pyuvdata Documentation

Release 1.0.0

HERA Collaboration

February 15, 2017

Contents

pyuvdata defines a pythonic interface to interferometric data sets. Currently pyuvdata supports reading and writing of miriad and uvfits files and reading of FHD (Fast Holographic Deconvolution) visibility save files.

CHAPTER 1

Motivation

The three main goals are:

1. To provide a high quality, well documented path to convert between data formats
2. Support the direct use of datasets from python with minimal software
3. Provide precise data definition via both human readable code and high quality online documentation

CHAPTER 2

Package Details

2.1 Tested File Paths

- uvfits -> miriad (aipy)
- miriad (aipy) -> uvfits
- FHD -> uvfits
- FHD -> miriad (aipy)

2.2 File standards

- miriad is supported for aipy-style analysis, further testing is required for use in the miriad package
- uvfits conforms to AIPS memo 117 (as of May 2015). It is tested against FHD, CASA, and AIPS. However AIPS is limited to <80 antennas and CASA imaging does not seem to support >255 antennas.
- FHD (read-only support, tested against MWA and PAPER data)

2.3 Known Issues and Planned Improvements

- different multiple spectral windows or multiple sources are not currently supported
- testing against miriad package
- replacing AIPY and pyephem with astropy+NOVAS for time and phase calculations
- support for direct reading and writing of Measurement Sets
- support for calibration solutions: define a cal object with read/write support for FITS and other formats

For details see the issue log.

2.4 Community Guidelines

Contributions to this package to add new file formats or address any of the issues in the issue log are very welcome. Please submit improvements as pull requests against the repo after verifying that the existing tests pass and any new code is well covered by unit tests.

Bug reports or feature requests are also very welcome, please add them to the issue log after verifying that the issue does not already exist. Comments on existing issues are also welcome.

CHAPTER 3

History

pyuvdata was originally developed in the low frequency 21cm community to support the development of calibration and foreground subtraction pipelines. Particular focus has been paid to supporting drift and phased array modes.

CHAPTER 4

Installation

4.1 Dependencies

First install dependencies. The numpy and astropy versions are important, so be sure to make sure these are up to date before you install.

- numpy >= 1.10
- scipy
- astropy >= 1.2
- pyephem
- aipy

4.2 Install pyuvdata

For simple installation, the latest stable version is available via pip using `pip install pyuvdata`

4.2.1 Optionally install the development version

For the development version, clone the repository using `git clone https://github.com/HERA-Team/pyuvdata/releas`

Navigate into the directory and run `python setup.py install`. Note that this will automatically install all dependencies. If you use anaconda or another package manager you might prefer not to do this.

To install without dependencies, run `python setup.py install --no-dependencies`

4.3 Tests

Requires installation of `nose` package. From the source pyuvdata directory run `nosetests pyuvdata`.

CHAPTER 5

API

The primary interface to data from python is via the UVData object. It provides import and export functionality to all supported file formats (UVFITS, Miriad, FHD) and can be interacted with directly. The attributes of the UVData object are described in the parameters description.

CHAPTER 6

Further Documentation

6.1 Tutorial

6.1.1 Example 1

Converting between tested data formats

a) miriad (aipy) -> uvfits

```
from pyuvdata import UVData
UV = UVData()
miriad_file = 'pyuvdata/data/new.uvA'
UV.read_miriad(miriad_file)  # this miriad file is known to be a drift scan
UV.write_uvfits('new.uvfits', force_phase=True, spoof_nonessential=True)  # write out the uvfits file
```

b) uvfits -> miriad (aipy)

```
from pyuvdata import UVData
UV = UVData()
uvfits_file = 'pyuvdata/data/day2_TDEM0003_10s_norx_1src_1spw.uvfits'
UV.read_uvfits(uvfits_file)
UV.write_uvfits('day2_TDEM0003_10s_norx_1src_1spw.uv')  # write out the miriad file
```

c) FHD -> uvfits

When reading FHD format, we need to point to several files.:

```
from pyuvdata import UVData
UV = UVData()
fhd_prefix = 'pyuvdata/data/fhd_vis_data/1061316296_'
# Construct the list of files
fhd_files = [fhd_prefix + f for f in ['flags.sav', 'vis_XX.sav', 'params.sav',
                                      'vis_YY.sav', 'vis_model_XX.sav',
                                      'vis_model_YY.sav', 'settings.txt']]
UV.read_fhd(fhd_files)
UV.write_uvfits('1061316296.uvfits', spoof_nonessential=True)
```

d) FHD -> miriad (aipy)

```
from pyuvdata import UVData
UV = UVData()
fhd_prefix = 'pyuvdata/data/fhd_vis_data/1061316296_'
# Construct the list of files
fhd_files = [fhd_prefix + f for f in ['flags.sav', 'vis_XX.sav', 'params.sav',
                                      'vis_YY.sav', 'vis_model_XX.sav',
                                      'vis_model_YY.sav', 'settings.txt']]
UV.read_fhd(fhd_files)
UV.write_uvfits('1061316296.uvfits')
```

6.1.2 Example 2

Phasing/unphasing data:

```
from pyuvdata import UVData
import ephem
UV = UVData()
miriad_file = 'pyuvdata/data/new.uvA'
UV.read_miriad(miriad_file)
print(UV.phase_type)  # Data is a drift scan
UV.phase_to_time(UV.time_array[0])  # Phases the data to the zenith at first time step
print(UV.phase_type)  # Data should now be phased
UV.unphase_to_drift()  # Undo phasing to try another phase center
UV.phase(5.23368, 0.710940, ephem.J2000)  # Phase to a specific ra/dec/epoch (in radians)
```

6.1.3 Example 3

Making a simple waterfall plot:

```
from pyuvdata import UVData
import numpy as np
import matplotlib.pyplot as plt
UV = UVData()
filename = 'pyuvdata/data/day2_TDEM0003_10s_norx_1src_1spw.uvfits'
UV.read_uvfits(filename)
print(UV.data_array.shape)  # Data should have shape (Nblts, Nspws, Nfreqs, Npols)
print(UV.Ntimes)  # Number of time samples in data
print(UV.Nfreqs)  # Number of frequency channels in data
bl = UV.antnums_to_baseline(1, 2)  # Convert antenna numbers  (e.g. 1, 2) to baseline number
bl_ind = np.where(UV.baseline_array == bl)[0]  # Indices corresponding to baseline
plt.imshow(np.abs(UV.data_array[bl_ind, 0, :, 0]))  # Amplitude waterfall for 0th spectral window an
plt.show()
```

6.2 Parameters

These are the standard attributes of UVData objects.

Under the hood they are actually properties based on UVParameter objects.

Angle type attributes also have convenience properties named the same thing with '_degrees' appended through which you can get or set the value in degrees.

Similarly location type attributes (which are given in topocentric xyz coordinates) have convenience properties named the same thing with '_lat_lon_alt' and '_lat_lon_alt_degrees' appended through which you can get or set the values using latitude, longitude and altitude values in radians or degrees and meters.

6.2.1 Required

These parameters are required to have a sensible UVData object and are required for most kinds of uv data files.

Nants_data Number of antennas with data present (i.e. number of unique entries in ant_1_array and ant_2_array). May be smaller than the number of antennas in the array

Nants_telescope Number of antennas in the array. May be larger than the number of antennas with data

Nbls Number of baselines

Nblts Number of baseline-times (i.e. number of spectra). Not necessarily equal to Nbls * Ntimes

Nfreqs Number of frequency channels

Npols Number of polarizations

Nspws Number of spectral windows (ie non-contiguous spectral chunks). More than one spectral window is not currently supported.

Ntimes Number of times

ant_1_array Array of first antenna indices, shape (Nblts), type = int, 0 indexed

ant_2_array Array of second antenna indices, shape (Nblts), type = int, 0 indexed

antenna_names List of antenna names, shape (Nants_telescope), with numbers given by antenna_numbers (which can be matched to ant_1_array and ant_2_array). There must be one entry here for each unique entry in ant_1_array and ant_2_array, but there may be extras as well.

antenna_numbers List of integer antenna numbers corresponding to antenna_names, shape (Nants_telescope). There must be one entry here for each unique entry in ant_1_array and ant_2_array, but there may be extras as well.

baseline_array Array of baseline indices, shape (Nblts), type = int; baseline = 2048 * (ant2+1) + (ant1+1) + 2^16

channel_width Width of frequency channels (Hz)

data_array Array of the visibility data, shape: (Nblts, Nspws, Nfreqs, Npols), type = complex float, in units of self.vis_units

flag_array Boolean flag, True is flagged, same shape as data_array.

freq_array Array of frequencies, shape (Nspws, Nfreqs), units Hz

history String of history, units English

instrument Receiver or backend. Sometimes identical to telescope_name

integration_time Length of the integration (s)

lst_array Array of lsts, center of integration, shape (Nblts), units radians

nsample_array Number of data points averaged into each data element, NOT required to be an integer. type = float, same shape as data_array

object_name Source or field observed (string)

phase_type String indicating phasing type. Allowed values are "drift", "phased" and "unknown"

polarization_array Array of polarization integers, shape (Npols). AIPS Memo 117 says: stokes 1:4 (I,Q,U,V); circular -1:-4 (RR,LL,RL,LR); linear -5:-8 (XX,YY,XY,YX)

spw_array Array of spectral window Numbers, shape (Nspws)

telescope_location Telescope location: xyz in ITRF (earth-centered frame). Can also be accessed using telescope_location_lat_lon_alt or telescope_location_lat_lon_alt_degrees properties

telescope_name Name of telescope (string)

time_array Array of times, center of integration, shape (Nblts), units Julian Date

uvw_array Projected baseline vectors relative to phase center, shape (3, Nblts), units meters

vis_units Visibility units, options are: "uncalib", "Jy" or "K str"

6.2.2 Optional

These parameters are defined by one or more file standard but are not always required. Some of them are required depending on the phase_type (as noted below).

antenna_positions Array giving coordinates of antennas relative to telescope_location (ITRF frame), shape (Nants_telescope, 3)

dut1 DUT1 (google it) AIPS 117 calls it UT1UTC

earth_omega Earth's rotation rate in degrees per day

extra_keywords Any user supplied extra keywords, type=dict

gst0 Greenwich sidereal time at midnight on reference date

phase_center_dec Required if phase_type = "phased". Declination of phase center (see uvw_array), units radians. Can also be accessed using phase_center_dec_degrees.

phase_center_epoch Required if phase_type = "phased". Epoch year of the phase applied to the data (eg 2000.)

phase_center_ra Required if phase_type = "phased". Right ascension of phase center (see uvw_array), units radians. Can also be accessed using phase_center_ra_degrees.

rdate Date for which the GST0 or whatever... applies

timesys We only support UTC

uvplane_reference_time FHD thing we do not understand, something about the time at which the phase center is normal to the chosen UV plane for phasing

zenith_dec Required if phase_type = "drift". Declination of zenith. units: radians, shape (Nblts). Can also be accessed using zenith_dec_degrees.

zenith_ra Required if phase_type = "drift". Right ascension of zenith. units: radians, shape (Nblts). Can also be accessed using zenith_ra_degrees.

last updated: 2017-02-14

6.3 UVData Class

UVData is the main user class. It provides import and export functionality to all supported file formats (UVFITS, Miriad, FHD) and can be interacted with directly.

class `pyuvdata.`**`UVData`**
A class for defining a radio interferometer dataset.

Currently supported file types: uvfits, miriad, fhd. Provides phasing functions.

UVParameter objects
For full list see Parameters (docs/parameters.rst or https://pyuvdata.readthedocs.io/en/latest/parameters.html). Some are always required, some are required for certain phase_types and others are always optional.

antnums_to_baseline(*ant1*, *ant2*, *attempt256=False*)
Get the baseline number corresponding to two given antenna numbers.

Parameters

- **ant1** – first antenna number (integer)
- **ant2** – second antenna number (integer)
- **attempt256** – Option to try to use the older 256 standard used in many uvfits files (will use 2048 standard if there are more than 256 antennas). Default is False.

Returns integer baseline number corresponding to the two antenna numbers.

baseline_to_antnums(*baseline*)
Get the antenna numbers corresponding to a given baseline number.

Parameters **baseline** – integer baseline number

Returns tuple with the two antenna numbers corresponding to the baseline.

check(*run_sanity_check=True*)
Add some extra checks on top of checks on UVBase class.

Check that all required parameters are set reasonably.

Check that required parameters exist and have appropriate shapes. Optionally check if the values are sane.

Parameters **run_sanity_check** – Option to check if values in required parameters are sane. Default is True.

juldate2ephem(*num*)
Convert Julian date to ephem date, measured from noon, Dec. 31, 1899.

Parameters **num** – Julian date

Returns ephem date, measured from noon, Dec. 31, 1899.

known_telescopes()
Retun a list of telescopes known to pyuvdata.

This is just a shortcut to uvdata.telescopes.known_telescopes()

phase(*ra*, *dec*, *epoch*)
Phase a drift scan dataset to a single ra/dec at a particular epoch.

Will not phase already phased data.

Parameters

- **ra** – The ra to phase to in radians.
- **dec** – The dec to phase to in radians.
- **epoch** – The epoch to use for phasing. Should be an ephem date, measured from noon Dec. 31, 1899.

phase_to_time(*time*)
Phase a drift scan dataset to the ra/dec of zenith at a particular time.

Parameters **time** – The time to phase to.

read_fhd (*filelist*, *use_model=False*, *run_check=True*, *run_sanity_check=True*)
Read in data from a list of FHD files.

Parameters

- **filelist** – The list of FHD save files to read from. Must include at least one polarization file, a params file and a flag file.
- **use_model** – Option to read in the model visibilities rather than the dirty visibilities. Default is False.
- **run_check** – Option to check for the existence and proper shapes of required parameters after reading in the file. Default is True.
- **run_sanity_check** – Option to sanity check the values of required parameters after reading in the file. Default is True.

read_miriad (*filepath*, *correct_lat_lon=True*, *run_check=True*, *run_sanity_check=True*)
Read in data from a miriad file.

Parameters

- **filepath** – The miriad file directory to read from.
- **run_check** – Option to check for the existence and proper shapes of required parameters after reading in the file. Default is True.
- **run_sanity_check** – Option to sanity check the values of required parameters after reading in the file. Default is True.

read_uvfits (*filename*, *run_check=True*, *run_sanity_check=True*)
Read in data from a uvfits file.

Parameters

- **filename** – The uvfits file to read from.
- **run_check** – Option to check for the existence and proper shapes of required parameters after reading in the file. Default is True.
- **run_sanity_check** – Option to sanity check the values of required parameters after reading in the file. Default is True.

set_drift ()
Set phase_type to 'drift' and adjust required parameters.

set_lsts_from_time_array ()
Set the lst_array based from the time_array.

set_phased ()
Set phase_type to 'phased' and adjust required parameters.

set_telescope_params (*overwrite=False*)
Set telescope related parameters.

If the telescope_name is in the known_telescopes, set any missing telescope-associated parameters (e.g. telescope location) to the value for the known telescope.

Parameters **overwrite** – Option to overwrite existing telescope-associated parameters with the values from the known telescope. Default is False.

set_unknown_phase_type ()
Set phase_type to 'unknown' and adjust required parameters.

unphase_to_drift()
Convert from a phased dataset to a drift dataset.

write_miriad(*filepath*, *run_check=True*, *run_sanity_check=True*, *clobber=False*)
Write the data to a miriad file.

Parameters

- **filename** – The miriad file directory to write to.
- **run_check** – Option to check for the existence and proper shapes of required parameters before writing the file. Default is True.
- **run_sanity_check** – Option to sanity check the values of required parameters before writing the file. Default is True.
- **clobber** – Option to overwrite the filename if the file already exists. Default is False.

write_uvfits(*filename*, *spoof_nonessential=False*, *force_phase=False*, *run_check=True*, *run_sanity_check=True*)
Write the data to a uvfits file.

Parameters

- **filename** – The uvfits file to write to.
- **spoof_nonessential** – Option to spoof the values of optional UVParameters that are not set but are required for uvfits files. Default is False.
- **force_phase** – Option to automatically phase drift scan data to zenith of the first timestamp. Default is False.
- **run_check** – Option to check for the existence and proper shapes of required parameters before writing the file. Default is True.
- **run_sanity_check** – Option to sanity check the values of required parameters before writing the file. Default is True.

6.4 Developer Docs

Documentation for all the under-the-hood classes and functions that most users won't need to interact with.

6.4.1 Classes

class parameter.**UVParameter**(*name*, *required=True*, *value=None*, *spoof_val=None*, *form=()*, *description=''*, *expected_type=None*, *sane_vals=None*, *sane_range=None*, *tols=(1e-05, 1e-08)*)
Data and metadata objects for interferometric data sets.

name
A string giving the name of the attribute. Used as the associated property name in classes based on UVBase.

required
A boolean indicating whether this is required metadata for the class with this UVParameter as an attribute. Default is True.

value
The value of the data or metadata.

spoof_val
A fake value that can be assigned to a non-required UVParameter if the metadata is required for a particular file-type. This is not an attribute of required UVParameters.

form
Either 'str' or a tuple giving information about the expected shape of the value. Elements of the tuple may be the name of other UVParameters that indicate data shapes.

Examples

'str': a string value

('Nblts', 3): the value should be an array of shape: Nblts (another UVParameter name), 3

description
A string description of the data or metadata in the object.

expected_type
The type that the data or metadata should be. Default is np.int or str if form is 'str'

sane_vals
Optional. List giving allowed values for elements of value.

sane_range
Optional. Tuple giving a range of allowed magnitudes for elements of value.

tols
Tolerances for testing the equality of UVParameters. Either a single absolute value or a tuple of relative and absolute values to be used by np.isclose()

__eq__(*other*)
Equal if classes match and values are identical.

__ne__(*other*)
Not equal.

apply_spoof()
Set value to spoof_val for non-required UVParameters.

expected_shape(*uvbase*)
Get the expected shape of the value based on the form.

Parameters **uvbase** – object with this UVParameter as an attribute. Needed because the form can refer to other UVParameters on this object.

Returns The expected shape of the value.

sanity_check()
Check that values are sane.

class parameter.**AntPositionParameter**(*name*, *required=True*, *value=None*, *spoof_val=None*, *form=()*, *description=''*, *expected_type=None*, *sane_vals=None*, *sane_range=None*, *tols=(1e-05*, *1e-08)*)
Subclass of UVParameter for antenna positions.

Overrides apply_spoof method to generate an array of the correct shape based on the number of antennas on the object with this UVParameter as an attribute.

apply_spoof(*uvbase*, *antnum_name*)
Set value to zeroed array of shape: number of antennas, 3.

Parameters

- **uvbase** – object with this UVParameter as an attribute. Needed to get the number of antennas.
- **antnum_name** – A string giving the name of the UVParameter containing the number of antennas.

class parameter.**AngleParameter**(*name*, *required=True*, *value=None*, *spoof_val=None*, *form=()*, *description=''*, *expected_type=None*, *sane_vals=None*, *sane_range=None*, *tols=(1e-05*, *1e-08)*)

Subclass of UVParameter for Angle type parameters.

Adds extra methods for conversion to & from degrees (used by UVBase objects for _degrees properties associated with these parameters).

degrees()
Get value in degrees.

set_degrees(*degree_val*)
Set value in degrees.

Parameters **degree_val** – value in degrees to use to set the value attribute.

class parameter.**LocationParameter**(*name*, *required=True*, *value=None*, *spoof_val=None*, *description=''*, *sane_range=(6350000.0*, *6390000.0)*, *tols=0.001*)

Subclass of UVParameter for Earth location type parameters.

Adds extra methods for conversion to & from lat/lon/alt in radians or degrees (used by UVBase objects for _lat_lon_alt and _lat_lon_alt_degrees properties associated with these parameters).

lat_lon_alt()
Get value in (latitude, longitude, altitude) tuple in radians.

lat_lon_alt_degrees()
Get value in (latitude, longitude, altitude) tuple in degrees.

sanity_check()
Check that values are sane. Special case for location, where we want to check the vector magnitude

set_lat_lon_alt(*lat_lon_alt*)
Set value from (latitude, longitude, altitude) tuple in radians.

Parameters **lat_lon_alt** – tuple giving the latitude (radians), longitude (radians) and altitude to use to set the value attribute.

set_lat_lon_alt_degrees(*lat_lon_alt_degree*)
Set value from (latitude, longitude, altitude) tuple in degrees.

Parameters **lat_lon_alt** – tuple giving the latitude (degrees), longitude (degrees) and altitude to use to set the value attribute.

class uvbase.**UVBase**
Base class for objects with UVParameter attributes.

This class is intended to be subclassed and its init method should be called in the subclass init after all associated UVParameter attributes are defined. The init method of this base class creates properties (named using UVParameter.name) from all the UVParameter attributes on the subclass. AngleParameter and LocationParameter attributes also have extra convenience properties defined:

AngleParameter:

UVParameter.name+'_degrees'

LocationParameter:

UVParameter.name+'_lat_lon_alt'

UVParameter.name+'_lat_lon_alt_degrees'

__eq__(*other*)
Equal if classes match and required parameters are equal.

__iter__()
Iterator for all UVParameter attributes.

__ne__(*other*)
Not equal.

check(*run_sanity_check=True*)
Check that all required parameters are set reasonably.

Check that required parameters exist and have appropriate shapes. Optionally check if the values are sane.

Parameters **run_sanity_check** – Option to check if values in required parameters are sane. Default is True.

degree_prop_fget(*param_name*)
Degree getter method for AngleParameter properties.

degree_prop_fset(*param_name*)
Degree setter method for AngleParameter properties.

extra()
Iterator for all non-required UVParameter attributes.

lat_lon_alt_degrees_prop_fget(*param_name*)
Lat/lon/alt degree getter method for LocationParameter properties.

lat_lon_alt_degrees_prop_fset(*param_name*)
Lat/lon/alt degree setter method for LocationParameter properties.

lat_lon_alt_prop_fget(*param_name*)
Lat/lon/alt getter method for LocationParameter properties.

lat_lon_alt_prop_fset(*param_name*)
Lat/lon/alt setter method for LocationParameter properties.

prop_fget(*param_name*)
Getter method for UVParameter properties.

prop_fset(*param_name*)
Setter method for UVParameter properties.

required()
Iterator for all required UVParameter attributes.

class uvfits.**UVFITS**
Defines a uvfits-specific subclass of UVData for reading and writing uvfits files. This class should not be interacted with directly, instead use the read_uvfits and write_uvfits methods on the UVData class.

uvfits_required_extra
Names of optional UVParameters that are required for uvfits.

read_uvfits(*filename*, *run_check=True*, *run_sanity_check=True*)
Read in data from a uvfits file.

Parameters

- **filename** – The uvfits file to read from.

- **run_check** – Option to check for the existence and proper shapes of required parameters after reading in the file. Default is True.
- **run_sanity_check** – Option to sanity check the values of required parameters after reading in the file. Default is True.

write_uvfits (*filename*, *spoof_nonessential=False*, *force_phase=False*, *run_check=True*, *run_sanity_check=True*)
Write the data to a uvfits file.

Parameters

- **filename** – The uvfits file to write to.
- **spoof_nonessential** – Option to spoof the values of optional UVParameters that are not set but are required for uvfits files. Default is False.
- **force_phase** – Option to automatically phase drift scan data to zenith of the first timestamp. Default is False.
- **run_check** – Option to check for the existence and proper shapes of required parameters before writing the file. Default is True.
- **run_sanity_check** – Option to sanity check the values of required parameters before writing the file. Default is True.

class miriad.**Miriad**
Defines a Miriad-speficic subclass of UVData for reading and writing Miriad files. This class should not be interacted with directly, instead use the read_miriad and write_miriad methods on the UVData class.

read_miriad (*filepath*, *correct_lat_lon=True*, *run_check=True*, *run_sanity_check=True*)
Read in data from a miriad file.

Parameters

- **filepath** – The miriad file directory to read from.
- **correct_lat_lon** – flag – that only matters if altitude is missing – to update the latitude and longitude from the known_telescopes list
- **run_check** – Option to check for the existence and proper shapes of required parameters after reading in the file. Default is True.
- **run_sanity_check** – Option to sanity check the values of required parameters after reading in the file. Default is True.

write_miriad (*filepath*, *run_check=True*, *run_sanity_check=True*, *clobber=False*)
Write the data to a miriad file.

Parameters

- **filename** – The miriad file directory to write to.
- **run_check** – Option to check for the existence and proper shapes of required parameters before writing the file. Default is True.
- **run_sanity_check** – Option to sanity check the values of required parameters before writing the file. Default is True.
- **clobber** – Option to overwrite the filename if the file already exists. Default is False.

class fhd.**FHD**
Defines a FHD-specific subclass of UVData for reading FHD save files. This class should not be interacted with directly, instead use the read_fhd method on the UVData class.

read_fhd(*filelist*, *use_model=False*, *run_check=True*, *run_sanity_check=True*)
Read in data from a list of FHD files.

Parameters

- **filelist** – The list of FHD save files to read from. Must include at least one polarization file, a params file and a flag file.
- **use_model** – Option to read in the model visibilities rather than the dirty visibilities. Default is False.
- **run_check** – Option to check for the existence and proper shapes of required parameters after reading in the file. Default is True.
- **run_sanity_check** – Option to sanity check the values of required parameters after reading in the file. Default is True.

class telescopes.**Telescope**
A class for defining a telescope for use with UVData objects.

citation
str

text giving source of telescope information

telescope_name
string, UVParameter

name of the telescope

telescope_location
array_like, UVParameter

telescope location xyz coordinates in ITRF (earth-centered frame).

6.4.2 Functions

telescopes.**known_telescopes**()
Get list of known telescopes.

telescopes.**get_telescope**(*telescope_name*)
Get Telescope object for a telescope in known_telescopes().

Parameters **telescope_name** – string name of a telescope, must be in known_telescopes().

Returns The Telescope object associated with telescope_name.

6.4.3 Utility Functions

Commonly used utility functions.

utils.**LatLonAlt_from_XYZ**(*xyz*)
Calculate lat/lon/alt from topocentric x,y,z.

Parameters **xyz** – numpy array, shape (3,), with topocentric x,y,z coordinates

Returns tuple of latitude, longitude, altitude values in radians & meters

utils.**XYZ_from_LatLonAlt**(*latitude*, *longitude*, *altitude*)
Calculate topocentric x,y,z from lat/lon/alt values.

Parameters

- **latitude** – latitude in radians
- **longitude** – longitude in radians
- **altitude** – altitude in meters

Returns numpy array, shape (3,), with topocentric x,y,z coordinates

u

Symbols

A

B

C

D

E

F

G

J

K

L

M

N

P

R

S

T

U

V

W

X