docs Documentation

Release 1.0

Will Rogers, Razvan Vasile

Jul 10, 2019

Contents

1 Overview **3**

2 Contents: **5**

2.1 Examples . . . 5
- 2.1.1 Installation . . . 5
- 2.1.2 Initialisation . . . 5
- 2.1.3 Print BPM PV names along with s position . . . 6
- 2.1.4 Get the value of the 'b1' field of the quad elements . . . 6
- 2.1.5 Tutorial . . . 7

2.2 Developers . . . 7
- 2.2.1 Installation . . . 7
- 2.2.2 Initialisation . . . 7

2.3 API Documentation . . . 7
- 2.3.1 pytac.cs module . . . 8
- 2.3.2 pytac.data_source module . . . 9
- 2.3.3 pytac.device module . . . 13
- 2.3.4 pytac.element module . . . 16
- 2.3.5 pytac.exceptions module . . . 19
- 2.3.6 pytac.lattice module . . . 20
- 2.3.7 pytac.load_csv module . . . 26
- 2.3.8 pytac.units module . . . 27
- 2.3.9 pytac.utils module . . . 31

3 Indices and tables **33**

Python Module Index **35**

Index **37**

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

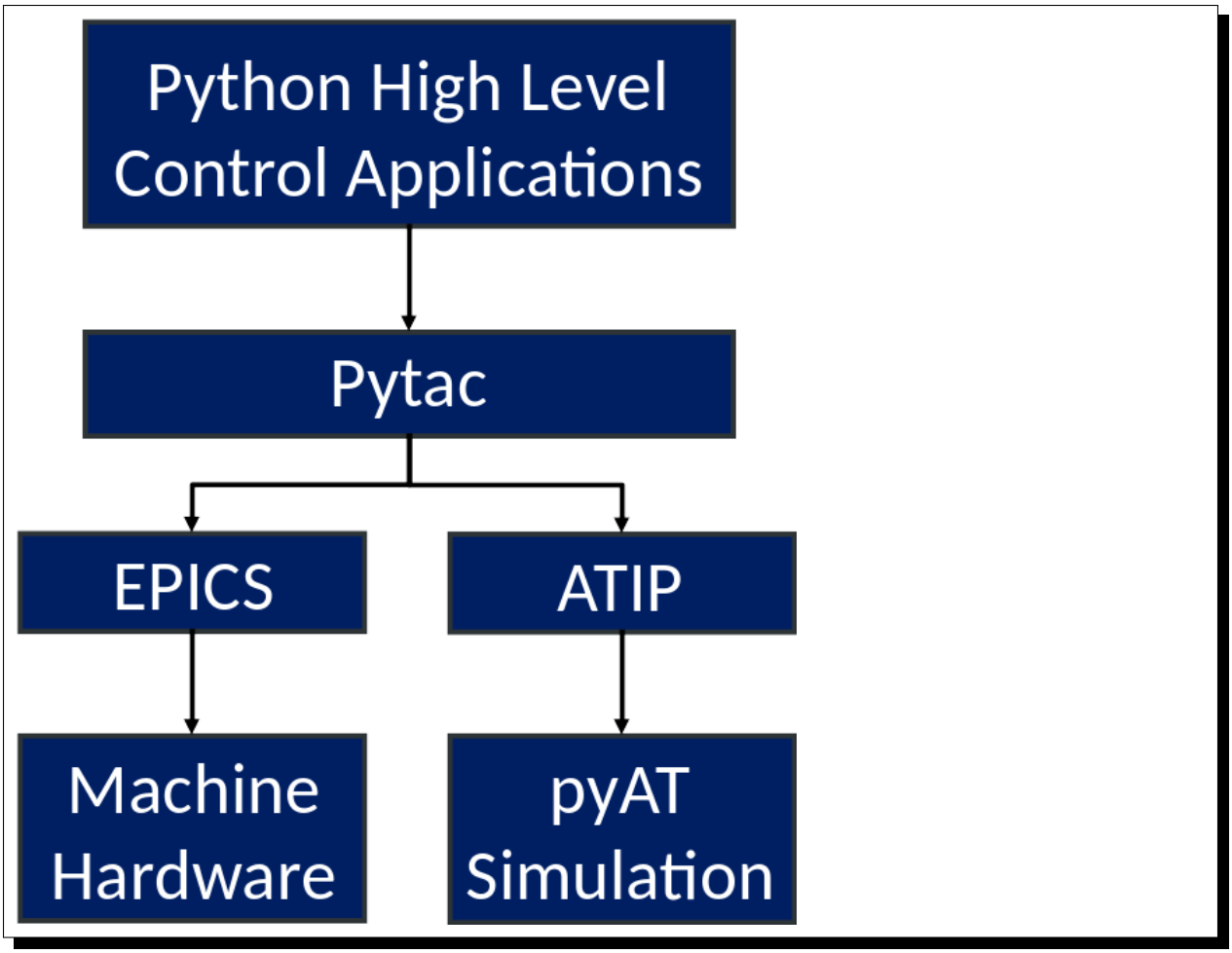
Overview

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

- Print the `QUAD` 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_values('QUAD', '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 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 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

Pytac: Python Toolkit for Accelerator Controls.

2.3.1 pytac.cs module

Class representing an abstract control system.

class pytac.cs.**ControlSystem**

Bases: object

Abstract base class representing a control system.

A specialised implementation of this class would be used to communicate over channel access with the hardware in the ring.

Methods:

get_multiple(*pvs*, *throw*)

Get the value for given PVs.

Parameters

- **pvs** (*sequence*) – PVs to get values of.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False None will be returned for any PV that fails and log a warning.

Returns the current values of the PVs.

Return type list(object)

Raises ControlSystemException – if it cannot connect to the specified PV.

get_single(*pv*, *throw*)

Get the value of a given PV.

Parameters

- **pv** (*string*) – PV to get the value of. readback or a setpoint PV.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False None will be returned for any PV that fails and log a warning.

Returns the current value of the given PV.

Return type object

Raises ControlSystemException – if it cannot connect to the specified PVs.

set_multiple(*pvs*, *values*, *throw*)

Set the values for given PVs.

Parameters

- **pvs** (*sequence*) – PVs to set the values of.
- **values** (*sequence*) – values to set no the PVs.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False return a list of True and False values corresponding to successes and failures and log a warning.

Raises

- ValueError – if the PVs or values are not passed in as sequences or if they have different lengths
- ControlSystemException – if it cannot connect to one or more PVs.

set_single(*pv*, *value*, *throw*)

Set the value of a given PV.

Parameters

- **pv** (*string*) – The PV to set the value of.
- **value** (*object*) – The value to set the PV to.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False log a warning.

Raises `ControlSystemException` – if it cannot connect to the specified PV.

2.3.2 pytac.data_source module

Module containing pytac data source classes.

class `pytac.data_source.`**DataSource**

Bases: `object`

Abstract base class for element or lattice data sources.

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

Attributes:

units

pytac.PHYS or pytac.ENG.

Type str

Methods:

get_fields()

Get all the fields represented by this data source.

Returns all fields.

Return type iterable

get_value(*field*, *handle*, *throw*)

Get a value for a field.

Parameters

- **field** (*str*) – field of the requested value.
- **handle** (*str*) – pytac.RB or pytac.SP
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False None will be returned for any PV that fails and log a warning.

Returns value for specified field and handle.

Return type float

set_value(*field*, *value*, *throw*)

Set a value for a field.

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

Parameters

- **field** (*str*) – field to set.
- **value** (*float*) – value to set.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False log a warning.

class pytac.data_source.**DataSourceManager**

Bases: object

Class that manages all the data sources and UnitConv objects associated with a lattice or element.

It receives requests from a lattice or element object and directs them to the correct data source. The unit conversion objects for all fields are also held here.

default_units

Holds the current default unit type, pytac.PHYS or pytac.ENG, for an element or lattice.

Type str

default_data_source

Holds the current default data source, pytac.LIVE or pytac.SIM, for an element or lattice.

Type str

Methods:

add_device (*field*, *device*, *uc*)

Add device and unit conversion objects to a given field.

A DeviceDataSource must be set before calling this method, this defaults to pytac.LIVE as that is the only data source that currently uses devices.

Parameters

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

Raises DataSourceException – if no DeviceDataSource is set.

get_device (*field*)

Get the device for the given field.

A DeviceDataSource must be set before calling this method, this defaults to pytac.LIVE as that is the only data source that currently uses devices.

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

Returns The device on the given field.

Return type *Device*

Raises DataSourceException – if no DeviceDataSource is set.

get_fields ()

Get all the fields defined on the manager.

Includes all fields defined by all data sources.

Returns

A dictionary of all the fields defined on the manager, separated by data source(key).

Return type dict

get_unitconv (*field*)

Get the unit conversion option for the specified field.

Parameters **field** (*str*) – The field associated with this conversion.

Returns The object associated with the specified field.

Return type *UnitConv*

Raises `FieldException` – if no unit conversion object is present.

get_value (*field*, *handle='readback'*, *units='default'*, *data_source='default'*, *throw=True*)

Get the value for a field.

Returns the value of a field on the manager. This value is uniquely identified by a field and a handle. The returned value is either in engineering or physics units. The data_source flag returns either real or simulated values. If handle, units or data_source are not given then the lattice default values are used.

Parameters

- **field** (*str*) – The requested field.
- **handle** (*str*) – pytac.SP or pytac.RB.
- **units** (*str*) – pytac.ENG or pytac.PHYS returned.
- **data_source** (*str*) – pytac.LIVE or pytac.SIM.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False None will be returned for any PV that fails and log a warning.

Returns The value of the requested field

Return type float

Raises

- `DataSourceException` – if there is no data source on the given field.
- `FieldException` – if the manager does not have the specified field.

set_data_source (*data_source*, *data_source_type*)

Add a data source to the manager.

Parameters

- **data_source** (`DataSource`) – the data source to be set.
- **data_source_type** (*str*) – the type of the data source being set pytac.LIVE or pytac.SIM.

set_unitconv (*field*, *uc*)

set the unit conversion option for the specified field.

Parameters

- **field** (*str*) – The field associated with this conversion.
- **uc** (`UnitConv`) – The unit conversion object to be set.

set_value (*field*, *value*, *handle='setpoint'*, *units='default'*, *data_source='default'*, *throw=True*)

Set the value for a field.

This sets a value on the machine or the simulation. If handle,units or data_source are not given then the lattice default values are used.

Parameters

- **field** (*str*) – The requested field.
- **value** (*float*) – The value to set.
- **handle** (*str*) – pytac.SP or pytac.RB.
- **units** (*str*) – pytac.ENG or pytac.PHYS.

- **data_source** (*str*) – pytac.LIVE or pytac.SIM.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False log a warning.

Raises

- `HandleException` – if the specified handle is not pytac.SP.
- `DataSourceException` – if arguments are incorrect.
- `FieldException` – if the manager does not have the specified field.

class pytac.data_source.**DeviceDataSource**

Bases: *pytac.data_source.DataSource*

Data source containing control system devices.

Attributes:

units

pytac.ENG or pytac.PHYS, pytac.ENG by default.

Type str

Methods:

add_device (*field*, *device*)

Add device to this data_source.

Parameters

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

get_device (*field*)

Get device from the data_source.

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

Returns The device of the specified field.

Return type *Device*

Raises `FieldException` – if the specified field doesn't exist on this data source.

get_fields ()

Get all the fields from the data_source.

Returns list of strings of all the fields of the data_source.

Return type list

get_value (*field*, *handle*, *throw=True*)

Get the value of a readback or setpoint PV for a field from the data_source.

Parameters

- **field** (*str*) – field of the requested value.
- **handle** (*str*) – pytac.RB or pytac.SP.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False None will be returned for any PV that fails and log a warning.

Returns The value of the PV.

Return type float

Raises `FieldException` – if the device does not have the specified field.

set_value(*field*, *value*, *throw=True*)
Set the value of a readback or setpoint PV for a field from the data_source.

Parameters

- **field** (*str*) – field for the requested value.
- **value** (*float*) – The value to set on the PV.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False log a warning.

Raises `FieldException` – if the device does not have the specified field.

2.3.3 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.**BasicDevice**(*value*, *enabled=True*)
Bases: *pytac.device.Device*

A basic implementation of the device class.

This device does not have a PV associated with it, nor does it interact with a simulator. In short this device acts as simple storage for data that rarely changes, as it is not affected by changes to other aspects of the accelerator.

Parameters

- **value** (*numeric*) – can be a number or a list of numbers.
- **enabled** (*bool-like*) – Whether the device is enabled. May be a PvEnabler object.

get_value(*handle=None*, *throw=None*)
Read the value from the device.

Parameters

- **handle** (*str*) – Irrelevant in this case as a control system is not used, only supported to conform with the base class.
- **throw** (*bool*) – Irrelevant in this case as a control system is not used, only supported to conform with the base class.

Returns the value of the device.

Return type numeric

is_enabled()
Whether the device is enabled.

Returns whether the device is enabled.

Return type bool

set_value(*value*, *throw=None*)
Set the value on the device.

Parameters

- **value** (*numeric*) – the value to set.

- **throw** (*bool*) – Irrelevant in this case as a control system is not used, only supported to conform with the base class.

class pytac.device.**Device**

Bases: object

A representation of a property of an element associated with a field.

Typically a control system will be used to set and get values on a device.

Methods:

get_value(*handle*, *throw*)

Read the value from the device.

Parameters

- **handle** (*str*) – pytac.SP or pytac.RB.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False None will be returned for any PV that fails and log a warning.

Returns the value of the PV.

Return type float

is_enabled()

Whether the device is enabled.

Returns whether the device is enabled.

Return type bool

set_value(*value*, *throw*)

Set the value on the device.

Parameters

- **value** (*float*) – the value to set.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False log a warning.

class pytac.device.**EpicsDevice**(*name*, *cs*, *enabled=True*, *rb_pv=None*, *sp_pv=None*)

Bases: *pytac.device.Device*

An EPICS-aware device.

Contains a control system, readback and setpoint PVs. A readback or setpoint PV is required when creating an epics device otherwise a DataSourceException is raised. The device is enabled by default.

Attributes:

name

The prefix of EPICS PVs for this device.

Type str

rb_pv

The EPICS readback PV.

Type str

sp_pv

The EPICS setpoint PV.

Type str

Parameters

- **name** (*str*) – The prefix of EPICS PV for this device.
- **cs** (ControlSystem) – The 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_pv** (*str*) – The EPICS readback PV.
- **sp_pv** (*str*) – The EPICS setpoint PV.

Raises DataSourceException – if no PVs are provided.

Methods:

get_pv_name(*handle*)
Get the PV name for the specified handle.

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

Returns A readback or setpoint PV.

Return type str

Raises HandleException – if the PV doesn't exist.

get_value(*handle*, *throw=True*)
Read the value of a readback or setpoint PV.

Parameters

- **handle** (*str*) – pytac.SP or pytac.RB.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False None will be returned for any PV that fails and log a warning.

Returns The value of the PV.

Return type float

Raises HandleException – if the requested PV doesn't exist.

is_enabled()
Whether the device is enabled.

Returns whether the device is enabled.

Return type bool

set_value(*value*, *throw=True*)
Set the device value.

Parameters

- **value** (*float*) – The value to set.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False log a warning.

Raises HandleException – if no setpoint PV exists.

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*) – The PV name.
- **enabled_value** (*str*) – The value for PV for which the device should be considered enabled.
- **cs** (ControlSystem) – The control system object.

Methods:

2.3.4 pytac.element module

Module containing the element class.

class pytac.element.**Element** (*length*, *element_type*, *name=None*, *lattice=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 each of its fields (e.g. 'b1' for a quadrupole).

Attributes:

name

The name identifying the element.

Type str

type_

The type of the element.

Type str

length

The length of the element in metres.

Type float

families

The families this element is a member of.

Type set

Parameters

- **length** (*float*) – The length of the element.
- **element_type** (*str*) – The type of the element.
- **name** (*str*) – The unique identifier for the element in the ring.
- **lattice** (Lattice) – The lattice to which the element belongs.

Methods:

add_device (*field*, *device*, *uc*)

Add device and unit conversion objects to a given field.

A DeviceDataSource must be set before calling this method, this defaults to pytac.LIVE as that is the only data source that currently uses devices.

Parameters

- **field** (*str*) – The key to store the unit conversion and device objects.

- **device** (Device) – The device object used for this field.
- **uc** (UnitConv) – The unit conversion object used for this field.

Raises DataSourceException – if no DeviceDataSource is set.

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

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

cell
The lattice cell this element is within.

N.B. If the element spans multiple cells then the cell it begins in is returned (lowest cell number).

Type int

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

A DeviceDataSource must be set before calling this method, this defaults to pytac.LIVE as that is the only data source that currently uses devices.

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

Returns The device on the given field.

Return type *Device*

Raises DataSourceException – if no DeviceDataSource is set.

get_fields ()
Get the all fields defined on an element.

Includes all fields defined by all data sources.

Returns

A dictionary of all the fields defined on an element, separated by data source(key).

Return type dict

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

Parameters **field** (*str*) – The field associated with this conversion.

Returns The object associated with the specified field.

Return type *UnitConv*

Raises FieldException – if no unit conversion object is present.

get_value (*field*, *handle='readback'*, *units='default'*, *data_source='default'*, *throw=True*)
Get the value for a field.

Returns the value of 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 data_source flag returns either real or simulated values.

Parameters

- **field** (*str*) – The requested field.
- **handle** (*str*) – pytac.SP or pytac.RB.
- **units** (*str*) – pytac.ENG or pytac.PHYS returned.

- **data_source** (*str*) – pytac.LIVE or pytac.SIM.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False None will be returned for any PV that fails and log a warning.

Returns The value of the requested field

Return type float

Raises

- `DataSourceException` – if there is no data source on the given field.
- `FieldException` – if the element does not have the specified field.

index

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

Type int

s

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

Type float

set_data_source (*data_source*, *data_source_type*)

Add a data source to the element.

Parameters

- **data_source** (`DataSource`) – the data source to be set.
- **data_source_type** (*str*) – the type of the data source being set pytac.LIVE or pytac.SIM.

set_lattice (*lattice*)

Set the stored lattice reference for this element to the passed lattice object.

Parameters **lattice** (`Lattice`) – lattice object to store a reference to.

set_unitconv (*field*, *uc*)

Set the unit conversion option for the specified field.

Parameters

- **field** (*str*) – The field associated with this conversion.
- **uc** (`UnitConv`) – The unit conversion object to be set.

set_value (*field*, *value*, *handle='setpoint'*, *units='default'*, *data_source='default'*, *throw=True*)

Set the value for a field.

This value can be set on the machine or the simulation.

Parameters

- **field** (*str*) – The requested field.
- **value** (*float*) – The value to set.
- **handle** (*str*) – pytac.SP or pytac.RB.
- **units** (*str*) – pytac.ENG or pytac.PHYS.
- **data_source** (*str*) – pytac.LIVE or pytac.SIM.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False log a warning.

Raises

- `DataSourceException` – if arguments are incorrect.
- `FieldException` – if the element does not have the specified field.

class `pytac.element.`**`EpicsElement`**(*length*, *element_type*, *name=None*, *lattice=None*)

Bases: *`pytac.element.Element`*

EPICS-aware element.

Adds get_pv_name() method.

Methods:

Parameters

- **`length`** (*`float`*) – The length of the element.
- **`element_type`** (*`str`*) – The type of the element.
- **`name`** (*`str`*) – The unique identifier for the element in the ring.
- **`lattice`** (`Lattice`) – The lattice to which the element belongs.

Methods:

`get_pv_name`(*field*, *handle*)

Get PV name for the specified field and handle.

Parameters

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

Returns The readback or setpoint PV for the specified field.

Return type str

Raises

- `DataSourceException` – if there is no data source for this field.
- `FieldException` – if the specified field doesn't exist.

2.3.5 pytac.exceptions module

Module containing all the exceptions used in pytac.

exception `pytac.exceptions.`**`ControlSystemException`**

Bases: `exceptions.Exception`

Exception associated with control system misconfiguration.

exception `pytac.exceptions.`**`DataSourceException`**

Bases: `exceptions.Exception`

Exception associated with Device misconfiguration or invalid requests to a data source.

exception `pytac.exceptions.`**`FieldException`**

Bases: `exceptions.Exception`

Exception associated with invalid field requests.

exception `pytac.exceptions.`**`HandleException`**

Bases: `exceptions.Exception`

Exception associated with requests with invalid handles.

exception pytac.exceptions.**UnitsException**

Bases: exceptions.Exception

Conversion not understood.

2.3.6 pytac.lattice module

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

class pytac.lattice.**EpicsLattice**(*name*, *epics_cs*, *symmetry=None*)

Bases: *pytac.lattice.Lattice*

EPICS-aware lattice class.

Allows efficient get_values() and set_values() methods, and adds get_pv_names() method.

Attributes:

name

The name of the lattice.

Type str

symmetry

The symmetry of the lattice (the number of cells).

Type int

Parameters

- **name** (*str*) – The name of the epics lattice.
- **epics_cs** (ControlSystem) – The control system used to store the values on a PV.
- **symmetry** (*int*) – The symmetry of the lattice (the number of cells).

Methods:

get_element_pv_names(*family*, *field*, *handle*)

Get the PV names for the given field, and handle, on all elements in the given family in the lattice.

Assume that the elements are EpicsElements that have the get_pv_name() method.

Parameters

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

Returns A list of PV names, strings.

Return type list

get_element_values(*family*, *field*, *handle='readback'*, *units='default'*, *data_source='default'*, *dtype=None*)

Get the value of the given field for all elements in the given family in the lattice.

Parameters

- **family** (*str*) – family of elements to request the values of.
- **field** (*str*) – field to request values for.
- **handle** (*str*) – pytac.RB or pytac.SP.

- **units** (*str*) – pytac.ENG or pytac.PHYS.
- **data_source** (*str*) – pytac.LIVE or pytac.SIM.
- **dtype** (*numpy.dtype*) – if None, return a list. If not None, return a numpy array of the specified type.

Returns The requested values.

Return type list or numpy.array

get_pv_name (*field*, *handle*)

Get the PV name for a specific field, and handle on this lattice.

Parameters

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

Returns The readback or setpoint PV for the specified field.

Return type str

set_element_values (*family*, *field*, *values*, *handle='setpoint'*, *units='default'*, *data_source='default'*)

Set the value of the given field for all elements in the given family in the lattice to the given values.

Parameters

- **family** (*str*) – family of elements on which to set values.
- **field** (*str*) – field to set values for.
- **values** (*sequence*) – A list of values to assign.
- **handle** (*str*) – pytac.SP or pytac.RB.
- **units** (*str*) – pytac.ENG or pytac.PHYS.
- **data_source** (*str*) – pytac.LIVE or pytac.SIM.

Raises `IndexError` – if the given list of values doesn't match the number of elements in the family.

class pytac.lattice.**Lattice** (*name*, *symmetry=None*)

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.

Attributes:

name

The name of the lattice.

Type str

symmetry

The symmetry of the lattice (the number of cells).

Type int

Parameters

- **name** (*str*) – The name of the lattice.

- **symmetry** (*int*) – The symmetry of the lattice (the number of cells).

Methods:

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

A DeviceDataSource must be set before calling this method, this defaults to pytac.LIVE as that is the only data source that currently uses devices.

Parameters

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

Raises DataSourceException – if no DeviceDataSource is set.

add_element (*element*)
Append an element to the lattice and update its lattice reference.

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

cell_bounds
The indexes of elements in which a cell boundary occurs.

Examples

A lattice of 5 equal length elements with 2 fold symmetry would return [1, 4, 5] 1 - because it is the start of the first cell. 4 - because it is the first element in the second cell as the boundary between the first and second cells occurs halfway into the length of element 3. 5 - (len(lattice)) because it is the end of the second (last) cell.

Type list (str)

cell_length
The average length of a cell in the lattice.

Type float

convert_family_values (*family*, *field*, *values*, *origin*, *target*)
Convert the given values according to the given origin and target units, using the unit conversion objects for the given field on the elements in the given family.

Parameters

- **family** (*str*) – the family of elements which the values belong to.
- **field** (*str*) – the field on the elements which the values are from.
- **values** (*sequence*) – values to be converted.
- **origin** (*str*) – pytac.ENG or pytac.PHYS.
- **target** (*str*) – pytac.ENG or pytac.PHYS.

get_all_families ()
Get all families of elements in the lattice.

Returns all defined families.

Return type set

get_default_data_source()
Get the default data source, pytac.LIVE or pytac.SIM.

Returns the default data source for the entire lattice.

Return type str

get_default_units()
Get the default unit type, pytac.ENG or pytac.PHYS.

Returns the default unit type for the entire lattice.

Return type str

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

A DeviceDataSource must be set before calling this method, this defaults to pytac.LIVE as that is the only data source that currently uses devices.

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

Returns The device on the given field.

Return type *Device*

Raises `DataSourceException` – if no DeviceDataSource is set.

get_element_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 a device for fields 'x' and 'y'.

Parameters

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

Returns device names for specified family and field.

Return type list

get_element_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

get_element_values(*family*, *field*, *handle='readback'*, *units='default'*, *data_source='default'*, *dtype=None*)
Get the value of the given field for all elements in the given family in the lattice.

Parameters

- **family** (*str*) – family of elements to request the values of.
- **field** (*str*) – field to request values for.

- **handle** (*str*) – pytac.RB or pytac.SP.
- **units** (*str*) – pytac.ENG or pytac.PHYS.
- **data_source** (*str*) – pytac.LIVE or pytac.SIM.
- **dtype** (*numpy.dtype*) – if None, return a list. If not None, return a numpy array of the specified type.

Returns The requested values.

Return type list or numpy.array

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

Raises `ValueError` – if there are no elements in the specified cell or family.

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

get_fields ()

Get the fields defined on the lattice.

Includes all fields defined by all data sources.

Returns

A dictionary of all the fields defined on the lattice, separated by data source(key).

Return type dict

get_length ()

Returns the length of the lattice, in meters.

Returns The length of the lattice (m).

Return type float

get_unitconv (*field*)

Get the unit conversion option for the specified field.

Parameters **field** (*str*) – The field associated with this conversion.

Returns The object associated with the specified field.

Return type *UnitConv*

Raises `FieldException` – if no unit conversion object is present.

get_value (*field*, *handle='readback'*, *units='default'*, *data_source='default'*, *throw=True*)
Get the value for a field on the lattice.

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

Parameters

- **field** (*str*) – The requested field.
- **handle** (*str*) – pytac.SP or pytac.RB.
- **units** (*str*) – pytac.ENG or pytac.PHYS returned.
- **data_source** (*str*) – pytac.LIVE or pytac.SIM.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False None will be returned for any PV that fails and log a warning.

Returns The value of the requested field

Return type float

Raises

- `DataSourceException` – if there is no data source on the given field.
- `FieldException` – if the lattice does not have the specified field.

set_data_source (*data_source*, *data_source_type*)
Add a data source to the lattice.

Parameters

- **data_source** (`DataSource`) – the data source to be set.
- **data_source_type** (*str*) – the type of the data source being set pytac.LIVE or pytac.SIM.

set_default_data_source (*default_ds*)
Sets the default data source for the lattice and all its elements.

Parameters **default_ds** (*str*) – The default data source to be set across the entire lattice, pytac.LIVE or pytac.SIM.

Raises `DataSourceException` – if specified default data source is not a valid data source.

set_default_units (*default_units*)
Sets the default unit type for the lattice and all its elements.

Parameters **default_units** (*str*) – The default unit type to be set across the entire lattice, pytac.ENG or pytac.PHYS.

Raises `UnitsException` – if specified default unit type is not a valid unit type.

set_element_values (*family*, *field*, *values*, *handle='setpoint'*, *units='default'*, *data_source='default'*)
Set the value of the given field for all elements in the given family in the lattice to the given values.

Parameters

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

- **handle** (*str*) – pytac.SP or pytac.RB.
- **units** (*str*) – pytac.ENG or pytac.PHYS.
- **data_source** (*str*) – pytac.LIVE or pytac.SIM.

Raises `IndexError` – if the given list of values doesn't match the number of elements in the family.

set_unitconv (*field*, *uc*)
Set the unit conversion option for the specified field.

Parameters

- **field** (*str*) – The field associated with this conversion.
- **uc** (`UnitConv`) – The unit conversion object to be set.

set_value (*field*, *value*, *handle='setpoint'*, *units='default'*, *data_source='default'*, *throw=True*)
Set the value for a field.

This value can be set on the machine or the simulation.

Parameters

- **field** (*str*) – The requested field.
- **value** (*float*) – The value to set.
- **handle** (*str*) – pytac.SP or pytac.RB.
- **units** (*str*) – pytac.ENG or pytac.PHYS.
- **data_source** (*str*) – pytac.LIVE or pytac.SIM.
- **throw** (*bool*) – On failure, if True raise ControlSystemException, if False log a warning.

Raises

- `DataSourceException` – if arguments are incorrect.
- `FieldException` – if the lattice does not have the specified field.

2.3.7 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.`**load** (*mode*, *control_system=None*, *directory=None*, *symmetry=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.
- **symmetry** (*int*) – The symmetry of the lattice (the number of cells).

Returns The lattice containing all elements.

Return type *Lattice*

Raises ControlSystemException – if the default control system, cothread, is not installed.

pytac.load_csv.**load_pchip_unitconv**(*filename*)

Load pchip unit conversions from a csv file.

Parameters **filename** (*path-like object*) – The pathname of the file from which to load the pchip unit conversions.

Returns A dictionary of the unit conversions.

Return type dict

pytac.load_csv.**load_poly_unitconv**(*filename*)

Load polynomial unit conversions from a csv file.

Parameters **filename** (*path-like object*) – The pathname of the file from which to load the polynomial unit conversions.

Returns A dictionary of the unit conversions.

Return type dict

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.

2.3.8 pytac.units module

Classes for use in unit conversion.

class pytac.units.**NullUnitConv**(*engineering_units=''*, *physics_units=''*)

Bases: *pytac.units.UnitConv*

Returns input value without performing any conversions.

Attributes:

eng_units

The unit type of the post conversion engineering value.

Type str

phys_units

The unit type of the post conversion physics value.

Type str

Parameters

- **engineering_units** (*str*) – The unit type of the post conversion engineering value.
- **physics_units** (*str*) – The unit type of the post conversion physics value.

class pytac.units.**PchipUnitConv**(*x*, *y*, *post_eng_to_phys=<function unit_function>*, *pre_phys_to_eng=<function unit_function>*, *engineering_units=''*, *physics_units=''*, *name=None*)

Bases: *pytac.units.UnitConv*

Piecewise Cubic Hermite Interpolating Polynomial unit conversion.

Attributes:

x
A list of points on the x axis. These must be in increasing order for the interpolation to work. Otherwise, a ValueError is raised.

Type list

y
A list of points on the y axis. These must be in increasing or decreasing order. Otherwise, a ValueError is raised.

Type list

pp
A pchip one-dimensional monotonic cubic interpolation of points on both x and y axes.

Type PchipInterpolator

name
An identifier for the unit conversion object.

Type str

eng_units
The unit type of the post conversion engineering value.

Type str

phys_units
The unit type of the post conversion physics value.

Type str

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.
- **engineering_units** (*str*) – The unit type of the post conversion engineering value.
- **physics_units** (*str*) – The unit type of the post conversion physics value.
- **name** (*str*) – An identifier for the unit conversion object.

Raises ValueError – if coefficients are not appropriately monotonic.

class pytac.units.**PolyUnitConv**(*coef*, *post_eng_to_phys=<function unit_function>*, *pre_phys_to_eng=<function unit_function>*, *engineering_units=''*, *physics_units=''*, *name=None*)

Bases: *pytac.units.UnitConv*

Linear interpolation for converting between physics and engineering units.

Attributes:

p
A one-dimensional polynomial of coefficients.

Type poly1d

name
An identifier for the unit conversion object.

Type str

eng_units
The unit type of the post conversion engineering value.

Type str

phys_units
The unit type of the post conversion physics value.

Type str

Parameters

- **coef** (*array-like*) – The polynomial's coefficients, in decreasing powers.
- **post_eng_to_phys** (*float*) – The value after conversion between ENG and PHYS.
- **pre_eng_to_phys** (*float*) – The value before conversion.
- **engineering_units** (*str*) – The unit type of the post conversion engineering value.
- **physics_units** (*str*) – The unit type of the post conversion physics value.
- **name** (*str*) – An identifier for the unit conversion object.

class pytac.units.**UnitConv**(*post_eng_to_phys=<function unit_function>*, *pre_phys_to_eng=<function unit_function>*, *engineering_units=''*, *physics_units=''*, *name=None*)

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.

Attributes:

name
An identifier for the unit conversion object.

Type str

eng_units
The unit type of the post conversion engineering value.

Type str

phys_units
The unit type of the post conversion physics value.

Type str

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.
- **engineering_units** (*str*) – The unit type of the post conversion engineering value.
- **physics_units** (*str*) – The unit type of the post conversion physics value.
- **name** (*str*) – An identifier for the unit conversion object.

Methods:

convert (*value*, *origin*, *target*)
Convert between two different unit types and chek the validity of the result.

Parameters

- **value** (*float*) – the value to be converted
- **origin** (*str*) – pytac.ENG or pytac.PHYS
- **target** (*str*) – pytac.ENG or pytac.PHYS

Returns The resulting value.

Return type float

Raises `UnitsException` – If the conversion is invalid; i.e. if there are no solutions, or multiple, within conversion limits.

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

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

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

Returns The result value.

Return type float

Raises `UnitsException` – If the conversion is invalid; i.e. if there are no solutions, or multiple, within conversion limits.

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

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

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

Returns The result value.

Return type float

Raises `UnitsException` – If the conversion is invalid; i.e. if there are no solutions, or multiple, within conversion limits.

set_conversion_limits(*lower_limit*, *upper_limit*)
Conversion limits to be applied before or after a conversion take place. Limits should be set in in engineering units.

Parameters

- **lower_limit** (*float*) – the lower conversion limit
- **upper_limit** (*float*) – the upper conversion limit

set_post_eng_to_phys(*post_eng_to_phys*)
Set the function to be applied after the initial conversion. :param post_eng_to_phys: Function to be applied after the

initial conversion.

set_pre_phys_to_eng(*pre_phys_to_eng*)
Set the function to be applied before the initial conversion. :param pre_phys_to_eng: Function to be applied before the

initial conversion.

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 float

2.3.9 pytac.utils module

Utility functions.

pytac.utils.**get_div_rigidity**(*energy*)

Parameters **energy** (*int*) – the energy of the lattice.

Returns div rigidity.

Return type function

pytac.utils.**get_mult_rigidity**(*energy*)

Parameters **energy** (*int*) – the energy of the lattice.

Returns mult rigidity.

Return type function

pytac.utils.**get_rigidity**(*energy_mev*)

Parameters **energy_mev** (*int*) – the energy of the lattice.

Returns p devided by the elementary charge.

Return type float

CHAPTER 3

Indices and tables

- genindex
- modindex
- search

Python Module Index

p

pytac, 7
pytac.cs, 8
pytac.data_source, 9
pytac.device, 13
pytac.element, 16
pytac.exceptions, 19
pytac.lattice, 20
pytac.load_csv, 26
pytac.units, 27
pytac.utils, 31

Index

A

add_device() (*pytac.data_source.DataSourceManager method*), 10
add_device() (*pytac.data_source.DeviceDataSource method*), 12
add_device() (*pytac.element.Element method*), 16
add_device() (*pytac.lattice.Lattice method*), 22
add_element() (*pytac.lattice.Lattice method*), 22
add_to_family() (*pytac.element.Element method*), 17

B

BasicDevice (*class in pytac.device*), 13

C

cell (*pytac.element.Element attribute*), 17
cell_bounds (*pytac.lattice.Lattice attribute*), 22
cell_length (*pytac.lattice.Lattice attribute*), 22
ControlSystem (*class in pytac.cs*), 8
ControlSystemException, 19
convert() (*pytac.units.UnitConv method*), 30
convert_family_values() (*pytac.lattice.Lattice method*), 22

D

DataSource (*class in pytac.data_source*), 9
DataSourceException, 19
DataSourceManager (*class in pytac.data_source*), 9
default_data_source (*pytac.data_source.DataSourceManager attribute*), 10
default_units (*pytac.data_source.DataSourceManager attribute*), 10
Device (*class in pytac.device*), 14
DeviceDataSource (*class in pytac.data_source*), 12

E

Element (*class in pytac.element*), 16
eng_to_phys() (*pytac.units.UnitConv method*), 30
eng_units (*pytac.units.NullUnitConv attribute*), 27
eng_units (*pytac.units.PchipUnitConv attribute*), 28
eng_units (*pytac.units.PolyUnitConv attribute*), 29
eng_units (*pytac.units.UnitConv attribute*), 29
EpicsDevice (*class in pytac.device*), 14
EpicsElement (*class in pytac.element*), 19
EpicsLattice (*class in pytac.lattice*), 20

F

families (*pytac.element.Element attribute*), 16
FieldException, 19

G

get_all_families() (*pytac.lattice.Lattice method*), 22
get_default_data_source() (*pytac.lattice.Lattice method*), 22
get_default_units() (*pytac.lattice.Lattice method*), 23
get_device() (*pytac.data_source.DataSourceManager method*), 10
get_device() (*pytac.data_source.DeviceDataSource method*), 12
get_device() (*pytac.element.Element method*), 17
get_device() (*pytac.lattice.Lattice method*), 23
get_div_rigidity() (*in module pytac.utils*), 31
get_element_device_names() (*pytac.lattice.Lattice method*), 23
get_element_devices() (*pytac.lattice.Lattice method*), 23
get_element_pv_names() (*pytac.lattice.EpicsLattice method*), 20
get_element_values() (*pytac.lattice.EpicsLattice method*), 20
get_element_values() (*pytac.lattice.Lattice method*), 23
get_elements() (*pytac.lattice.Lattice method*), 24
get_family_s() (*pytac.lattice.Lattice method*), 24

get_fields() (*pytac.data_source.DataSource method*), 9
get_fields() (*pytac.data_source.DataSourceManager method*), 10
get_fields() (*pytac.data_source.DeviceDataSource method*), 12
get_fields() (*pytac.element.Element method*), 17
get_fields() (*pytac.lattice.Lattice method*), 24
get_length() (*pytac.lattice.Lattice method*), 24
get_mult_rigidity() (*in module pytac.utils*), 31
get_multiple() (*pytac.cs.ControlSystem method*), 8
get_pv_name() (*pytac.device.EpicsDevice method*), 15
get_pv_name() (*pytac.element.EpicsElement method*), 19
get_pv_name() (*pytac.lattice.EpicsLattice method*), 21
get_rigidity() (*in module pytac.utils*), 31
get_single() (*pytac.cs.ControlSystem method*), 8
get_unitconv() (*pytac.data_source.DataSourceManager method*), 10
get_unitconv() (*pytac.element.Element method*), 17
get_unitconv() (*pytac.lattice.Lattice method*), 24
get_value() (*pytac.data_source.DataSource method*), 9
get_value() (*pytac.data_source.DataSourceManager method*), 11
get_value() (*pytac.data_source.DeviceDataSource method*), 12
get_value() (*pytac.device.BasicDevice method*), 13
get_value() (*pytac.device.Device method*), 14
get_value() (*pytac.device.EpicsDevice method*), 15
get_value() (*pytac.element.Element method*), 17
get_value() (*pytac.lattice.Lattice method*), 24

H

HandleException, 19

I

index (*pytac.element.Element attribute*), 18
is_enabled() (*pytac.device.BasicDevice method*), 13
is_enabled() (*pytac.device.Device method*), 14
is_enabled() (*pytac.device.EpicsDevice method*), 15

L

Lattice (*class in pytac.lattice*), 21
length (*pytac.element.Element attribute*), 16
load() (*in module pytac.load_csv*), 26
load_pchip_unitconv() (*in module pytac.load_csv*), 27
load_poly_unitconv() (*in module pytac.load_csv*), 27
load_unitconv() (*in module pytac.load_csv*), 27

N

name (*pytac.device.EpicsDevice attribute*), 14
name (*pytac.element.Element attribute*), 16
name (*pytac.lattice.EpicsLattice attribute*), 20
name (*pytac.lattice.Lattice attribute*), 21
name (*pytac.units.PchipUnitConv attribute*), 28
name (*pytac.units.PolyUnitConv attribute*), 29
name (*pytac.units.UnitConv attribute*), 29
NullUnitConv (*class in pytac.units*), 27

P

p (*pytac.units.PolyUnitConv attribute*), 29
PchipUnitConv (*class in pytac.units*), 28
phys_to_eng() (*pytac.units.UnitConv method*), 30
phys_units (*pytac.units.NullUnitConv attribute*), 27
phys_units (*pytac.units.PchipUnitConv attribute*), 28
phys_units (*pytac.units.PolyUnitConv attribute*), 29
phys_units (*pytac.units.UnitConv attribute*), 30
PolyUnitConv (*class in pytac.units*), 28
pp (*pytac.units.PchipUnitConv attribute*), 28
PvEnabler (*class in pytac.device*), 15
pytac (*module*), 7
pytac.cs (*module*), 8
pytac.data_source (*module*), 9
pytac.device (*module*), 13
pytac.element (*module*), 16
pytac.exceptions (*module*), 19
pytac.lattice (*module*), 20
pytac.load_csv (*module*), 26
pytac.units (*module*), 27
pytac.utils (*module*), 31

R

rb_pv (*pytac.device.EpicsDevice attribute*), 14

S

s (*pytac.element.Element attribute*), 18
set_conversion_limits() (*pytac.units.UnitConv method*), 30
set_data_source() (*pytac.data_source.DataSourceManager method*), 11
set_data_source() (*pytac.element.Element method*), 18
set_data_source() (*pytac.lattice.Lattice method*), 25
set_default_data_source() (*pytac.lattice.Lattice method*), 25
set_default_units() (*pytac.lattice.Lattice method*), 25
set_element_values() (*pytac.lattice.EpicsLattice method*), 21
set_element_values() (*pytac.lattice.Lattice method*), 25

set_lattice() (*pytac.element.Element method*), 18
set_multiple() (*pytac.cs.ControlSystem method*), 8
set_post_eng_to_phys() (*pytac.units.UnitConv method*), 31
set_pre_phys_to_eng() (*pytac.units.UnitConv method*), 31
set_single() (*pytac.cs.ControlSystem method*), 8
set_unitconv() (*pytac.data_source.DataSourceManager method*), 11
set_unitconv() (*pytac.element.Element method*), 18
set_unitconv() (*pytac.lattice.Lattice method*), 26
set_value() (*pytac.data_source.DataSource method*), 9
set_value() (*pytac.data_source.DataSourceManager method*), 11
set_value() (*pytac.data_source.DeviceDataSource method*), 13
set_value() (*pytac.device.BasicDevice method*), 13
set_value() (*pytac.device.Device method*), 14
set_value() (*pytac.device.EpicsDevice method*), 15
set_value() (*pytac.element.Element method*), 18
set_value() (*pytac.lattice.Lattice method*), 26
sp_pv (*pytac.device.EpicsDevice attribute*), 14
symmetry (*pytac.lattice.EpicsLattice attribute*), 20
symmetry (*pytac.lattice.Lattice attribute*), 21

T

type_ (*pytac.element.Element attribute*), 16

U

unit_function() (*in module pytac.units*), 31
UnitConv (*class in pytac.units*), 29
units (*pytac.data_source.DataSource attribute*), 9
units (*pytac.data_source.DeviceDataSource attribute*), 12
UnitsException, 19

X

x (*pytac.units.PchipUnitConv attribute*), 28

Y

y (*pytac.units.PchipUnitConv attribute*), 28