PHOEBE quick-reference

Markus Rasmussen Mosbech

June 27, 2017

Abstract

This document is intended as a supplement to my full guide and the official PHOEBE tutorials, it is intended to provide a quick place to look up commands rather than give detailed explanations to teach how to use them. For this, see my other guide or the excellent online tutorials as http://phoebe-project.org.

Contents

Chapter 1

Preliminaries

Loading PHOEBE and the PHEOBE-units:

```
import phoebe
from phoebe import u
```

Initializing a logger:

```
logger = phoebe.logger(clevel=LEVEL,
        flevel=LEVEL, filename='FILENAME')
```

clevel and above are printed in the terminal, flevel and above are saved in the logfile. Levels are, from low to high importance:

- 'DEBUG'
- 'INFO'
- 'WARNING'
- 'ERROR'
- 'CRITICAL'

Chapter 2

Parameters

The parameter types

```
param = phoebe.parameters.StringParameter(ARGS)
param = phoebe.parameters.ChoiceParameter(ARGS)
param = phoebe.parameters.FloatParameter(ARGS)
param = phoebe.parameters.IntParameter(ARGS)
param = phoebe.parameters.BoolParameter(ARGS)
param = phoebe.parameters.FloatArrayParameter(ARGS)
```

Possible ARGS vary between different parameter types, I provide here a list of possible ARGS:

- qualifier= the name of the parameter
- description = a description of the parameter
- default_unit = the default unit of the parameter (only for Float and FloarArray)
- limits = the limits on the value
- value = the value of the parameter
- choices = possible values for a ChoiceParameter
- kind = the kind of parameter (see guide)
- context = the context of the parameter (see guide)

Access tags and attributes of a parameter as object attributes or dictionary entries.

```
param.TAG
param['TAG']
```

Replace TAG with any attribute or tag of the parameter.
To get all tags

```
param.meta
```

To get all attributes

```
param.attributes
```

To change the value of a parameter (if float or floatarray, unit can be multiplied with value)

```
param.set_value(VALUE)
```

To get a value (in default units, for different units add `unit = desired unit` as argument)

```
param.get_value()
```

to get as astropy quantity (for specifik unit, use `unit = desired unit` as argument)

```
param.get_quantity()
```

Default unit can be changed with

```
param.set_default_unit(UNIT)
```

Chapter 3

ParameterSets

Initializing a ParameterSet from parameters

```
ps = phoebe.parameters.ParameterSet(
[param1, param2, param3])
```

To get a parameter from a ParameterSet, use twigs (see guide)

```
ps.get(TWIG)
```

This returns a parameter, so anything possible with a parameter is possible here.

To get or set value of a specific parameter in a set, use the twig

```
ps.set_value(TWIG, VALUE)
ps.get_value(TWIG)
```

To get all tags

```
ps.meta.keys()
```

To get the value of a tag for the ParameterSet, use

```
ps.TAG
```

To get all values of a tag for parameters part of the set, use plural

```
ps.TAGS
```

To filter a ParameterSet, returning a new ParameterSet containing only parameters with `TAG=TAGVALUE`. Multiple tags can be used, seperated by commas. Note: ALWAYS returns ParameterSet, even if it only contains one parameter.

```
ps.filter(TAG=TAGVALUE)
```

To filter using twigs

```
ps['TAGVALUE']
```

To use more than one tag

```
ps['TAGVALUE1']['TAGVALUE2']
ps['TAGVALUE1@TAGVALUE2']
```

When using twigs, if only one parameter fits the filter, it is returned rather than a set.

Chapter 4

Bundles

To initialize an empty Bundle

```
b = phoebe.bundle()
```

To initialize a Bundle from parameters

```
b = phoebe.bundle([param1, param2])
```

To set the value of a parameter in a Bundle, use either of

```
b[TWIG].set_value(VALUE)
b.set_value(TWIG, value)
```

To save a bundle to a file or load from a file, use

```
b.save('filename,phoebe')
b = phoebe.Bundle.open('filename.open')
```

To perform a linear interpolation, use

```
yval = b['y'].interp_value(x=xval)
```

this yields the value of y at x=xval.

Chapter 5

Systems

To create a system in the Bundle, add components. This is done with

```
b.add_component(Componenttype,
                component=Qualifier, Args)
```

The Componenttype can be input using two different syntaxes, either `phoebe.component.componer`
or the componenttype as a string. The qualifier is just the name. Stars and orbits
can also be added with their own commands

```
b.add_star(Qualifier, Args)
b.add_orbit(Qualifier, Args)
```

The Args can be used to set the values of parameters added with the component.

To get the hierarchy of a bundle

```
b.get_hierarchy()
```

To set the hierarchy use either of

```
b.set_hierarchy(phoebe.hierarchy.binaryorbit,
        b['binary'], b['primary'], b['secondary'])
b.set_hierarchy(phoebe.hierarchy.binaryorbit(
        b['binary'], b['primary'], b['secondary']))
b.set_hierarchy(
        'orbit:binary(star:primary, star:secondary)')
```

To get different parts of the hierarchy, use

```
b.hierarchy.get_stars()
b.hierarchy.get_orbits()
```

To get the parts of a hierarchy with a specific relation to a component, use

```
b.hierarchy.get_parent_of(component)
b.hierarchy.get_children_of(component)
b.hierarchy.get_child_of(component, index)
b.hierarchy.get_sibling_of(component)
b.hierarchy.get_primary_or_secondary(component)
```

To see which other parameters a parameter is constrained by, use its twig with

```
b[twig].constrained_by
```

To switch which parameter is calculated from which use the twig of the calculated parameter and the qualifier of the one to be calculated now

```
b.flip_constraint(twig, newparameter)
```

Chapter 6

Datasets and computation

To add a dataset to a bundle, use

```
b.add_dataset(datasettype, data, dataset='name', components)
```

Datasettype uses either `phoebe.datasets.type` or `'type'`. Data may have several components, such as times, fluxes etc. Added as numpy arrays. `dataset='name'` sets the dataset name, components are optional, tells which components this dataset is for. Dataset types are orb, rv, lc and mesh.

To remove a dataset, use

```
b.remove_dataset('name')
```

If you want to remove more than one dataset, a filter can also be used instead of just a name.

A new set of computation options can be added with

```
b.add_compute(backend, compute='name', Args)
```

Backend uses either `phoebe.compute.backend` or `'backend'`. Included are phoebe and legacy. `compute='name'` sets the name and Args are used to set computation options.

To run computations, use

```
b.run_compute(compute='computename', model='modelname')
```

`'computename'` is the set of compute options used, `'modelname'` is the name the resulting model is saved with. Multiple compute option sets can be used, if included as a list.

To plot synthetic models using PHOEBE's built-in facilities, use

```
axs, artists = b.plot()
```

To plot only a specific result, use its twig

```
axs, artists = b[twig].plot()
```

to highlight a specific time, include a `time=` argument in plot, using the chosen time. Arguments can also be added to change the look of the highlight or plot in a specific way.

To get the ephemeris, use

```
b.get_ephemeris()
```

To calculate phase from time or vice versa, use

```
b.to_phase(time)
b.to_time(phase)
```