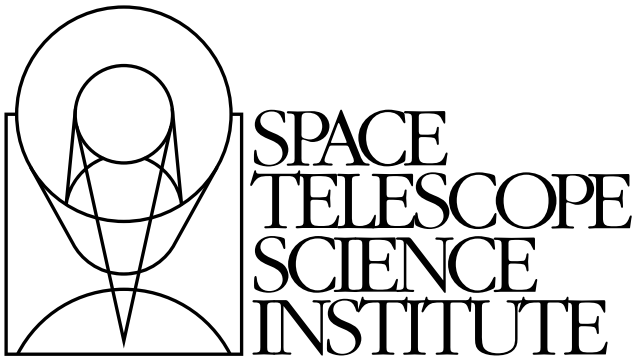

PySynphot Documentation

Release 0.9.4dev

STScI

May 03, 2013

CONTENTS

Package
Astrolib Pysynphot

CHAPTER

ONE

PURPOSE

Object-oriented replacement for STSDAS synphot package.

This __init__ file is used to expose the desired elements of the user interface for interactive use.

CHAPTER

TWO

DEPENDENCIES

- numpy 1.5.1 or greater
- pyfits 2.4 or greater

CHAPTER

THREE

ENVIRONMENT

The environment variable PYSYN_CDBS must be set.

CHAPTER

FOUR

EXAMPLE

In the examples below, items which may be installation- or platform- specific are commented out so as to be excluded from doctest. However users are still encouraged to try these examples.

A quickstart tutorial containing further examples and other documentation can be found at U{http://stsdas.stsci.edu/pysynphot/

```
>>> import pysynphot as S
>>> import os
>>> print S.__version__
0.9.4dev
>>> #Read a spectrum from a file
>>> vega=S.FileSpectrum(S.locations.VegaFile)

>>> bb=S.BlackBody(40000)
>>> print bb
BB(T=40000)

>>> pl=S.PowerLaw(10000,-2)
>>> print pl
Power law: refwave 10000 angstrom, index -2

>>> g1=S.GaussianSource(18.3,18000,2000,fluxunits='abmag')
>>> print g1
Gaussian: mu=18000 angstrom,fwhm=2000 angstrom, total flux=18.3 abmag

>>> unitflux=S.FlatSpectrum(18,fluxunits='abmag')
>>> print unitflux
Flat spectrum of 18 abmag

>>> bp1=S.ObsBandpass('acs,hrc,f555w')
>>> print bp1
acs,hrc,f555w
>>> print bp1.wave
[   500.   1000.   1010. ...,  11999.  30000.  30010.]
>>> print bp1.throughput
[ 0.  0.  0. ...,  0.  0.  0.]

>>> len(bp1)
6

>>> print bp1.waveunits
angstrom
```

```
>>> obs1=S.Observation(vega,bp1)

>>> print obs1.waveunits
angstrom
>>> print obs1.fluxunits
flam
```

CHAPTER

FIVE

MODULES

5.1 pysynphot.Cache

pysynphot.Cache.**reset_catalog_cache**()
Empty the CATALOG_CACHE global variable.

5.2 pysynphot.catalog

class pysynphot.catalog.**Icat**(*catdir*, *Teff*, *metallicity*, *log_g*)
This class constructs a model from the grid available in catalogs such as the Castelli & Kurucz. See the Synphot User's Data Manual, Appendix A, for more information at http://www.stsci.edu/hst/HST_overview/documents/synphot/AppA_Catalogs4.html#48115

spec = Icat(CDBS directory name,Teff,metallicity,logG).

Parameters

catdir : str

name of directory holding the catalogs

Teff : float

effective temperature of model

metallicity : float

metallicity of model

log_g : float

log of gravity term for model

5.3 pysynphot.exceptions

Custom exceptions for pysynphot to raise

exception pysynphot.exceptions.**AmbiguousObsmode**(*msg*)

exception pysynphot.exceptions.**BadRow**(*msg*, *rows=None*)

exception pysynphot.exceptions.**DisjointError**(*msg*)

exception pysynphot.exceptions.**DuplicateWavelength**(*msg*, *rows=None*)

exception pysynphot.exceptions.**ExtrapolationNotAllowed**(*msg*)

exception pysynphot.exceptions.**GraphtabError**(*msg*)

exception pysynphot.exceptions.**IncompatibleSources**(*msg*)

exception pysynphot.exceptions.**IncompleteObsmode**(*msg*)

exception pysynphot.exceptions.**OverlapError**(*msg*)

exception pysynphot.exceptions.**ParameterOutOfBounds**(*msg*)

exception pysynphot.exceptions.**PartialOverlap**(*msg*)

exception pysynphot.exceptions.**PysynphotError**(*msg*)
parent class

exception pysynphot.exceptions.**TableFormatError**(*msg*, *rows=None*)

exception pysynphot.exceptions.**UndefinedBinset**(*msg*)

exception pysynphot.exceptions.**UnsortedWavelength**(*msg*, *rows=None*)

exception pysynphot.exceptions.**UnusedKeyword**(*msg*)

exception pysynphot.exceptions.**ZeroWavelength**(*msg*, *rows=None*)

5.4 pysynphot.extinction

class pysynphot.extinction.**DeprecatedExtinction**(*extinction in magnitudes, 'gal1|smc|lmc reddening laws*)

Extinction mimics as a spectral element.

class pysynphot.extinction.**Gal1**(*extval*)

citation = 'Seaton 1979 (MNRAS 187:75)'

name = 'gal1'

class pysynphot.extinction.**Gal2**(*extval*)

citation = 'Savage & Mathis 1979 (ARA&A 17:73)'

name = 'gal2'

class pysynphot.extinction.**Gal3**(*extval*)

citation = 'Cardelli, Clayton & Mathis 1989 (ApJ 345:245)'

name = 'gal3'

class pysynphot.extinction.**Lmc**(*extval*)

citation = 'Howarth 1983 (MNRAS 203:301)'

name = 'LMC'

class pysynphot.extinction.**Smc**(*extval*)

citation = 'Prevot et al.1984 (A&A 132:389)'

name = 'SMC'

class pysynphot.extinction.**Xgal**(*extval*)

citation = 'Calzetti, Kinney and Storchi-Bergmann, 1994 (ApJ 429:582)'

name = 'XGAL'

pysynphot.extinction.**factory**(*redlaw*, **args*, ***kwargs*)

5.4.1 Global variables

pysynphot.extinction.**_seatonx = array([0. , 1. , 1.1, 1.2, 1.3, 1.4, 1.5, 1.6, 1.7, 1.8, 1.9, 2. , 2.1, 2.2, 2.3, 2.4, 2.5, 2.6, 2.7]**

pysynphot.extinction.**_seatone = array([0. , 1.36, 1.64, 1.84, 2.04, 2.24, 2.44, 2.66, 2.88, 3.14, 3.36, 3.56, 3.77, 3.96, 4.15**

pysynphot.extinction.**_lmcx = array([0. , 0.29, 0.45, 0.8 , 1.11, 1.43, 1.83])**

pysynphot.extinction.**_lmce = array([0. , 0.16, 0.38, 0.87, 1.5 , 2.32, 3.1])**

pysynphot.extinction.**_smcx = array([0. , 0.29, 0.45, 0.8 , 1.11, 1.43, 1.82, 2.35, 2.7 , 3.22, 3.34, 3.46, 3.6 , 3.75, 3.92, 4.09,**

pysynphot.extinction.**_smce = array([0. , 0.16, 0.38, 0.87, 1.5 , 2.32, 3.1 , 4.1 , 4.77, 5.39, 5.75, 6.1 , 6.25, 6.59, 7.01, 7.34,**

pysynphot.extinction.**_waveset = array([20. , 19.92113104, 19.8425731 , ..., 0.38766684, 0.3861381 , 0.38476739])**

pysynphot.extinction.**_seaton = array([71.17 , 70.48569062, 69.807759 , ..., 0.20438839, 0.20277958, 0.20134248])**

pysynphot.extinction.**_lmc = array([54.10590134, 53.73888694, 53.3746187 , ..., 0.29429191, 0.29218989, 0.29030516])**

pysynphot.extinction.**_smc = array([23.74 , 23.74 , 23.74 , ..., 0.29429191, 0.29218989, 0.29030516])**

pysynphot.extinction.**_xgal = array([94.7214 , 93.42724771, 92.15200482, ..., 1.35077283, 1.34571793, 1.34118379])**

5.5 pysynphot.graphtab

Graph table re-implementation Data structure & traversal algorithm suggested by Alex Martelli, http://stackoverflow.com/questions/844505/is-a-graph-library-eg-networkx-the-right-solution-for-my-python-problem

class pysynphot.graphtab.**CompTable**(*fname*)
This class will cooperate with a GraphPath to produce a realized list of files

inittab()

class pysynphot.graphtab.**GraphNode**
Structure to hold all the information associated with a single innode of the graph table. The constructor produces an empty node, which must be filled later. This structure will be the value associated with the GraphTab dict.

((default_outnode, compname, thcompname), {'kwd':(outnode, compname, thcompname)})

get_default()

get_named(*kwd*)

set_default(*outnode*, *compname*, *thcompname*)

set_named(*kwd*, *outnode*, *compname*, *thcompname*)

class pysynphot.graphtab.**GraphPath**(*obsmode_string*, *optical*, *thermal*, *params*, *tname*)
Simple class containing the result of a traversal of the GraphTable

Parameters
optical : list of strings

optical component names

thermal : list of strings

thermal component names

params : dict

dictionary of {compname:parameterized value} for
any parameterized keywords used in the obsmode string

class pysynphot.graphtab.**GraphTable**(*fname*)

add_descendants(*node*, *updateset=None*)
auxiliary function: add all descendants of node to someset

inittab()

traverse(*icss*, *verbose=False*)

validate()
Simulataneously checks for loops and unreachable nodes

pysynphot.graphtab.**extract_keywords**(*icss*)
Helper function

Parameters
icss : string

comma-separated string

Returns
kws : list of string

set of keywords

paramdict : dict

dict of {parameterized_keyword: parameter_value}

5.6 pysynphot.locations

pysynphot.locations.**get_data_filename**(*filename*)

pysynphot.locations.**irafconvert**(*iraffilename*)
Convert the IRAF file name (in directory$file format) to its unix equivalent

Parameters
Input: string iraffilename :

Returns
Output: string unixfilename :

If '$' not found in the input string, just return the input string Non-string input raises an AttributeError

5.6.1 Global Variables

pysynphot.locations.**rootdir** = ''
str(object) -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pysynphot.locations.**specdir** = '/Users/sienkiew/plugh/lib/python2.7/site-packages/pysynphot-0.9.5-py2.7-macosx-10.6-
str(object) -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pysynphot.locations.**CAT_TEMPLATE** = 'grid/*/catalog.fits'
str(object) -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pysynphot.locations.**KUR_TEMPLATE** = 'grid/*'
str(object) -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pysynphot.locations.**VegaFile** = '/Users/sienkiew/plugh/lib/python2.7/site-packages/pysynphot-0.9.5-py2.7-macosx-10.
str(object) -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pysynphot.locations.**EXTDIR** = 'grid/extinction'
str(object) -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pysynphot.locations.**RedLaws** = {}

pysynphot.locations.**wavecat** = '/Users/sienkiew/plugh/lib/python2.7/site-packages/pysynphot-0.9.5-py2.7-macosx-10.6-
str(object) -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pysynphot.locations.**CONVERTDICT** = {'crfields': 'fields', 'crcalobs': 'calobs', 'crnirspeccomp': 'comp/nirspec', 'crfocc

5.7 pysynphot.obsbandpass

class pysynphot.obsbandpass.**ObsModeBandpass**(*ob*)
Bandpass instantiated from an obsmode string

Instantiate a COmpositeSpectralElement by means of an ObservationMode (which the caller must have already created from an obstring

pixel_range(*waverange*, *waveunits=None*, *round='round'*)
Returns the number of wavelength bins within waverange.

Note: This calls the pixel_range function with self.binset as the first argument. See pixel_range for full documentation.

Parameters
waveunits : str, optional

The units of the wavelengths given in waverange. Defaults to None. If None, the wavelengths are assumed to be in the units of the waveunits attribute.

Raises
pysynphot.exceptions.UndefinedBinset :

If the binset attribute is None.

See also:

pixel_range, pysynphot.exceptions.UndefinedBinset

showfiles()
Defer to ObservationMode component

thermback()
Expose the thermal background calculation presently hidden in the obsmode class. Only bandpasses for which thermal information has been supplied in the graph table supports this method; all others will raise a NotImplementedError.

wave_range(*cenwave*, *npix*, *waveunits=None*, *round='round'*)
Get the wavelength range covered by a number of pixels, npix, centered on wavelength cenwave.

Note: This calls the wave_range function with self.binset as the first argument. See wave_range for full documentation.

Parameters
waveunits : str, optional

Wavelength units of cenwave and the returned wavelength range. Defaults to None. If None, the wavelengths are assumed to be in the units of the waveunits attribute.

Raises
exceptions.UndefinedBinset :

If the binset attribute is None.

See also:

wave_range, pysynphot.exceptions.UndefinedBinset

pysynphot.obsbandpass.**ObsBandpass**(*obstring*, *graphtable=None*, *comptable=None*, *component_dict={}*)
Generate an ObsModeBandPass or TabularSpectralElement instance

obsband = ObsBandpass(string specifying obsmode; for details see the Synphot Data User's Guide at http://www.stsci.edu/hst/HST_overview/documents/synphot/hst_synphotTOC.html

pysynphot.obsbandpass.**pixel_range**(*bins*, *waverange*, *round='round'*)
Returns the number of wavelength bins within waverange.

Parameters
bins : ndarray

Wavelengths of pixel centers. Must be in the same units as waverange.

waverange : array_like

A sequence containing the wavelength range of interest. Only the first and last elements are used. Assumed to be in increasing order. Must be in the same units as bins.

round : {'round','min','max',None}, optional

How to deal with pixels at the edges of the wavelength range. All of the options, except None, will return an integer number of pixels. Defaults to 'round'.

When set to 'round' wavelength ends that fall in the middle of a pixel are counted if more than half of the pixel is within waverange. Ends that fall in the center of a pixel are rounded up to the nearest pixel edge.

When set to 'min' only pixels wholly within waverange are counted.

When set to 'max' end pixels that are within `waverange` by any margin are counted.

When set to None the exact number of encompassed pixels, including fractional pixels, is returned.

Returns

num : int or float

Number of wavelength bins within `waverange`.

Raises

ValueError :

If `round` is not an allowed value.

pysynphot.exceptions.OverlapError :

If `waverange` exceeds the bounds of `bins`.

`pysynphot.obsbandpass.`**`wave_range`**(*bins*, *cenwave*, *npix*, *round='round'*)

Get the wavelength range covered by a number of pixels, `npix`, centered on wavelength `cenwave`.

Parameters

bins : ndarray

Wavelengths of pixel centers. Must be in the same units as `cenwave`.

cenwave : float

Central wavelength of range. Must be in the same units as `bins`.

npix : int

Number of pixels in range, centered on `cenwave`.

round : {'round','min','max',None}, optional

How to deal with pixels at the edges of the wavelength range. All of the options, except None, will return wavelength ends that correpsonds to pixel edges. Defaults to 'round'.

When set to None an exact wavelength range is returned. The wavelength ends returned may not correspond to pixel edges, but will cover exactly `npix` pixels.

When set to 'round' a wavelength range is returned such that the ends are pixel edges and the range spans exactly `npix` pixels. Ends that fall in the center of bins are rounded up to the nearest pixel edge.

When set to 'min' the returned wavelength range is shrunk so that it includes an integer number of pixels and the ends fall on pixel edges. May not span exactly `npix` pixels.

When set to 'max' the returned wavelength range is expanded so that it includes an integer number of pixels and the ends fall on pixel edges. May not span exactly `npix` pixels.

Returns

waverange : tuple of floats

The range of wavelengths spanned by `npix` centered on `cenwave`.

Raises

ValueError :

If `round` is not an allowed value.

pysynphot.exceptions.OverlapError :

If `cenwave` is not within the `binset` attribute, or the returned `waverange` would exceed the limits of the `binset` attribute.

5.8 pysynphot.observation

class `pysynphot.observation.`**`Observation`**(*Spectrum object*, *Bandpass object*, *binset=numpy array to be used for binning when converting to counts.*)

Most ObsBandpass objects have a built-in binset that is optimized for use with the specified observing mode; specifying the binset in the Observation constructor would overrirde that binset.

An Observation is the end point of a chain of spectral manipulation.

The normal means of producing an Observation is by means of the .observe() method on the spectral element.

`as_spectrum`(*binned=True*)

Reduce the Observation to a TabularSourceSpectrum.

An Observation is a complex object with some restrictions on its capabilities. At times it would be useful to work with the simulated Observation as a simpler object that is easier to manipulate and takes up less memory. This method returns a TabularSourceSpectrum made from either the (wave, flux) or the (binwave, binflux) properties of the Observation.

Parameters

binned: bool :

If True, use (binwave, binflux); otherwise use (wave, flux).

Returns

result: TabularSourceSpectrum :

`countrate`(*binned=True*, *range=None*, *force=False*)

This is the calculation performed when the ETC invokes countrate. Essentially it wants the effstim in counts.

Parameters

binned : bool [Default: True]

if True, operations will be performed on (binwave,binflux);
otherwise on (wave,flux)

range : {'low','high', None}

if range is not None, it is expected to be a sequence with two floating-point elements specifying the low and high wavelength range (specified in self.waveunits) over which the integration will be performed.

This is an _inclusive_ range.

Disjoint or partially-overlapping ranges will raise an exception by default. If force=True is set, then a partial overlap will return the calculated value rather than raise an exception.

If the specified range does not exactly match a value in the waveset:

```
- if binned=True, the bin containing the range value will
  be used. (Recall values of binwave specify bin centers.)
- if binned=False, the wave and flux arrays will be
  interpolated to the specified values.
```

force : bool [Default: False]

efflam(*binned=True*)
Calculation performed based on observation.py _EfflamCalculator, which produces EFFLPHOT results!.

effstim(*fluxunits='photlam'*)
Compute effective stimulation in specified units

initbinflux()
This routine performs the integration of the spectrum on the specified binned waveset. It uses the natural waveset of the spectrum in performing this integration.

Note: This method is implemented under the assumption that the wavelength values in the binned waveset are the *centers* of the bins.

By contrast, the native wave/flux arrays should be considered samples of a continuous function.

Thus, it makes sense to interpolate .wave/.flux; it does not make sense to interpolate .binwave/.binflux.

initbinset(*binset=None*)

pivot(*binned=True*)
This is the calculation performed when the ETC invokes calcphot. Does this need to be calculated on binned waveset, or may it be calculated on native waveset?

pixel_range(*waverange*, *waveunits=None*, *round='round'*)
Returns the number of wavelength bins within `waverange`.

Note: This calls the `pixel_range()` function with `self.binwave` as the first argument. See `pixel_range()` for full documentation.

Parameters
waveunits : str, optional
The units of the wavelengths given in `waverange`. Defaults to None. If None, the wavelengths are assumed to be in the units of the `waveunits` attribute.
Raises
pysynphot.exceptions.UndefinedBinset :
If the `binwave` attribute is None.

See also:

`pysynphot.obsbandpass.pixel_range`

redshift(*z*)

sample(*swave*, *binned=True*, *fluxunits='counts'*)
Samples the observation at the wavelength(s) swave, specified in waveunits. The binned keyword determines whether the sampling is performed on binwave/binflux, in which case no interpolation is performed, or on the native wave/flux, in which case interpolation is performed.

validate_overlap(*force*)
By default, it is required that the spectrum and bandpass fully overlap. Partial overlap will raise an error in the absence of the force keyword, which may be set to "taper" or "extrap".

wave_range(*cenwave*, *npix*, *waveunits=None*, *round='round'*)
Get the wavelength range covered by a number of pixels, `npix`, centered on wavelength `cenwave`.

Note: This calls the `obsbandpass.wave_range` function with `self.binwave` as the first argument. See `obsbandpass.wave_range` for full documentation.

Parameters
waveunits : str, optional
Wavelength units of `cenwave` and the returned wavelength range. Defaults to None. If None, the wavelengths are assumed to be in the units of the `waveunits` attribute.
Raises
pysynphot.exceptions.UndefinedBinset :
If the `binwave` attribute is None.

See also:

`obsbandpass.wave_range`

writefits(*fname*, *clobber=True*, *trimzero=True*, *binned=True*, *hkeys=None*)
All we really want to do here is flip the default value of 'binned' from the vanilla spectrum case.

binflux
Flux on binned wavelength set property

pysynphot.observation.**check_overlap**(*a*, *b*)
Check for wavelength overlap between two psyn instances.

Generalized from psyn.SpectralElement.check_overlap().

Parameters
a : `Integrator` instance

b : `Integrator` instance

Typically a psyn.SourceSpectrum, psyn.SpectralElement, psyn.Observation, or psyn.ObsBandpass

Returns
result : {'full','partial','none'}

See also:

`pysynphot.spectrum.Integrator`, `pysynphot.spectrum.SpectralElement`

pysynphot.observation.**validate_overlap**(*comp1*, *comp2*, *force*)
Validate the overlap between the wavesets of the two components. If force is not None, the components may be adjusted.

5.9 pysynphot.observationmode

class pysynphot.observationmode.**BaseObservationMode**(*obsmode*, *method='HSTGraphTable'*, *graphtable=None*)
Class that handles the graph table, common to both optical and thermal obsmodes.

GetFileNames()

bandWave()
Return the binned waveset most appropriate for the obsmode, as defined by the wavecat.dat file.

showfiles()
Duplicate synphot showfiles behavior

class pysynphot.observationmode.**ObservationMode**(*obsmode*, *method='HSTGraphTable'*, *graphtable=None*, *comptable=None*, *component_dict={}*)

Sensitivity()
Calculate the sensitivity by combining the throughput curves with hc/lambda to convert erg/cm^2/sec/Angstrom to counts/sec. Multiplying this by the flux in erg/cm^2/sec/Angstrom will give counts/sec/Angstrom

ThermalSpectrum()

Throughput()
Throughput returns the TabularSpectralElement obtained by multiplying the SpectralElement components together. Unitless

5.9.1 Variables

pysynphot.observationmode.**rootdir = ''**
str(object) -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pysynphot.observationmode.**datadir = '/Users/sienkiew/plugh/lib/python2.7/site-packages/pysynphot-0.9.5-py2.7-mac**
str(object) -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pysynphot.observationmode.**wavecat = '/Users/sienkiew/plugh/lib/python2.7/site-packages/pysynphot-0.9.5-py2.7-mac**
str(object) -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pysynphot.observationmode.**CLEAR = 'clear'**
str(object) -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

5.10 pysynphot.planck

pysynphot.planck.**bb_photlam_arcsec**(*wave*, *temperature*)
Planck function in photlam / square arcsec. wavelength in Angstrom, temperature in Kelvin. Translated from Anand's spp code in synphot.

pysynphot.planck.**bbfunc**(*wave*, *temperature*)
Planck function in photlam. wavelength in Angstrom, temperature in Kelvin. Adapted from bbfunc in synphot.

pysynphot.planck.**llam_SI**(*wave*, *temperature*)
Planck function in standard units. wavelength in meters, temperature in Kelvin. Adapted from Anand's spp code in synphot.

5.10.1 Variables

pysynphot.planck.**H = 6.6262e-27**
float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

pysynphot.planck.**HS = 6.6262e-34**
float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

pysynphot.planck.**C = 299792500.0**
float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

pysynphot.planck.**K = 1.38062e-23**
float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

pysynphot.planck.**C1 = 1.1910666467986475e-16**
float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

pysynphot.planck.**C2 = 0.014388354967333514**
float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

pysynphot.planck.**F = 2.350438638182907e-25**
float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

pysynphot.planck.**LOWER = 0.0001**
float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

pysynphot.planck.**UPPER = 85.0**
float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

5.11 pysynphot.pysynphot_utils

pysynphot.pysynphot_utils.**calcbinflux**()
Calculate binned flux.

5.12 pysynphot.reddening

class pysynphot.reddening.**CustomRedLaw**(*wave=None*, *waveunits='InverseMicrons'*, *Avs-caled=None*, *name='Unknown Reddening Law'*, *litref=None*)

reddening(*extval*)
Compute the reddening for the provided value of the extinction.

class pysynphot.reddening.**RedLaw**(*filename*)
Defines a reddening law from a FITS file.

pysynphot.reddening.**Extinction**(*extinction (E(B-V)) in magnitudes*, *'reddening law'*)
If no name is provided, the average Milky Way extinction will be used. Run the print_red_laws function to see available names.

pysynphot.reddening.**print_red_laws**()
Print information regarding the extinction laws currently available on CDBS. The printed names may be used with the Extinction function to retrieve available reddening laws.

5.13 pysynphot.refs

pysynphot.refs.**getref**()
Collects & returns the current refdata as a dictionary

pysynphot.refs.**set_default_waveset**(*minwave=500*, *maxwave=26000*, *num=10000.0*, *delta=None*, *log=True*)
Set the default waveset for pysynphot spectral types. Calculated wavesets are inclusive of `minwave` and exclusive of `maxwave`.

Parameters

minwave : float, optional

The starting point of the waveset.

maxwave : float, optional

The end point of the waveset.

num : int, optional

The number of elements in the waveset. If `delta` is not None this is ignored.

delta : float, optional

Delta between values in the waveset. If not None, this overrides the `num` parameter. If `log` is True then `delta` is assumed to be the spacing in log space.

log : bool, optional

Sets whether the waveset is evenly spaced in log or linear space. If `log` is True then `delta` is assumed to be the delta in log space. `minwave` and `maxwave` should be given in normal space regardless of the value of `log`.

pysynphot.refs.**setref**(*graphtable=None*, *comptable=None*, *thermtable=None*, *area=None*, *waveset=None*)
provide user access to global reference data. Graph/comp/therm table names must be fully specified.

pysynphot.refs.**showref**()
Prints the values settable by setref

5.13.1 Variables

pysynphot.refs.**_default_waveset = array([500. , 500.19760122, 500.39528054, ..., 25969.1985582 , 25979.46164894, 25**

pysynphot.refs.**_default_waveset_str = 'Min: 500, Max: 26000, Num: 10000.0, Delta: None, Log: True'**
str(object) -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pysynphot.refs.**GRAPHTABLE = None**

pysynphot.refs.**GRAPHDICT = {}**

pysynphot.refs.**COMPTABLE = None**

pysynphot.refs.**COMPDICT = {}**

pysynphot.refs.**THERMTABLE** = None

pysynphot.refs.**THERMDICT** = {}

pysynphot.refs.**PRIMARY_AREA** = 45238.93416
float(x) -> floating point number

Convert a string or number to a floating point number, if possible.

5.14 pysynphot.renorm

pysynphot.renorm.**DefineStdSpectraForUnits**()
Adorn the units with the appropriate kind of spectrum for renormalizing. This is done here to avoid circular imports.

pysynphot.renorm.**StdRenorm**(*spectrum*, *band*, *RNval*, *RNunitstring*, *force=False*)
Another approach to renormalization

5.15 pysynphot.spark

pysynphot.spark.**__version__** = 'SPARK-0.6.1'
str(object) -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

class pysynphot.spark.**GenericASTBuilder**(*AST*, *start*)

buildASTNode(*args*, *lhs*)

nonterminal(*type*, *args*)

preprocess(*rule*, *func*)

terminal(*token*)

class pysynphot.spark.**GenericASTMatcher**(*start*, *ast*)

foundMatch(*args*, *func*)

match(*ast=None*)

match_r(*node*)

preprocess(*rule*, *func*)

resolve(*list*)

class pysynphot.spark.**GenericASTTraversal**(*ast*)

default(*node*)

postorder(*node=None*)

preorder(*node=None*)

prune()

typestring(*node*)

class pysynphot.spark.**GenericASTTraversalPruningException**

class pysynphot.spark.**GenericParser**(*start*)

addRule(*doc*, *func*)

ambiguity(*children*)

augment(*start*)

buildState(*token*, *states*, *i*, *tree*)

buildTree(*tokens*, *tree*, *root*)

buildTree_r(*stack*, *tokens*, *tokpos*, *tree*, *root*)

collectRules()

error(*token*)

makeFIRST()

parse(*tokens*)

preprocess(*rule*, *func*)

resolve(*list*)

typestring(*token*)

class pysynphot.spark.**GenericScanner**

error(*s*, *pos*)

makeRE(*name*)

reflect()

t_default(*s*)
(. | n)+

tokenize(*s*)

5.16 pysynphot.spectrum

class pysynphot.spectrum.**AnalyticSpectrum**(*waveunits='angstrom'*, *fluxunits='photlam'*)
Base class for analytic functions. These are spectral forms which are defined, by default, on top of the default synphot waveset.

All AnalyticSpectra must set wave & flux units; do it here.

GetWaveSet()

class pysynphot.spectrum.**ArraySourceSpectrum**(*wave=None*, *flux=None*, *waveunits='angstrom'*, *fluxunits='photlam'*, *name='UnnamedArraySpectrum'*, *keepneg=False*)
Create a spectrum from arrays.

spec = ArraySpectrum(numpy array containing wavelength table, numpy array containing flux table, waveunits, fluxunits, name=human-readable nickname for spectrum, keepneg=True to override the default behavior of setting negative flux values to zero)

Parameters

wave : ndarray

Wavelength array

flux : ndarray

Flux array

waveunits : `WaveUnits` object or subclass

Units of wave

fluxunits : `FluxUnits` object or subclass

Units of flux

name : string

Description of this array

keepneg : bool [Default: False]

If true, negative flux values will be retained; by default, they are forced to zero

class pysynphot.spectrum.**ArraySpectralElement**(*wave=None*, *throughput=None*, *waveunits='angstrom'*, *name='UnnamedArrayBandpass'*)

spec = ArraySpectrum(numpy array containing wavelength table, numpy array containing throughput table, waveunits, name=human-readable nickname for bandpass.

Create a spectrum from arrays.

Parameters

wave : ndarray

Wavelength array

throughput : ndarray

Throughput array

waveunits : WaveUnits object or subclass

Units of wave

name : string

Description of this spectral element

class pysynphot.spectrum.**BlackBody**(*temperature*)

Blackbody spectrum with specified temperature, in Kelvin.

spec = BlackBody(T in Kelvin)

The flux of the spectrum is normalized to a star of solar radius at a distance of 1 kpc.L

class pysynphot.spectrum.**Box**(*center*, *width*, *waveunits=None*)

bandpass = Box(central wavelength, width) - both in Angstroms

Both center and width are assumed to be in Angstrom units, according to the synphot definition.

class pysynphot.spectrum.**CompositeSourceSpectrum**(*source1*, *source2*, *operation*)

Composite Source Spectrum object, handles addition, multiplication and keeping track of the wavelength set.

GetWaveSet()

Obtain the wavelength set for the composite source by forming the union of wavelengths from each component.

complist()

tabulate()

Evaluate the spectrum in order to return a tabular source spectrum

class pysynphot.spectrum.**CompositeSpectralElement**(*component1*, *component2*)

CompositeSpectralElement Class, which knows how to calculate its throughput by delegating the calculating the calculating to its components.

GetWaveSet()

This method returns a wavelength set appropriate for a composite object by forming the union of the wavelengths of the components.

complist()

wave

wave for CompositeSpectralElement

class pysynphot.spectrum.**FileSourceSpectrum**(*filename*, *fluxname=None*, *keepneg=False*)
Create a spectrum from a file.

spec = FileSpectrum(filename (FITS or ASCII), fluxname=column name containing flux (for FITS tables only), keepneg=True to override thedefault behavior of setting negative flux values to zero)

Parameters
filename : string

FITS or ASCII file containing the spectrum

fluxname : string

Column name specifying the flux (FITS only)

keepneg : bool [Default: False]

If true, negative flux values will be retained; by default, they are forced to zero

class pysynphot.spectrum.**FileSpectralElement**(*filename*, *thrucol=None*)
Create a bandpass from a file.

spec = FileSpectrum(filename (FITS or ASCII), throughputname=column name containing throughput (for FITS tables only), keepneg=True to override the default behavior of setting negative throughput values to zero)

Parameters
filename : string

FITS or ASCII file containing the bandpass

thrucol : string

Column name specifying the throughput (FITS only)

class pysynphot.spectrum.**FlatSpectrum**(*fluxdensity*, *waveunits='angstrom'*, *fluxunits='photlam'*)
Defines a flat spectrum in units of fluxunits.

spec = FlatSpectrum(Flux density, waveunits, fluxunits).

redshift(*z*)
Call the parent's method, which returns a TabularSourceSpectrum, then use its results to create a new FlatSpectrum with the correct value.

class pysynphot.spectrum.**GaussianSource**(*flux*, *center*, *fwhm*, *waveunits='angstrom'*, *fluxunits='flam'*)
Defines a gaussian source

spec = GaussianSource(TotalFlux under Gaussian,
central wavelength of Gaussian, FWHM of Gaussian, waveunits, fluxunits)

Parameters
flux : float

TotalFlux under gaussian

center : float

central wavelength of gaussian

fwhm : float

full-width half-maximum (FWHM) of gaussian

waveunits : string [Default: 'angstrom']

units of input wavelengths

fluxunits : string [Default: 'flam']

units of input fluxes

GetWaveSet()
Return a wavelength set that describes the Gaussian. Overrides the base class to compute 101 values, from center - 5*sigma to center + 5*sigma, in units of 0.1*sigma

class pysynphot.spectrum.**Integrator**
Integrator engine.

trapezoidIntegration(*x*, *y*)

validate_fluxtable()
Enforce non-negative fluxes

validate_wavetable()
Enforce monotonic, ascending wavelengths with no zero values

class pysynphot.spectrum.**InterpolatedSpectralElement**(*fileName*, *wavelength*)
The InterpolatedSpectralElement class handles spectral elements that are interpolated from columns stored in FITS tables

The file name contains a suffix with a column name specification in between square brackets, such as [fr388n#]. The wavelength parameter (poorly named – it is not always a wavelength) is used to interpolate between two columns in the file.

class pysynphot.spectrum.**Powerlaw**(*refwave*, *index*, *waveunits='angstrom'*, *fluxunits='photlam'*)
Defines a power law spectrum

spec=PowerLaw(refwave, exponent, waveunits, fluxunits).

Power law spectrum of the form (lambda/refval)**exponent, where refval is in Angstroms. The spectrum is normalized to a flux of 1 in "fluxunits" at "refval".

class pysynphot.spectrum.**SourceSpectrum**
Base class for the Source Spectrum object.

addmag(*magval*)
Adding a magnitude is like multiplying a flux. Only works for numbers – not arrays, spectrum objects, etc

convert(*targetunits*)
Convert to other units. This method actually just changes the wavelength and flux units objects, it does not recompute the internally kept wave and flux data; these are kept always in internal units. Method getArrays does the actual computation.

effstim(*fluxunits='photlam'*)

getArrays()
Returns wavelength and flux arrays as a tuple, performing units conversion.

integrate(*fluxunits='photlam'*)

redshift(*z*)
Returns a new redshifted spectrum.

renorm(*RNval*, *RNUnits*, *band*, *force=False*)
Renormalize the spectrum to the specified value (in the specified flux units) in the specified band. Calls a function in another module to alleviate circular import issues.

sample(*wave*, *interp=True*)
Return a flux array, in self.fluxunits, on the provided wavetable

setMagnitude(*band*, *value*)
Makes the magnitude of the source in the band equal to value. band is a SpectralElement. This method is marked for deletion once the .renorm method is well tested.

validate_units()
Ensure that waveunits are WaveUnits and fluxunits are FluxUnits

writefits(*filename*, *clobber=True*, *trimzero=True*, *binned=False*, *precision=None*, *hkeys=None*)
Write the spectrum to a FITS file.

Parameters

filename : string
name of file to write to

clobber : bool [Default: True]
Will clobber existing file by default

trimzero : bool [Default:True]
Will trim zero-flux elements from both ends by default

binned : bool [Default: False]
Will write in native waveset by default

precision : {'single','double', None}
Will write in native precision by default

hkeys : dict
Optional dictionary of {keyword:(value,comment)} to be added to primary FITS header

flux
Flux property

wave
Wavelength property

class pysynphot.spectrum.**SpectralElement**
Base class for a Spectral Element (e.g. Filter, Detector...).

GetThroughput()
Return the throughput for the internal wavetable.

GetWaveSet()
Return the waveset in the requested units.

ToInternal()
Convert wavelengths to the internal representation of angstroms.. Note: This is not yet used, but should be for safety when creating TabularSpectralElements from files. It will also be necessary for the ArraySpectralElement class that we want to create RSN.

avgwave()
Implement the equation for lambda nought as defined in Koornneef et al 1987, p 836. Should be equivalent to bandpar.avglam = bandpar.avgwv

check_overlap(*other*)
Check whether the wavelength range of other is defined everywhere that the wavelength range of self is defined. Returns "full", "partial", "none". Normally used for checking whether a spectrum is fully defined over the range of a bandpass. Note that the full overlap case is asymmetric: if the range of 'self' extends past the limits of 'other', this will return a partial overlap.

check_sig(*other*)
Only call this if check_overlap returns 'partial'. Returns True if the LACK of overlap is INsignificant: i.e., it is ok to go ahead and do whatever we are doing.

convert(*targetunits*)
Convert to other units. This method actually just changes the wavelength unit objects, it does not recompute the internally kept wave data; these are kept always in internal units. Method getWaveSet does the actual computation.

efficiency()
QTLAM = dimensionless efficience = INT(THRU / LAM)

equivwidth(*THRU*)

fwhm()

integrate(*wave=None*)
Integrate the throughput over the specified waveset, if None, integrate over the full waveset.

photbw(*floor=0*)
This is a compatibility function allowing pysynphot to calculate the bandpass RMS width in the same way as Synphot (documented in the Synphot Manual section 5.1). This is the value returned in the BANDW keyword by Synphot's bandpar function.

This function is designed only for use to get the same results as Synphot. To calculate the bandpass RMS width use the `rmswidth` method.

Parameters
floor : float, optional
Points with throughputs below this threshold are not included in the calculation. By default all points are included.

Returns
photbw : float
RMS width of the bandpass.

pivot(*binned=False*)
This is the calculation performed when the ETC invokes calcphot. Does this need to be calculated on binned waveset, or may it be calculated on native waveset?

rectwidth(*THRU) / MAX(THRU*)

resample(*resampledWaveTab*)
Interpolate throughput given a wavelength array that is monotonically increasing and the TabularSpectralElement object.

rmswidth(*floor=0*)
Calculate the RMS width as in Koornneef et al 1987, p 836.

Parameters
floor : float, optional
Points with throughputs below this threshold are not included in the calculation. By default all points are included.

Returns
rmswidth : float
RMS width of the bandpass.

sample(*wave*)
Provide a more normal user interface to the __call__

taper()
Taper the spectrum by adding zeros to each end.

unit_response()
Returns flux, in flam, of a star that produces a response of one photon per second in this passband.

Only correct if waveunits are Angstrom.

validate_units()
Ensure that waveunits are WaveUnits

writefits(*filename*, *clobber=True*, *trimzero=True*, *precision=None*, *hkeys=None*)
Write the bandpass to a FITS file.

Parameters

filename : string
name of file to write to

clobber : bool [Default: True]
Will clobber existing file by default

trimzero : bool [Default: True]
Will trim zero-flux elements from both ends by default

precision : {'single','double',None}
Will write in native precision by default

hkeys : dict, optional
Optional dictionary of {keyword:(value,comment)}
to be added to primary FITS header

throughput
Throughput for bandpass

wave
Waveset for bandpass

class pysynphot.spectrum.**TabularSourceSpectrum**(*filename=None*, *fluxname=None*, *keepneg=False*)
Class for a source spectrum that is read in from a table.

GetWaveSet()
For a TabularSource Spectrum, the WaveSet is just the _wavetable member. Return a copy so that there is no reference to the original object.

ToInternal()
Convert to the internal representation of (angstroms, photlam).

resample(*resampledWaveTab*)
Interpolate flux given a wavelength array that is monotonically increasing and the TabularSourceSpectrum object.

Parameters

resampledWaveTab : ndarray
new wavelength table IN ANGSTROMS

taper()
Taper the spectrum by adding zeros to each end.

class pysynphot.spectrum.**TabularSpectralElement**(*FITS or ASCII filename*, *thrucol= name of column containing throughput values (for FITS tables only*)
__init__ takes a character string argument that contains the name of the file with the spectral element table.

ToInternal()
Convert wavelengths to the internal representation of angstroms..

getHeaderKeywords(*header*)
This is a placeholder for subclasses to get header keywords without having to reopen the file again.

class pysynphot.spectrum.**ThermalSpectralElement**(*fileName*)
The ThermalSpectralElement class handles spectral elements that have associated thermal properties read from a FITS table.

ThermalSpectralElements differ from regular SpectralElements in that they carry thermal parameters such as temperature and beam filling factor, but otherwise they operate just as regular SpectralElements. They dont know how to apply themselves to an existing beam, in the sense that their emissivities should be handled explictly, outside the objects themselves.

getHeaderKeywords(*header*)
Overrides base class in order to get thermal keywords.

class pysynphot.spectrum.**UniformTransmission**(*dimensionless throughput*)

GetWaveSet()

check_overlap(*spectrum*)
Apply special overlap logic for UniformTransmission.

By definition, a UniformTransmission is defined everywhere. Therefore, this is a special case for which the overlap check should be ignored (because the alternative is that it will always fail and always require users to override it, so it becomes meaningless).

pysynphot.spectrum.**MergeWaveSets**(*waveset1*, *waveset2*)
Return the union of the two wavesets, unless one or both of them is None.

pysynphot.spectrum.**trimSpectrum**(*sp*, *minw*, *maxw*)
Creates a new spectrum with trimmed upper and lower ranges.

5.17 pysynphot.spparser

This file implements the pysynphot language parser.

The language definition is in the docstring of class BaseParser, function p_top. The parser code in spark.py builds its internal tables by reading the docstring, so you can't put anything else (like documentation) there.

```
l = scan('text') returns a list of tokens

t = parse(l) converts the list of tokens into an Abstract Syntax Tree

r = interpret(t) converts that abstract syntax tree into a (tree
  of?) pysynphot object, based on the conversion rules in class Interpreter
```

In class Interpreter, the docstring of every function named with p_ is part of the instructions to the parser.

class pysynphot.spparser.**AST**(*type*)

class pysynphot.spparser.**BaseParser**(*ASTclass*, *start='top'*)

nonterminal(*type*, *args*)

p_top(*args*)
top ::= expr top ::= FILELIST expr ::= expr + term expr ::= expr - term expr ::= term term ::= term * factor term ::= term / factor value ::= LPAREN expr RPAREN term ::= factor factor ::= unaryop value factor ::= value unaryop ::= + unaryop ::= - value ::= INTEGER value ::= FLOAT value ::= IDENTIFIER value ::=

function_call function_call ::= IDENTIFIER LPAREN arglist RPAREN arglist ::= arglist , expr arglist ::= expr

terminal(*token*)

class pysynphot.spparser.**BaseScanner**

t_comma(*s*)
,

t_filelist(*s*)
@w+

t_identifier(*s*)
[$a-z_A-Z///][w/.$:#]*

t_integer(*s*)
d+

t_lparens(*s*)
(

t_op(*s*)
+ | * | -

t_rparens(*s*)
)

t_whitespace(*s*)
s+

tokenize(*input*)

class pysynphot.spparser.**Interpreter**(*ast*)

error(*token*)

p_arglist(*tree*)
V ::= arglist (V , V)

p_expr_minus(*tree*)
V ::= expr (V - V)

p_expr_plus(*tree*)
V ::= expr (V + V)

p_factor_unary_minus(*tree*)
V ::= factor (- V)

p_factor_unary_plus(*tree*)
V ::= factor (+ V)

p_float(*tree*)
V ::= FLOAT

p_functioncall(*tree*)
V ::= function_call (V LPAREN V RPAREN)

p_identifier(*tree*)
V ::= IDENTIFIER

p_int(*tree*)
V ::= INTEGER

p_term_div(*tree*)
V ::= term (V / V)

p_term_mult(*tree*)
V ::= term (V * V)

p_value_paren(*tree*)
V ::= value (LPAREN V RPAREN)

class pysynphot.spparser.**Scanner**

t_divop(*s*)
s/s

t_float(*s*)
((d*.d+)|(d+.d*)|(d+)) ([eE][-+]?d+)?

class pysynphot.spparser.**Token**(*type=None*, *attr=None*)

pysynphot.spparser.**convertstr**(*value*)

pysynphot.spparser.**interpret**(*ast*)

pysynphot.spparser.**parse**(*tokens*)

pysynphot.spparser.**parse_spec**(*syncommand*)
Parse the synphot-classic command and return the resulting spectrum

pysynphot.spparser.**ptokens**(*tlist*)

pysynphot.spparser.**scan**(*input*)

5.17.1 Global Variables

pysynphot.spparser.**syfunctions = ['spec', 'unit', 'box', 'bb', 'pl', 'em', 'icat', 'rn', 'z', 'ebmvx', 'band']**
list() -> new empty list list(iterable) -> new list initialized from iterable's items

pysynphot.spparser.**synforms = ['fnu', 'flam', 'photnu', 'photlam', 'counts', 'abmag', 'stmag', 'obmag', 'vegamag', 'jy**
list() -> new empty list list(iterable) -> new list initialized from iterable's items

pysynphot.spparser.**syredlaws = ['gal1', 'gal2', 'gal3', 'smc', 'lmc', 'xgal']**
list() -> new empty list list(iterable) -> new list initialized from iterable's items

5.18 pysynphot.tables

class pysynphot.tables.**CompTable**(*CFile=None*)

CompTable class; opens the specified comptable and populates 1-d arrays of component names and file names in the members compnames and filenames

__init__ instantiates the CompTable object, given the comptable file name as an input string.

Parameters

Input : ndarray of chars

string CFile containing comptable name

Effect : ndarray of chars

populates two data members: compnames and filenames

class pysynphot.tables.**GraphTable**(*GFile=None*)

GraphTable class; opens the specified graph table and populates 1-d arrays of keyword names, innodes, outnodes and component names in the members keywords, innodes, outnodes and compnames

__init__ instantiates the GraphTable object, given the graph table name as an input string.

Parameters

Input : string

GFile containing graph table name

Effect : dict

populates four data members:

```
keywords: CharArray of keyword names
innodes:  Int32 array of innodes
outnodes: Int32 array of outnodes
compnames:CharArray of components names
```

GetComponentsFromGT(*modes*, *innode*)

GetComponentsFromGT returns two lists of component names corresponding to those obtained by waling down the graph table starting at innode. The first list contains the optical components, the second list, the thermal components.

GetNextNode(*modes*, *innode*)

GetNextNode returns the outnode that matches an element from the modes list, starting at the given innode. This method isnt actually used, its just a helper method for debugging purposes

pysynphot.tables.**DEBUG = False**

bool(x) -> bool

Returns True when the argument x is true, False otherwise. The builtins True and False are the only two instances of the class bool. The class bool is a subclass of the class int, and cannot be subclassed.

5.19 pysynphot.units

Units class hierarchy: is used to manage both wavelength and flux unit conversions

Warning: vegamag unit conversions require spectrum and locations modules => circular imports.

class pysynphot.units.**ABMag**

ToPhotlam(*wave*, *flux*, ***kwargs*)

unitResponse(*band*)

StdSpectrum = <pysynphot.spectrum.FlatSpectrum object at 0x1075a0190>

class pysynphot.units.**Angstrom**

ToAngstrom(*wave*)

ToCm(*wave*)

ToHz(*wave*)

ToInverseMicron(*wave*)

ToMeter(*wave*)

ToMicron(*wave*)

ToMm(*wave*)

ToNm(*wave*)

class pysynphot.units.**BaseUnit**(*uname*)
Base class for all units; defines UI

Convert(*wave*, *flux*, *target_units*)

class pysynphot.units.**Cm**

class pysynphot.units.**Counts**

ToPhotlam(*wave*, *flux*, *area=None*)

unitResponse(*band*)

StdSpectrum = <pysynphot.spectrum.CompositeSourceSpectrum object at 0x108837c10>

class pysynphot.units.**Flam**
flam = erg cm^-2 s^-1 Ang^-1

ToPhotlam(*wave*, *flux*, ***kwargs*)

unitResponse(*band*)

StdSpectrum = <pysynphot.spectrum.FlatSpectrum object at 0x108837cd0>

class pysynphot.units.**FluxUnits**
All FluxUnits know how to convert themselves to Photlam

Convert(*wave*, *flux*, *target_units*, *area=None*)
FluxUnits need both wavelength and flux tables to do a unit conversion.

ToPhotlam(*wave*, *flux*, *area=None*)

class pysynphot.units.**Fnu**
fnu = erg cm^-2 s^-1 Hz^-1

ToPhotlam(*wave*, *flux*, ***kwargs*)

unitResponse(*band*)

StdSpectrum = <pysynphot.spectrum.FlatSpectrum object at 0x1088379d0>

class pysynphot.units.**Hz**

ToAngstrom(*wave*)

class pysynphot.units.**InverseMicron**

ToAngstrom(*wave*)

class pysynphot.units.**Jy**
jy = 10^-23 erg cm^-2 s^-1 Hz^-1

ToPhotlam(*wave*, *flux*, ***kwargs*)

unitResponse(*band*)

StdSpectrum = <pysynphot.spectrum.FlatSpectrum object at 0x108837590>

class pysynphot.units.**LogFluxUnits**
Base class for magnitudes, which often require special handling

class pysynphot.units.**Meter**

class pysynphot.units.**Micron**

class pysynphot.units.**Mm**

class pysynphot.units.**Nm**

class pysynphot.units.**OBMag**

ToPhotlam(*wave*, *flux*, *area=None*)

unitResponse(*band*)

StdSpectrum = <pysynphot.spectrum.CompositeSourceSpectrum object at 0x1075a0750>

class pysynphot.units.**Photlam**
photlam = photons cm^-2 s^-1 Ang^-1)

ToABMag(*wave*, *flux*, ***kwargs*)

ToCounts(*wave*, *flux*, *area=None*)

ToFlam(*wave*, *flux*, ***kwargs*)

ToFnu(*wave*, *flux*, ***kwargs*)

ToJy(*wave*, *flux*, ***kwargs*)

ToOBMag(*wave*, *flux*, *area=None*)

ToPhotlam(*wave*, *flux*, ***kwargs*)

ToPhotnu(*wave*, *flux*, ***kwargs*)

ToSTMag(*wave*, *flux*, ***kwargs*)

ToVegaMag(*wave*, *flux*, ***kwargs*)

TomJy(*wave*, *flux*, ***kwargs*)

TomuJy(*wave*, *flux*, ***kwargs*)

TonJy(*wave*, *flux*, ***kwargs*)

unitResponse(*band*)
Put a flat spectrum of 1 photlam through this band, & integrate

StdSpectrum = <pysynphot.spectrum.FlatSpectrum object at 0x108837bd0>

class pysynphot.units.**Photnu**
photnu = photon cm^-2 s^-1 Hz^-1

ToPhotlam(*wave*, *flux*, ***kwargs*)

unitResponse(*band*)

StdSpectrum = <pysynphot.spectrum.FlatSpectrum object at 0x1088374d0>

class pysynphot.units.**STMag**

ToPhotlam(*wave*, *flux*, ***kwargs*)

unitResponse(*band*)

StdSpectrum = <pysynphot.spectrum.FlatSpectrum object at 0x1075a0050>

class pysynphot.units.**VegaMag**

ToPhotlam(*wave*, *flux*, ***kwargs*)

unitResponse(*band*)

StdSpectrum = <pysynphot.spectrum.FileSourceSpectrum object at 0x102a70c50>

class pysynphot.units.**WaveUnits**
All WaveUnits know how to convert themselves to Angstroms

Convert(*wave*, *target_units*)
WaveUnits only need a wavelength table to do a conversion.

ToAngstrom(*wave*)

class pysynphot.units.**mJy**
mjy = 10^-26 erg cm^-2 s^-1 Hz^-1

ToPhotlam(*wave*, *flux*, ***kwargs*)

unitResponse(*band*)

StdSpectrum = <pysynphot.spectrum.FlatSpectrum object at 0x108837dd0>

class pysynphot.units.**muJy**
mujy = 10^-29 erg cm^-2 s^-1 Hz^-1

ToPhotlam(*wave*, *flux*, ***kwargs*)

unitResponse(*band*)

class pysynphot.units.**nJy**
njy = 10^-32 erg cm^-2 s^-1 Hz^-1

ToPhotlam(*wave*, *flux*, ***kwargs*)

unitResponse(*band*)

pysynphot.units.**Units**(*uname*)
This needs to be a factory function in order to return an object of the correct subclass.

pysynphot.units.**factory**(*uname*, **args*, ***kwargs*)

pysynphot.units.**ismatch**(*a*, *b*)
Method to allow smart comparisons between classes, instances, and string representations of units and give the right answer.

5.20 pysynphot.wavetable

class pysynphot.wavetable.**Wavetable**(*fname*)
Class to handle wavecat.dat initialization and access. (This class may need a better name; wavetable and waveset are awfully close.) Also, put the default waveset into this object with a key of NONE.

Instantiate a Wavetable from a file

5.20.1 Global Variables

pysynphot.wavetable.**wavecat_file = '/Users/sienkiew/plugh/lib/python2.7/site-packages/pysynphot-0.9.5-py2.7-macos**
str(object) -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

pysynphot.wavetable.**wavetable = <pysynphot.wavetable.Wavetable object at 0x102a79490>**
Class to handle wavecat.dat initialization and access. (This class may need a better name; wavetable and waveset are awfully close.) Also, put the default waveset into this object with a key of NONE.

5.21 SVN Version

pysynphot.svn_version.**__svn_version__ = '3373'**
str(object) -> string

Return a nice string representation of the object. If the argument is a string, the return value is the same object.

CHAPTER

SIX

INDICES AND TABLES

- *genindex*
- *modindex*
- *search*

p

p

Symbols

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

T

U

V

W

X

Z