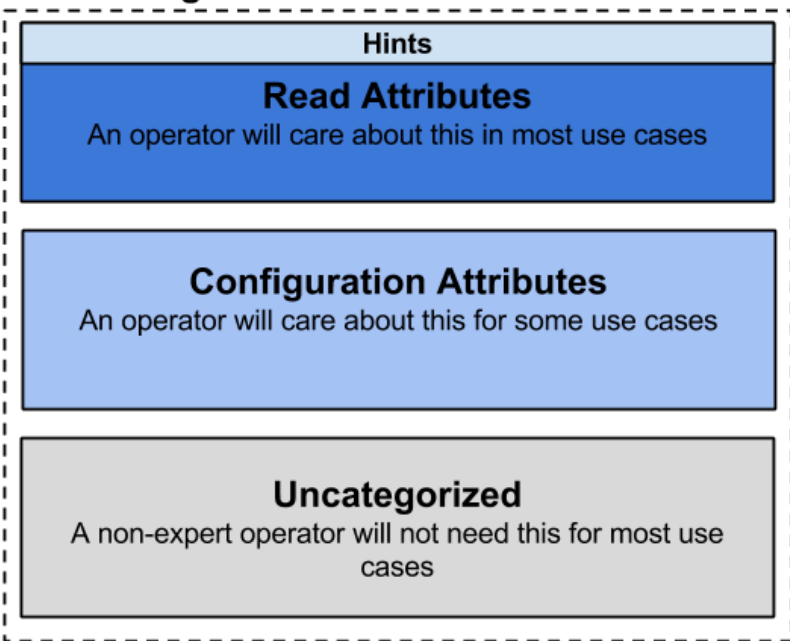
# Typhon

- Automatically build PyDM displays based on the structure of an Ophyd Device
- Use the abstraction layer of Ophyd signals with PyDM to display raw and interpreted data
- Consolidate tools formerly kept in separate applications



# Ophyd as a Device Data Structure

## Device Signals

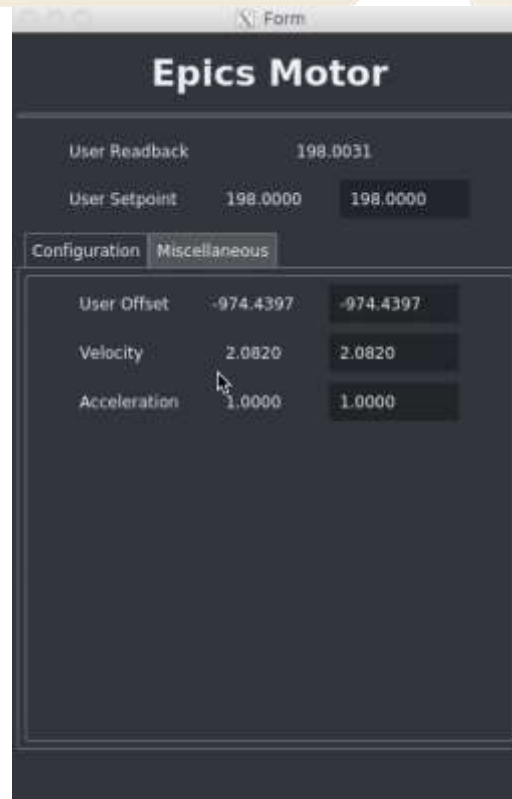


- Contained within an Ophyd Device:
  - Clear component hierarchy
  - Terse but human readable aliases for control points
  - Categorization of signals based on the relevance to standard operation

# Basic Signal Organization

- Show most important attributes at the top while the remainder are into tabs
- Automatically recognize signal types including:
  - Read-Only
  - Enum
  - Simulated

```
1 motor = ophyd.EpicsMotor('Tst:Mtr:01', name='epics_motor')
2 device_display = typhon.DeviceDisplay(motor)
```



The screenshot shows a web-based control interface for an Epics Motor. The title is "Epics Motor". It displays several parameters and their values:

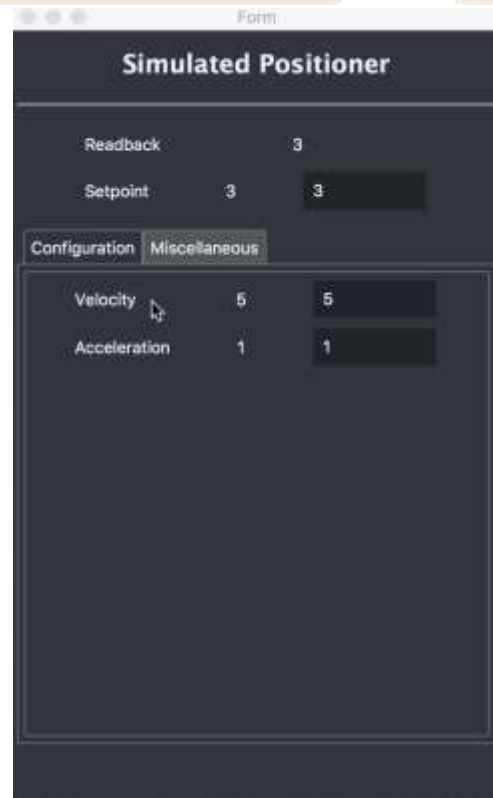
Parameter	Value	Value
User Readback	198.0031	
User Setpoint	198.0000	198.0000
User Offset	-974.4397	-974.4397
Velocity	2.0820	2.0820
Acceleration	1.0000	1.0000

The interface has two tabs: "Configuration" and "Miscellaneous". The "Miscellaneous" tab is currently selected. A mouse cursor is visible over the "Velocity" value.

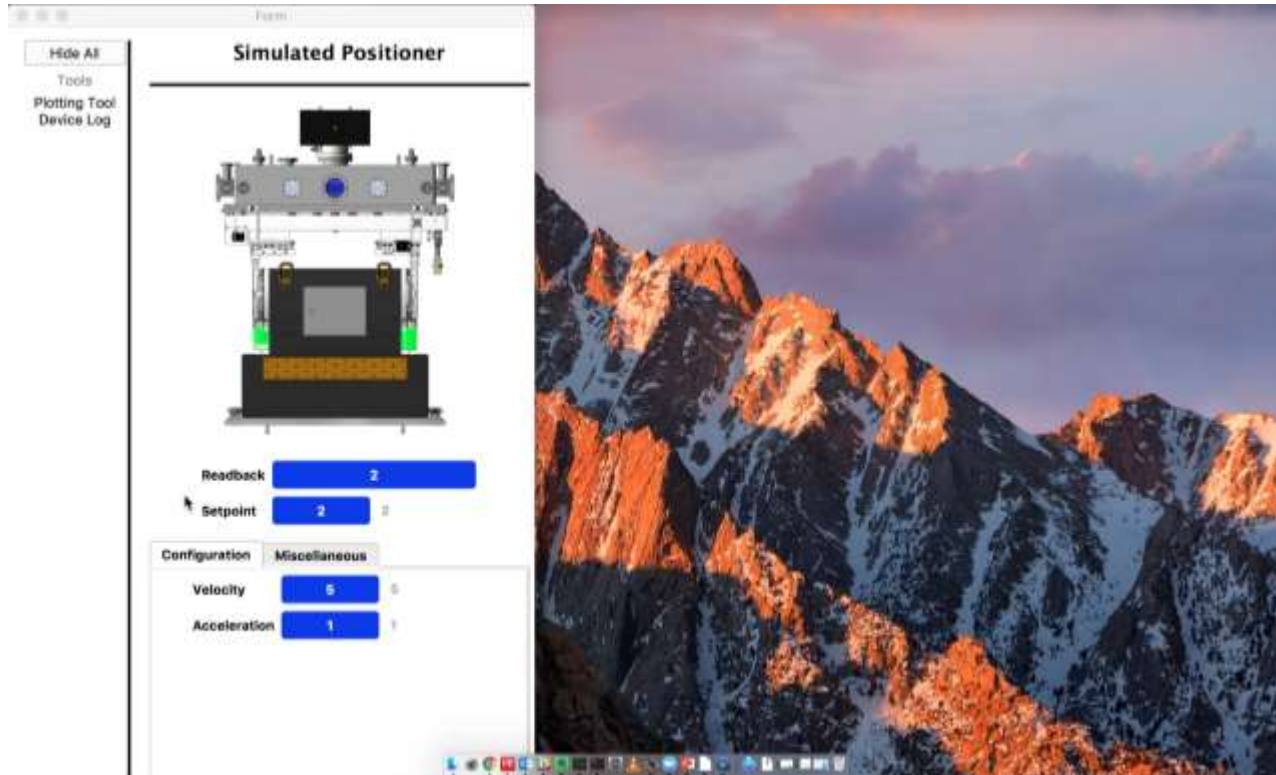
# Basic Signal Organization

- Show most important attributes at the top while the remainder are into tabs
- Automatically recognize signal types including:
  - Read-Only
  - Enum
  - Simulated

```
1 motor = SynPositioner(name='simulated_positioner')
2 device_display = typhon.DeviceDisplay(motor)
```



# Expanding the Interface





# Complex Devices

The screenshot displays a control interface for a 'Complex Stage'. On the left, a sidebar contains a 'Hide All' button, a 'Components' section with 'X', 'Y', and 'Pitch' options, and a 'Tools' section with 'Plotting Tool' and 'Device Log'. The main area features a 3D model of the stage, which is a rectangular platform with a central square opening, supported by four vertical columns. Below the model, the following readback values are shown:

X Readback	0.000
Y Readback	0.000
Pitch Readback	0.000

At the bottom, there are two tabs: 'Configuration' and 'Miscellaneous'. The 'Configuration' tab is active, showing three rows of velocity settings:

X Velocity	5	5
Y Velocity	5	5
Pitch Velocity	5	5

The background of the interface is a scenic image of snow-capped mountains under a purple and blue sky.

# Including Logic

```
def move(self, x: float, y: float, z: float, simultaneous: bool=True):
    """
    Move the XYZStage
    """
    for stage, request in zip((self.x, self.y, self.z), (x, y, z)):
        stage.move(request, wait=not simultaneous)

def move_radially(self, radius: float, theta: float, phi: float,
                  simultaneous: bool=True):
    """
    Move the XYZStage as specified in spherical coordinates
    """
    x = r * sin(theta) * cos(phi)
    y = r * sin(theta) * sin(phi)
    z = r * cos(theta)
    self.move(x, y, z, simultaneous=simultaneous)
```

# Including Logic



# Major Advantages

- Common way to present complex devices
- Spend more time thinking about how to organize your device rather than dragging and dropping
- Inclusion of higher level logical operations is simple
- Synergy between command line and Graphical User Interface

The screenshot displays a graphical user interface for a 'Complex Stage'. On the left, there is a sidebar with a 'Hide All' button and two sections: 'Components' containing 'X', 'Y', and 'Z', and 'Tools' containing 'Plotting Tool' and 'Device Log'. The main area features a 3D model of a mechanical stage. To the right of the model, a green text label reads '[In [13]: xyz.x.velocity = 4'. Below the model, there are three blue buttons for 'X Readback', 'Y Readback', and 'Z Readback', each displaying the value '0.000'. At the bottom, there are two tabs: 'Configuration' and 'Miscellaneous'. Under the 'Miscellaneous' tab, there are three blue buttons for 'X Velocity', 'Y Velocity', and 'Z Velocity', each displaying the value '5'.

- **Current Status**

- V0.2.0 released in beta
- Will be deployed in the Fall for operator feedback

- **Planned Improvements**

- Handle positioners uniquely
- Inclusion of more external tools
- Prototype for a RunEngine with fixed plans
- Beamline process specific screens, not device specific



# Support for Complete Screen Lifecycle

1. Screen is automatically generated by Typhon
2. Screen is saved to disk for use by operators
3. Adjustments to screens are made by operators
4. As updates to tools and devices are made, screens are re-imported and updated in a programmatic fashion

## Documentation

<https://pcdshub.github.io/typhon/>

## Repository

<https://github.com/pcdshub/typhon>