docs Documentation

Release 1.0

Will Rogers, Razvan Vasile

Jul 01, 2022

Contents

Python Toolkit for Accelerator Controls (Pytac) is a Python library for working with elements of particle accelerators, developed at Diamond Light Source.

It is hosted on Github at: https://github.com/dls-controls/pytac

Two pieces of software influenced its design:

- Matlab Middlelayer, used widely by accelerator physicists.
- APHLA, high-level applications written in Python by the NSLS-II accelerator physics group.

CHAPTER 1

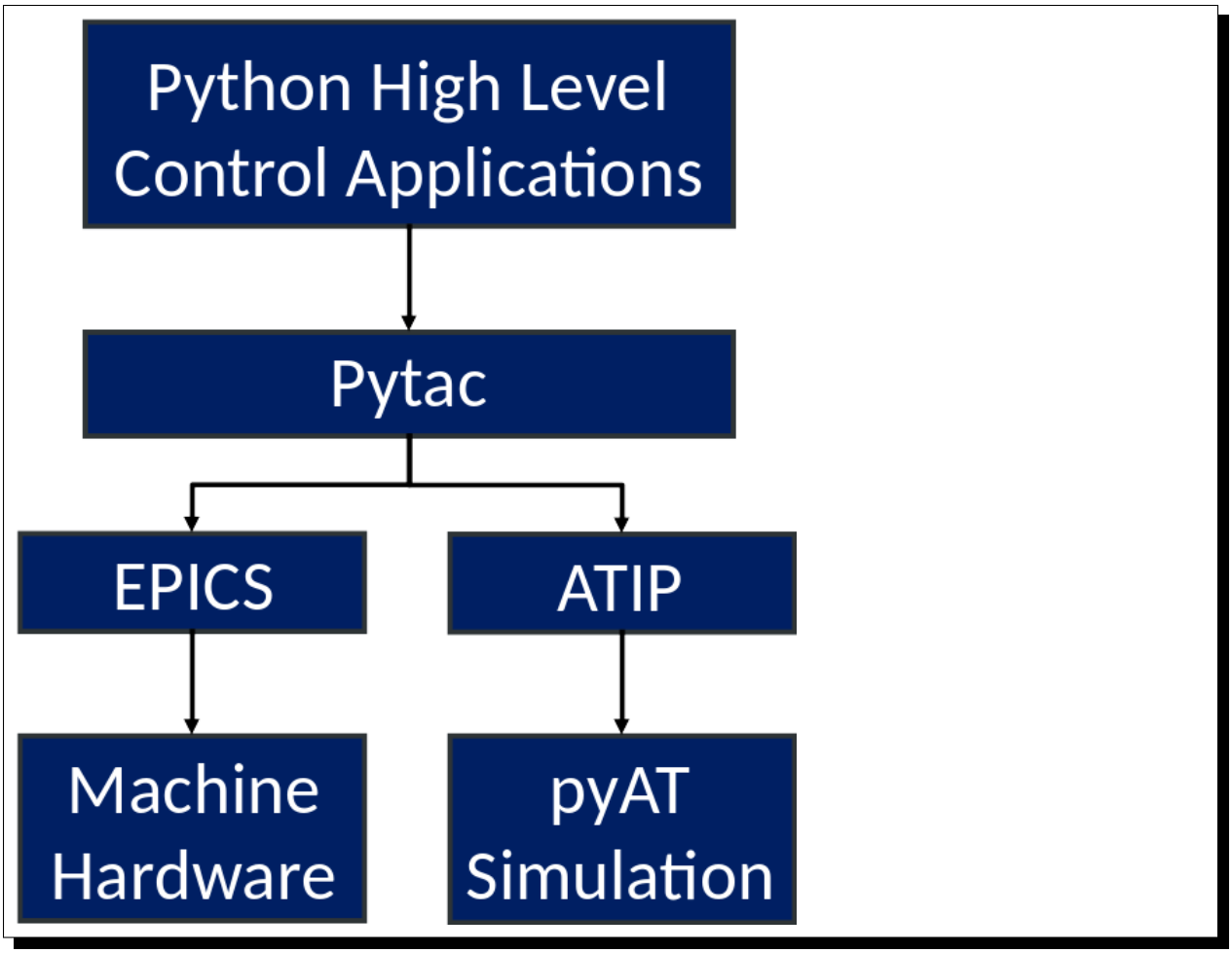
Overview

Pytac provides a Python library, `pytac`, that makes it easier to communicate with machine hardware for online applications. Although it currently works with EPICS, it should be possible to adapt to support other control systems.

The design is based around a `Lattice` object that contains a sequence of `Element` s. Each element represents a physical component in the accelerator, such as an electromagnet, drift, or BPM. Each element may have zero or more 'fields', each representing a parameter of the component that may change e.g. a BPM element has fields 'x' and 'y' that represent the beam position, and a quadrupole magnet has 'b1' that represents the quadrupolar magnetic field. Each field has one `Device` object for monitoring and control purposes, these devices contain the necessary information to get and set parameter data using the control system.

Elements may be grouped into families (an element may be in more than one family), and requested from the lattice object in those families. The current control system integrates with EPICS and uses EPICS PV (process variable) objects to tell EPICS which IOC (input/output controller - an EPICS server process) to communicate with. The type of the PV specifies which operations can be performed, there are two types of PV: readback, which can only be used to retrieve data; and setpoint, which can be used to set a value as well as for retrieving data. A single component may have both types; and so some methods take 'handle' as an argument, this is to tell the control system which PV to use when interfacing with EPICS, readback (`pytac.RB`) or setpoint (`pytac.SP`).

An example control structure.



Data may be set to or retrieved from different data sources, from the live machine (`pytac.LIVE`) or from a simulator (`pytac.SIM`). By default the 'live' data source is implemented using Cothread to communicate with EPICS, as described above. The 'simulation' data source is left unimplemented, as Pytac does not include a simulator. However, ATIP, a module designed to integrate the Accelerator Toolbox simulator into Pytac can be found here.

Data may also be requested or sent in engineering (`pytac.ENG`) or physics (`pytac.PHYS`) units and will be converted as appropriate. This conversion is a fundamental part of how Pytac integrates with the physical accelerator, as physics units are what our description of the accelerator works with (e.g. the magnetic field inside a magnet) and engineering units are what the IOCs on the physical components use (e.g. the current in a magnet). Two types of unit conversion are available:

- Polynomial (`PolyUnitConv`; often used for linear conversion);
- Piecewise Cubic Hermite Interpolating Polynomial (`PchipUnitConv`; often used for magnet data where field may not be linear with current).

In the case that measurement data (used to set up the conversion objects) is not in the same units as the physical models, further functions may be given to these objects to complete the conversion correctly.

Models of accelerators, physical or simulated, are defined using a set of `.csv` files, located by default in the `pytac/data` directory. Each model should be saved in its own directory i.e. different models of the same accelerator should be separate, just as models of different accelerators would be.

CHAPTER 2

Contents:

2.1 Examples

2.1.1 Installation

This is only required on your first use.

- Ensure you have Pip, then install Pytac and Cothread:

```
$ pip install pytac
$ pip install cothread
$ # Cothread is required for EPICS functionality, but Pytac can run without it.
```

2.1.2 Initialisation

This is required each time you want to start up Pytac.

- Navigate to your Pytac directory and start Python:

```
$ cd <directory-path>
$ python
Python 3.7.2 (default, Jan 20 2020, 11:03:41)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Import Pytac and initialise the lattice from the `VMX` directory:

```
>>> import pytac.load_csv
>>> lattice = pytac.load_csv.load('VMX')
```

The `lattice` object is used for interacting with elements of the accelerator.

2.1.3 Print BPM PV names along with s position

- Get all elements that represent `BPM` s:

```
>>> bpms = lattice.get_elements('BPM')
```

- Print the device names and s position of each BPM:

```
>>> for bpm in bpms:
>>>     print('BPM {} at position {}'.format(bpm.get_device('x').name, bpm.s))
BPM SR01C-DI-EBPM-01 at position 4.38
BPM SR01C-DI-EBPM-02 at position 8.8065
BPM SR01C-DI-EBPM-03 at position 11.374
BPM SR01C-DI-EBPM-04 at position 12.559
BPM SR01C-DI-EBPM-05 at position 14.9425
...
```

- Get PV names and positions for BPMs directly from the lattice object:

```
>>> lattice.get_element_pv_names('BPM', 'x', pytac.RB)
['SR01C-DI-EBPM-01:SA:X',
'SR01C-DI-EBPM-02:SA:X',
'SR01C-DI-EBPM-03:SA:X'
...
>>> lattice.get_element_pv_names('BPM', 'y', pytac.RB)
['SR01C-DI-EBPM-01:SA:Y',
'SR01C-DI-EBPM-02:SA:Y',
'SR01C-DI-EBPM-03:SA:Y',
...
>>> lattice.get_family_s('BPM')
[4.38,
8.806500000000002,
11.374000000000002,
...
```

2.1.4 Get the value of the 'b1' field of the quad elements

- Get all Quadrupole elements and print their 'b1' field read back values:

```
>>> quads = lattice.get_elements('Quadrupole')
>>> for quad in quads:
>>>     print(quad.get_value('b1', pytac.RB))
71.3240509033
129.351394653
98.2537231445
...
```

- Print the `Quadrupole` read back values of the 'b1' field using the lattice. This is more efficient since it uses only one request to the control system:

```
>>> lattice.get_element_values('Quadrupole', 'b1', pytac.RB)
[71.32496643066406,
129.35191345214844,
98.25287628173828,
...
```

2.1.5 Tutorial

For an introduction to pytac concepts and finding your way around, an interactive tutorial is available using Jupyter Notebook. Take a look in the `jupyter` directory - the `README.rst` there describes how to access the tutorial.

2.2 Developers

The installation and initialisation steps are slightly different if you want to work on Pytac. N.B. This guide uses pipenv but a virtualenv will also work.

2.2.1 Installation

This is only required on your first use.

- Ensure you have the following requirements: Pip, Pipenv, and a local copy of Pytac.
- Install dev-packages and Cothread for EPICS support:

```
$ pipenv install --dev
$ pip install cothread
$ # Cothread is required for EPICS functionality, but Pytac can run without it.
```

2.2.2 Initialisation

This is required each time you want to start up Pytac.

- Navigate to your `pytac` directory and activate a Pipenv shell, and start Python:

```
$ cd <directory-path>
$ pipenv shell
$ python
Python 3.7.2 (default, Jan 20 2020, 11:03:41)
[GCC 4.8.5 20150623 (Red Hat 4.8.5-39)] on linux
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Import Pytac and initialise the lattice from the `VMX` directory:

```
>>> import pytac.load_csv
>>> lattice = pytac.load_csv.load('VMX')
```

The `lattice` object is used for interacting with elements of the accelerator.

2.3 API Documentation

CHAPTER 3

Indices and tables

- genindex
- modindex
- search