pymodeler Documentation

Release 0.1.3+0.g482d3e4.dirty

Alex Drlica-Wagner, Eric Charles

May 30, 2018

Contents:

CHAPTER 1

Installation

1.1 Installing with pip

These instructions cover installation with the `pip` package management tool. This will install pymodeler and its dependencies into your python distribution.

Before starting the installation process, you will need to determine whether you have setuptools and pip installed in your local python environment. The following command will install both packages in your local environment:

```
$ curl https://bootstrap.pypa.io/get-pip.py | python -
```

Check if pip is correctly installed:

```
$ which pip
```

Once again, if this isn't the pip in your python environment something went wrong. Now install pymodeler by running:

```
$ pip install pymodeler
```

Finally, check that pymodeler imports:

```
$ python
Python 2.7.8 (default, Aug 20 2015, 11:36:15)
[GCC 4.2.1 Compatible Apple LLVM 6.0 (clang-600.0.56)] on darwin
Type "help", "copyright", "credits" or "license" for more information.
>>> import pymodeler
>>> pymodeler.__file__
```

The instructions describe how to install development versions of Pymodeler. Before installing a development version we recommend first installing a tagged release following the *Installing with pip* instructions above.

The development version of Pymodeler can be installed by running `pip install` with the URL of the git repository:

```
$ pip install git+https://github.com/kadrlica/pymodeler.git
```

This will install the most recent commit on the master branch. Note that care should be taken when using development versions as features/APIs under active development may change in subsequent versions without notice.

1.2 Upgrading

By default installing pymodeler with `pip` will get the latest tagged released available on the PyPi package respository. You can check your currently installed version of pymodeler with `pip show`:

```
$ pip show pymodeler
```

To upgrade your pymodeler installation to the latest version run the installation command with `--upgrade --no-deps` (remember to also include the `--user` option if you're running at SLAC):

```
$ pip install pymodeler --upgrade --no-deps
Collecting pymodeler
Installing collected packages: pymodeler
  Found existing installation: pymodeler 0.1.0
    Uninstalling pymodeler-0.1.0:
      Successfully uninstalled pymodeler-0.1.0
Successfully installed pymodeler-0.1.1
```

1.3 Developer Installation

These instructions describe how to install pymodeler from its git source code repository using the `setup.py` script. Installing from source can be useful if you want to make your own modifications to the pymodeler source code. Note that non-developers are recommended to install a tagged release of pymodeler following the *Installing with pip* or instructions above.

First clone the pymodeler git repository and cd to the root directory of the repository:

```
$ git clone https://github.com/kadrlica/pymodeler.git
$ cd pymodeler
```

To install the latest commit in the master branch run `setup.py install` from the root directory:

```
# Install the latest commit
$ git checkout master
$ python setup.py install --user
```

A useful option if you are doing active code development is to install your working copy of the package. This will create an installation in your python distribution that is linked to the copy of the code in your local repository. This allows you to run with any local modifications without having to reinstall the package each time you make a change. To install your working copy of pymodeler run with the `develop` argument:

```
# Install a link to your source code installation
$ python setup.py develop --user
```

You can later remove the link to your working copy by running the same command with the `--uninstall` flag:

```
# Install a link to your source code installation
$ python setup.py develop --user --uninstall
```

Specific release tags can be installed by running `git checkout` before running the installation command:

```
# Checkout a specific release tag
$ git checkout X.X.X
$ python setup.py install --user
```

To see the list of available release tags run `git tag`.

1.4 Issues

If you are running OSX El Capitan or newer you may see errors like the following:

```
dyld: Library not loaded
```

In this case you will need to disable the System Integrity Protections (SIP). See here for instructions on disabling SIP on your machine.

CHAPTER 2

pymodeler package

2.1 Module contents

Infrastructure for creating parametrized models in python.

2.2 class Model

class pymodeler.**Model**(***kwargs*)

Bases: object

A base class to manage Parameters and Properties

Users should define Model sub-classes and override the _params and _mapping static data members to define the parameters and mappings they want.

Examples:

```
class ModelExample:
# Define the parameters for this class
_params = odict([('fuel_rate',Property(default=10.,dtype=float,units="km/l")),
                 ('fuel_type',Property(default="diesel",dtype=str)),
                 ('distance',Parameter(default=10.,units="km")),
                 ('fuel_needed',Derived(units="l"))])

    # Define mappings for this class
    _mapping = odict([("dist","distance"),
                      ("rate","fuel_rate")])

    # Define the loader function for the fuel_needed Derived property
    def _fuel_needed(self):
        return self.distance / self.fuel_rate

Construction:
```

(continues on next page)

(continued from previous page)

```
Default, all Properties take their default values:
m = ModelExample()

Setting Properties:
m = ModelExample(fuel_rate=7, distance=12.)

Setting Properties using the Mapping:
m = ModelExample(rate=7, dist=12.)

Setting Paramter errors / bounds:
m = ModelExample(distance = dict(value=12,errors=[1.,1.],bounds=[7.,15.]))

Access to properties:
Get the value of a Property, Parameter or Derived Parameter:
m.fuel_rate
m.distance
m.fuel_neded
m.dist                      # Uses the mapping

Get access to a Property, e.g.,to know something about it besides the value,
note that this can also be used to modify the attributes of the properties:
m.getp('fuel_rate').dtype
m.getp('distance').errors

Get acess to only the Parameter type properties
m.get_params()             # Get all of the Parameters
m.get_params(paramNames)   # Get a subset of the Parameters, by name

Setting Properties or Paramaters:

Set the value of a Property or Parameter:
m.fuel_rate = 8.
m.fuel_rate = "xx"          # This will throw a TypeError
m.fuel_type = "gasoline"
m.distance = 10.
m.dist = 10.                # Uses the mapping

Set the attributes of a Property:
m.setp('fuel_rate',value=7.)    # equivalent to m.fuel_rate = 7.
m.setp('fuel_rate',value="xx")  # This will throw a TypeError
m.setp('distance',value=12,errors=[1.,1.],bounds=[7.,15.])

Set all the Properties using a dictionary or mapping
m.set_attributes(``**kwargs``)

Clear all of the Derived properties (to force recomputation)
m.clear_derived()

Output:

Convert to an ~collections.OrderedDict
m.todict()

Convert to a yaml string:
m.dump()
```

(continues on next page)

(continued from previous page)

```
Access the values of all the Parameter objects:
m.param_values()            # Get all the parameter values
m.param_values(paramNames)  # Get a subset of the parameter values, by name

Access the errors of all the Parameter objects:
m.param_errors()            # Get all the parameter values
m.param_errors(paramNames)  # Get a subset of the parameter values, by name
```

clear_derived()
Reset the value of all Derived properties to None

This is called by setp (and by extension __setattr__)

defaults
Ordered dictionary of default parameters.

dump()
Dump this object as a yaml string

get_params(*pnames=None*)
Return a list of Parameter objects

Parameters **pname** (*list or None*) – If a list get the Parameter objects with those names

If none, get all the Parameter objects

Returns **params** – list of Parameters

Return type list

getp(*name*)
Get the named *Property*.

Parameters **name** (*str*) – The property name.

Returns **param** – The parameter object.

Return type *Property*

mappings
Ordered dictionary of mapping of names.

This can be used to assign multiple names to a single parameter

param_errors(*pnames=None*)
Return an array with the parameter errors

Parameters **pname** (*list of string or none*) – If a list of strings, get the Parameter objects with those names

If none, get all the Parameter objects

Returns

- *~numpy.array of parameter errors*
- *Note that this is a N x 2 array.*

param_values(*pnames=None*)
Return an array with the parameter values

Parameters **pname** (*list or None*) – If a list, get the values of the *Parameter* objects with those names

If none, get all values of all the *Parameter* objects

Returns **values** – Parameter values

Return type *np.array*

set_attributes(***kwargs*)
Set a group of attributes (parameters and members). Calls *setp* directly, so kwargs can include more than just the parameter value (e.g., bounds, free, etc.).

setp(*name*, *clear_derived=True*, *value=None*, *bounds=None*, *free=None*, *errors=None*)
Set the value (and bounds) of the named parameter.

Parameters

- **name** (*str*) – The parameter name.
- **clear_derived** (*bool*) – Flag to clear derived objects in this model
- **value** – The value of the parameter, if None, it is not changed
- **bounds** (*tuple or None*) – The bounds on the parameter, if None, they are not set
- **free** (*bool or None*) – Flag to say if parameter is fixed or free in fitting, if None, it is not changed
- **errors** (*tuple or None*) – Uncertainties on the parameter, if None, they are not changed

todict()
Return self cast as an '~collections.OrderedDict' object

2.3 class Property

class pymodeler.**Property**(***kwargs*)
Bases: object

Base class for model properties.

This class and its sub-classes implement variations on the concept of a 'mutable' value or 'l-value', i.e., an object that can be assigned a value.

This class defines some interfaces that help read/write heirachical sets of properties between various formats (python dictionaries, yaml files, astropy tables, etc..)

The pymodeler.model.Model class maps from property names to Property instances.

Parameters

- **value** – Property value [None]
- **help** – Help description ['']
- **format** – Format string for printing ['%s']
- **dtype** – Data type [None]
- **default** – Default value [None]
- **required** – Is this propery required? [False]
- **unit** – Units associated to value [None]

check_bounds(*value*)
Hook for bounds-checking, invoked during assignment.

Sub-classes can raise an exception for out-of-bounds input values.

check_type(*value*)
Hook for type-checking, invoked during assignment.

raises TypeError if neither value nor self.dtype are None and they do not match.

will not raise an exception if either value or self.dtype is None

clear_value()
Set the value to None

This can be useful for sub-classes that use None to indicate an un-initialized value.

Note that this invokes hooks for type-checking and bounds-checking that may be implemented by sub-classes, so it should will need to be re-implemented if those checks do note accept None as a valid value.

defaults = [('value', None, 'Property value'), ('help', '', 'Help description'), ('for

classmethod defaults_docstring(*header=None*, *indent=None*, *footer=None*)
Add the default values to the class docstring

dump()
Dump this object as a yaml string

innertype()
Return the type of the current value

set(***kwargs*)
Set the value to kwargs['value']

The invokes hooks for type-checking and bounds-checking that may be implemented by sub-classes.

set_value(*value*)
Set the value

This invokes hooks for type-checking and bounds-checking that may be implemented by sub-classes.

todict()
Convert to a '~collections.OrderedDict' object.

By default this only assigns {'value':self.value}

value
Return the current value

This may be modified by sub-classes to do additional operations (such as caching the results of complicated operations needed to compute the value)

2.4 class Parameter

class pymodeler.**Parameter**(***kwargs*)
Bases: pymodeler.parameter.Property

Property sub-class for defining a numerical Parameter.

This includes value, bounds, error estimates and fixed/free status (i.e., for fitting)

Adapted from MutableNum: https://gist.github.com/jheiv/6656349

Parameters

- **value** – Property value [None]
- **help** – Help description [‘’]
- **format** – Format string for printing [‘%s’]
- **dtype** – Data type [<Number>]
- **default** – Default value [None]
- **required** – Is this propery required? [False]
- **unit** – Units associated to value [None]
- **bounds** – Allowed bounds for value [None]
- **errors** – Errors on this parameter [None]
- **free** – Is this propery allowed to vary? [False]

bounds
Return the parameter bounds.

None implies unbounded.

check_bounds(*value*)
Hook for bounds-checking, invoked during assignment.

raises ValueError if value is outside of bounds. does nothing if bounds is set to None.

check_type(*value*)
Hook for type-checking, invoked during assignment. Allows size 1 numpy arrays and lists, but raises TypeError if value can not be cast to a scalar.

d = ('free', False, 'Is this propery allowed to vary?')

defaults = [('value', None, 'Property value'), ('help', '', 'Help description'), ('for

dump()
Dump this object as a yaml string

errors
Return the parameter uncertainties.

None implies no error estimate. Single value implies symmetric errors. Two values implies low,high asymmetric errors.

free
Return the fixd/free status

idx = 3

item()
For asscalar

static representer(*dumper*, *data*)
http://stackoverflow.com/a/14001707/4075339 http://stackoverflow.com/a/21912744/4075339

set(***kwargs*)
Set the value,bounds,free,errors based on corresponding kwargs

The invokes hooks for type-checking and bounds-checking that may be implemented by sub-classes.

set_bounds(*bounds*)
Set bounds

set_errors(*errors*)
Set parameter error estimate

set_free(*free*)
Set free/fixed status

symmetric_error
Return the symmertic error

Similar to above, but zero implies no error estimate, and otherwise this will either be the symmetric error, or the average of the low,high asymmetric errors.

todict()
Convert to a '~collections.OrderedDict' object.

This assigns {'value':self.value,'bounds'=self.bounds, 'free'=self.free,'errors'=self.errors}

2.5 class Derived

class pymodeler.**Derived**(***kwargs*)
Bases: pymodeler.parameter.Property

Property sub-class for derived model properties (i.e., properties that depend on other properties)

This allows specifying the expected data type and formatting string for printing, and specifying a 'loader' function by name that is used to compute the value of the property.

Parameters

- **value** – Property value [None]
- **help** – Help description ['']
- **format** – Format string for printing ['%s']
- **dtype** – Data type [None]
- **default** – Default value [None]
- **required** – Is this propery required? [False]
- **unit** – Units associated to value [None]
- **loader** – Function to load datum [None]

defaults = [('value', None, 'Property value'), ('help', '', 'Help description'), ('for

value
Return the current value.

This first checks if the value is cached (i.e., if *self.__value__* is not None)

If it is not cached then it invokes the *loader* function to compute the value, and caches the computed value

CHAPTER 3

Changelog

This page is a changelog for releases of pymodeler. You can also browse releases on Github.

3.1 0.1.1 (3/22/2018)

Added __bool__ to Parameter

3.2 0.1.1 (3/17/2018)

Addded sphinx documentation and did a lot of pylint related cleanup. None of the interfaces have w.r.t. previous releases.

3.3 0.1.0 (6/29/2016)

Implemented the Property class and restructuring of Parameter to inherit from it. Some general restructuring and updated documentation throughout.

Note, the model definition interface has changed from previous versions (and is now better documented).

3.4 0.0.5 (3/24/2016)

First release pushed to PyPI

CHAPTER 4

Indices and tables

Python Module Index

p

Index