docs Documentation

Release 1.0

Will Rogers, Razvan Vasile

Jun 20, 2019

Contents

Python Toolkit for Accelerator Controls (Pytac) is a Python library for working with elements of particle accelerators.

It is hosted on Github at https://github.com/willrogers/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 accelerator, such as an electromagnet, drift or BPM. For control purposes, each element may have zero or more 'fields' that represent parameters that may change: a BPM element has fields 'x' and 'y' that represent beam position, and a quadrupole magnet has 'b1' that represents the quadrupolar magnetic field. For monitoring and controlling these fields, elements have one `Device` object per field. These devices contain the necessary information to get and set live data using the control system.

Data may be requested in `ENG` or `PHYS` units and will be converted as appropriate. Two types of unit conversion are available: Polynomial (often used for linear conversion) and Piecewise Cubic Hermite Interpolating Polynomial (Pchip; 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.

Elements may be grouped into families (an element may be in more than one family) and requested from the lattice object in those families.

Machines are defined using a set of `.csv` files, located by default in the `pytac/data` directory.

CHAPTER 2

Contents:

2.1 Examples

2.1.1 Initialisation

- Install pytac and cothread for EPICS support, and start Python:

```
$ pip install pytac
$ pip install cothread
$ python
Python 2.7.3 (default, Nov  9 2013, 21:59:00)
[GCC 4.4.7 20120313 (Red Hat 4.4.7-3)] on linux2
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

- Import 'pytac' and initialise the VMX ring mode:

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

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

2.1.2 Print BPM pvs along with s position

- Get elements representing BPMs:

```
>>> 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))
```

(continues on next page)

(continued from previous page)

```
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_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_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,
 ...
```

2.1.3 Get the pv value from the quad elements

- Get the Quad elements and print their readback values on the b1 field:

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

- Print the quad pv values on the b1 field using the lattice. This is more efficient since it uses only one request to the control system:

```
>>> lattice.get_pv_values('QUAD', 'b1', pytac.RB)
[71.32496643066406,
 129.35191345214844,
 98.25287628173828,
...
```

2.2 API documentation

Pytac: Python Toolkit for Accelerator Controls.

2.2.1 pytac.cs module

Class representing an abstract control system.

class pytac.cs.**ControlSystem**
Bases: object

Abstract base class representing a control system.

get(*pv*)
Get the value of the given pv.

Parameters **pv** (*string*) – The pv to get the value of.

Returns The numeric value of the pv.

Return type Number

put(*pv*, *value*)
Put the value of a given pv.

Parameters

- **pv** (*string*) – The pv to put the value for.
- **value** (*Number*) – The value to be set.

2.2.2 pytac.device module

The device class used to represent a particular function of an accelerator element.

A physical element in an accelerator may have multiple devices: an example at DLS is a sextupole magnet that contains also horizontal and vertical corrector magnets and a skew quadrupole.

class pytac.device.**Device**(*name*, *cs*, *enabled=True*, *rb_pv=None*, *sp_pv=None*)
Bases: object

A device attached to an element.

Contains a control system, readback and setpoint pvs. A readback or setpoint pv is required when creating a device otherwise a DeviceException is raised. The device is enabled by default.

Parameters

- **name** – prefix of EPICS PVs for this device
- **cs** (ControlSystem) – Control system object used to get and set the value of a pv.
- **enabled** (*bool-like*) – Whether the device is enabled. May be a PvEnabler object.
- **rb_suffix** (*str*) – EPICS readback pv
- **sp_suffix** (*str*) – EPICS setpoint pv

get_cs()

get_pv_name(*handle*)
Get a pv name on a specified handle.

Parameters **handle** (*str*) – The readback or setpoint handle to be returned.

Returns A readback or setpoint pv.

Return type str

Raises *DeviceException* – if the PV doesn't exist.

get_value(*handle*)
Read the value of a readback or setpoint pv.

Parameters **handle** (*str*) – Handle used to get the value off a readback or setpoint pv.

Returns The value of the pv.

Return type float

Raises *DeviceException* – if the requested pv doesn't exist.

is_enabled()
Whether the device is enabled.

Returns whether the device is enabled

Return type boolean

set_value(*value*)
Set the device value.

Parameters **value** (*float*) – The value to set on the pv.

Raises *DeviceException* – if no setpoint pv exists.

exception pytac.device.**DeviceException**
Bases: exceptions.Exception

Exception associated with Device misconfiguration or invalid requests.

class pytac.device.**PvEnabler**(*pv*, *enabled_value*, *cs*)
Bases: object

A PvEnabler class to check whether a device is enabled.

The class will behave like True if the pv value equals enabled_value, and False otherwise.

Parameters

- **pv** (*str*) – pv name
- **enabled_value** (*str*) – value for pv for which the device should be considered enabled
- **cs** – Control system object

2.2.3 pytac.element module

Module containing the element class.

class pytac.element.**Element**(*name*, *length*, *element_type*, *s=None*, *index=None*, *cell=None*)
Bases: object

Class representing one physical element in an accelerator lattice.

An element has zero or more devices (e.g. quadrupole magnet) associated with a field ('b1' for a quadrupole).

name
name identifying the element

Type str

type_
type of the element

Type str

length
length of the element in metres

Type number

s
the element's start position within the lattice in metres

Type float

index
the element's index within the ring, starting at 1

Type int

cell
the element's cell within the lattice

Type int

families
the families this element is a member of

Type set

Parameters

- **name** (*int*) – Unique identifier for the element in the ring.
- **length** (*float*) – The length of the element.
- **element_type** (*str*) – Type of the element.
- **s** (*float*) – Position of the start of the element in the ring
- **index** (*float*) – Index of the element in the ring, starting at 1
- **cell** (*int*) – lattice cell this element is wihin

add_device(*field*, *device*, *uc*)
Add device and unit conversion objects to a given field.

A DeviceModel must be set before calling this method.

Parameters

- **field** (*str*) – The key to store the unit conversion and device objects.
- **device** (Device) – device object used for this field.
- **uc** (UnitConv) – unit conversion object used for this field.

Raises KeyError if no DeviceModel is set

add_to_family(*family*)
Add the element to the specified family.

Parameters **family** (*str*) – Represents the name of the family

get_cs(*field*)

get_device(*field*)
Get the device for the given field.

A DeviceModel must be set before calling this method.

Parameters **field** (*str*) – The lookup key to find the device on an element.

Returns The device on the given field.

Return type *Device*

Raises KeyError if no DeviceModel is set

get_fields()
Get the fields defined on an element.

Includes all fields defined by all models.

Returns A sequence of all the fields defined on an element.

Return type list

get_pv_name(*field*, *handle*)
Get a pv name on a device.

Parameters

- **field** (*str*) – requested field
- **handle** (*str*) – pytac.RB or pytac.SP

Returns readback or setpoint pv for the specified field

Return type str

Raises DeviceException if there is no device for this field

get_unitconv(*field*)
Get the unit conversion option for the specified field.

Parameters **field** (*str*) – Field associated with this conversion

Returns UnitConv object associated with the specified field

Raises KeyError if no unit conversion object is present.

get_value(*field*, *handle='readback'*, *units='engineering'*, *model='live'*)
Get the value for a field.

Returns the value for a field on the element. This value is uniquely identified by a field and a handle. The returned value is either in engineering or physics units. The model flag returns either real or simulated values.

Parameters

- **field** (*str*) – requested field
- **handle** (*str*) – pytac.SP or pytac.RB
- **unit** (*str*) – pytac.ENG or pytac.PHYS returned.
- **model** (*str*) – pytac.LIVE or pytac.SIM

Returns value of the requested field

Return type Number

Raises DeviceException if there is no device on the given field

set_model(*model*, *model_type*)
Add a model to the element.

Parameters

- **model** (Model) – instance of Model

- **model_type** (*str*) – pytac.LIVE or pytac.SIM

set_value (*field*, *value*, *handle='setpoint'*, *units='engineering'*, *model='live'*)
Set the value on a uniquely identified device.

This value can be set on the machine or the simulation. A field is required to identify a device. Returned value can be engineering or physics.

Parameters

- **field** (*str*) – requested field
- **value** (*float*) – value to set
- **unit** (*str*) – pytac.ENG or pytac.PHYS
- **model** (*str*) – pytac.LIVE or pytac.SIM

Raises DeviceException if arguments are incorrect

2.2.4 pytac.lattice module

Representation of a lattice object which contains all the elements of the machine.

class pytac.lattice.**Lattice** (*name*, *control_system*, *energy*)
Bases: object

Representation of a lattice.

Represents a lattice object that contains all elements of the ring. It has a name and a control system to be used for unit conversion.

Parameters

- **name** (*str*) – The name of the lattice.
- **control_system** (ControlSystem) – The control system used to store the values on a pv.
- **energy** (*float*) – The total energy of the lattice.

add_element (*element*)
Append an element to the lattice.

Parameters **element** (Element) – element to append

get_all_families ()
Get all families of elements in the lattice

Returns all defined families

Return type set(str)

get_device_names (*family*, *field*)
Get the names for devices attached to a specific field for elements in the specfied family.

Typically all elements of a family will have devices associated with the same fields - for example, BPMs each have device for fields 'x' and 'y'.

Parameters

- **family** (*str*) – family of elements
- **field** (*str*) – field specifying the devices

Returns devices names for specified family and field

Return type list(str)

get_devices (*family*, *field*)
Get devices for a specific field for elements in the specfied family.

Typically all elements of a family will have devices associated with the same fields - for example, BPMs each have a device for fields 'x' and 'y'.

Parameters

- **family** (*str*) – family of elements
- **field** (*str*) – field specifying the devices

Returns devices for specified family and field

Return type list(devices)

get_elements (*family=None*, *cell=None*)
Get the elements of a family from the lattice.

If no family is specified it returns all elements.

Elements are returned in the order they exist in the ring.

Parameters

- **family** (*str*) – requested family
- **cell** (*int*) – restrict elements to those in the specified cell

Returns list containing all elements of the specified family

Return type list(*Element*)

get_energy ()
Function to get the total energy of the lattice.

Returns energy of the lattice

Return type float

get_family_s (*family*)
Get s positions for all elements from the same family.

Parameters **family** (*str*) – requested family

Returns list of s positions for each element

Return type list(float)

get_length ()
Returns the length of the lattice.

Returns The length of the lattice.

Return type float

get_pv_names (*family*, *field*, *handle*)
Get all pv names for a specific family, field and handle.

Parameters

- **family** (*str*) – requested family
- **field** (*str*) – requested field
- **handle** (*str*) – pytac.RB or pytac.SP

Returns list of pv names

Return type list(str)

get_s (*elem*)
Find the s position of an element in the lattice.

Note that the given element must exist in the lattice.

Parameters **elem** (Element) – The element that the position is being asked for.

Returns the position of the given element.

Return type float

Raises LatticeException: if element doesn't exist in the lattice.

get_values (*family*, *field*, *handle*, *dtype=None*)
Get all values for a family and field.

Parameters

- **family** (*str*) – family to request the values of
- **field** (*str*) – field to request values for
- **handle** (*str*) – pytac.RB or pytac.SP
- **dtype** (*numpy.dtype*) – if None, return a list. If not None, return a numpy array of the specified type.

Returns sequence of values

Return type list or numpy array

set_model (*model*)

set_values (*family*, *field*, *values*)
Sets the values for a family and field.

The pvs are determined by family and device. Note that only setpoint pvs can be modified.

Parameters

- **family** (*str*) – family on which to set values
- **field** (*str*) – field to set values for
- **values** (*sequence*) – A list of values to assign

Raises *LatticeException* – if the given list of values doesn't match the number of elements in the family

exception pytac.lattice.**LatticeException**
Bases: exceptions.Exception

2.2.5 pytac.load_csv module

Module to load the elements of the machine from csv files.

The csv files are stored in one directory with specified names:

- elements.csv
- devices.csv

- families.csv
- unitconv.csv
- uc_poly_data.csv
- uc_pchip_data.csv

pytac.load_csv.**get_div_rigidity**(*energy*)

pytac.load_csv.**get_mult_rigidity**(*energy*)

pytac.load_csv.**load**(*mode*, *control_system=None*, *directory=None*)
Load the elements of a lattice from a directory.

Parameters

- **mode** (*str*) – The name of the mode to be loaded.
- **control_system** (ControlSystem) – The control system to be used. If none is provided an EpicsControlSystem will be created.
- **directory** (*str*) – Directory where to load the files from. If no directory is given the data directory at the root of the repository is used.

Returns The lattice containing all elements.

Return type *Lattice*

pytac.load_csv.**load_pchip_unitconv**(*filename*)
Load pchip unit conversions from a csv file.

pytac.load_csv.**load_poly_unitconv**(*filename*)
Load polynomial unit conversions from a csv file.

pytac.load_csv.**load_unitconv**(*directory*, *mode*, *lattice*)
Load the unit conversion objects from a file.

Parameters

- **directory** (*str*) – The directory where the data is stored.
- **mode** (*str*) – The name of the mode that is used.
- **lattice** (Lattice) – The lattice object that will be used.

2.2.6 pytac.model module

Module containing pytac model classes.

class pytac.model.**DeviceModel**
Bases: object

Model containing control system devices.

add_device(*field*, *device*)
Add device to this model.

Parameters

- **field** (*str*) – field this device represents
- **device** (Device) – device object

get_device(*field*)
Get device from the model.

Parameters **field** (*str*) – field of the requested device

get_fields()

get_pv_name(*field*, *handle*)
Get PV name for a field and handle.

Parameters

- **field** (*str*) – field of the requested PV
- **handle** (*str*) – pytac.RB or pytac.SP

Returns pv name for specified field and handle

Return type str

get_value(*field*, *handle*)

set_value(*field*, *value*)

class pytac.model.**Model**
Bases: object

Abstract base classes for element models.

Typically an instance would represent hardware via a control system, or a simulation.

units
pytac.PHYS or pytac.ENG

Type str

get_fields()
Get all the fields represented by this model.

Returns all fields

Return type iterable

get_value(*field*, *handle*)
Get a value for a field.

Parameters

- **field** (*str*) – field of the requested value
- **handle** (*str*) – pytac.RB or pytac.SP

Returns value for specified field and handle

Return type float

set_value(*field*, *value*)
Set a value for a field.

This is always set to pytac.SP, never pytac.RB.

Parameters

- **field** (*str*) – field to set
- **value** (*str*) – value to set

2.2.7 pytac.units module

Classes for use in unit conversion.

class pytac.units.**PchipUnitConv**(*x*, *y*, *post_eng_to_phys=<function unit_function>*, *pre_phys_to_eng=<function unit_function>*)

Bases: *pytac.units.UnitConv*

PChip interpolation for converting between physics and engineering units.

Parameters

- **x** (*list*) – A list of points on the x axis. These must be in increasing order for the interpolation to work. Otherwise, a ValueError is raised.
- **y** (*list*) – A list of points on the y axis. These must be in increasing or decreasing order. Otherwise, a ValueError is raised.

Raises

- ValueError – An error occured when the given y coefficients are neither in
- increasing or decreasing order.

class pytac.units.**PolyUnitConv**(*coef*, *post_eng_to_phys=<function unit_function>*, *pre_phys_to_eng=<function unit_function>*)

Bases: *pytac.units.UnitConv*

Linear interpolation for converting between physics and engineering units.

Parameters **coef** (*array_like*) – The polynomial's coefficients, in decreasing powers.

class pytac.units.**UnitConv**(*post_eng_to_phys=<function unit_function>*, *pre_phys_to_eng=<function unit_function>*)

Bases: object

Class to convert between physics and engineering units.

This class does not do conversion but does return values if the target units are the same as the provided units. Subclasses should implement _raw_eng_to_phys() and _raw_phys_to_eng() in order to provide complete unit conversion.

The two arguments to this function represent functions that are applied to the result of the initial conversion. One happens after the conversion, the other happens before the conversion back.

Parameters

- **post_eng_to_phys** (*function*) – Function to be applied after the initial conversion.
- **pre_phys_to_eng** (*function*) – Function to be applied before the initial conversion.

convert(*value*, *origin*, *target*)

eng_to_phys(*value*)

Function that does the unit conversion.

Conversion from engineering to physics units. An additional function may be casted on the initial conversion.

Parameters **value** (*float*) – Value to be converted from engineering to physics units.

Returns The result value.

Return type result (float)

phys_to_eng(*value*)
Function that does the unit conversion.

Conversion from physics to engineering units. An additional function may be casted on the initial conversion.

Parameters **value** (*float*) – Value to be converted from physics to engineering units.

Returns The result value.

Return type result (float)

exception pytac.units.**UnitsException**
Bases: exceptions.Exception

pytac.units.**unit_function**(*value*)
Default value for the pre and post functions used in unit conversion.

Parameters **value** (*float*) – The value to be converted.

Returns The result of the conversion.

Return type value (float)

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

Index

N

P

S

T

U