py FITS IDI Documentation

Release 0.1

Danny Price

May 11, 2011

CONTENTS

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

CHAPTER

ONE

ABOUT PYFITSIDI

pyFitsidi is a python module that creates FITS IDI files.

pyFitsidi is a collection of functions that create headers and data units that conform to the FITS-IDI convention. It was written primarily to convert data from a CASPER correlator into a format that can be imported into data reduction packages such as AIPS++ and CASA.

Figure 1.1: The Northern Cross, Medicina, Italy. Photo by G. Foster.

CHAPTER

TWO

THE FITS IDI CONVENTION

The FITS Interferometry Data Interchange Convention ("FITS-IDI") is a set of conventions layered upon the standard FITS format to assist in the interchange of data recorded by interferometric telescopes, particularly at radio frequencies and very long baselines.

FITS IDI is a registered convention in the NASA/IAU FITS working group registry, meaning it is defined and documented to "a minimum level of completeness and clarity". FITS IDI files can be read by AIPS, AIPS++ and CASA (although I have only tested with CASA). They are used by the VLBA (Very Long Baseline Array) and JIVE (Joint Institute for VLBI in Europe), among others.

There are a few alternative formats to FITS IDI, such as Miriad FITS files, UVFITS files, and Measurement Sets (MS). Miriad is getting a little long in the tooth, UVFITS is not very well documented, and Measurement Sets are a bit of a pain to create – plus, a MS isn't a FITS file, so it can't be read by FITS readers.

Much more info on the FITS IDI convention can be found on its official page. If you're not familiar with FITS files, you might also want to look at the actual FITS definition. Finally, there are useful FITS resources at the NASA FITS website.

CHAPTER

THREE

PREREQUISITES

The actual reading/writing of FITS files is done by the PyFITS package. You'll also need numpy for array handling and lxml for parsing config XML files.

In the file createMedicinaFITS.py, we make use of pyEphem for astronomical calculations, and pyTables for HDF5 file I/O. Yo

CHAPTER

FOUR

EXAMPLE USAGE

A simple example of how to use pyFitsidi.py can be found in the createFitsIDI.py:

```
>>> """
... createFitsIDI.py
... =============
...
... Creates a basic FITS IDI file, with headers created from an XML configuration file.
...
... Created by Danny Price on 2011-04-20.
... Copyright (c) 2011 The University of Oxford. All rights reserved.
...
... """
...
... # Required modules
... import sys, os
... import pyfits as pf, numpy as np
...
... # import modules from this package
... from pyFitsidi import *
...
... def main():
...     """
...     Generate a blank FITS IDI file.
...     """
...
...     # What are the filenames for our datasets?
...     fitsfile = 'blank.fits'
...     config   = 'config/config.xml'
...
...     # Make a new blank FITS HDU
...     print('Creating Primary HDU')
...     print('-----------------------------------\n')
...     hdu = make_primary(config)
...     print hdu.header.ascardlist()
...
...     # Go through and generate required tables
...     print('\nCreating ARRAY_GEOMETRY')
...     print('-----------------------------------')
...     tbl_array_geometry = make_array_geometry(config=config, num_rows=32)
...     print tbl_array_geometry.header.ascardlist()
...
...     print('\nCreating FREQUENCY')
...     print('-----------------------------------')
...     tbl_frequency = make_frequency(config=config, num_rows=1)
```

```
...     print tbl_frequency.header.ascardlist()
...
...     print('\nCreating SOURCE')
...     print('------------------------------------')
...     tbl_source = make_source(config=config, num_rows=1)
...     print tbl_source.header.ascardlist()
...
...     print('\nCreating ANTENNA')
...     print('------------------------------------')
...     tbl_antenna = make_antenna(config=config, num_rows=32)
...     print tbl_antenna.header.ascardlist()
...
...     print('\nCreating UV_DATA')
...     print('------------------------------------')
...     # Number of rows req. depends on num. time dumps and num. of baselines
...     # Once you have data, it would be worth setting these dimensions automatically
...     # For now, we're hard-wiring in 1 time dump, 528 baselines (32 element array)
...     (t_len, bl_len) = (1,528)
...     tbl_uv_data = make_uv_data(config=config, num_rows=t_len*bl_len)
...     print tbl_antenna.header.ascardlist()
...
...     print('\nCreating HDU list')
...     print('------------------------------------')
...     hdulist = pf.HDUList(
...                 [hdu,
...                 tbl_array_geometry,
...                 tbl_frequency,
...                 tbl_antenna,
...                 tbl_source,
...                 tbl_uv_data
...                 ])
...
...     print('\nVerifying integrity...')
...     hdulist.verify()
...
...     if(os.path.isfile(fitsfile)):
...       print('Removing existing file %s...')%fitsfile
...       os.remove(fitsfile)
...     print('Writing to file %s...')%fitsfile
...     hdulist.writeto(fitsfile)
...
...     print('Done.')
...
...
... if __name__ == '__main__':
...     main()
```

A more complicated example is shown in createMedicinaFITS.py. In this file, a complete FITS IDI file is generated from an XML config file and a HDF5 data file. While the XML and HDF5 data are specific to the Medicina BEST-2 telescope, this file could be adapted as required.

In createMedicinaFITS.py, only the mandatory binary tables are created: array_geometry, antenna, frequency, source and uv_data.

CHAPTER

FIVE

MODULE DOCUMENTATION

5.1 pyFitsidi.py

Created by Danny Price on 2011-04-28. Copyright (c) 2011 The University of Oxford. All rights reserved.

This file is a collection of modules for creating blank FITS IDI files. It consists mainly of pretty basic functions to create blank tables.

The FITS handling is all done by pyFits, and blank arrays are created with numpy. So you'll need to have both of these installed on your machine. In addition, the config xml file is read with lxml, so install that too.

5.1.1 FITS-IDI Table Overview

Here is a quick rundown of the tables that this script can make. Not all these tables are mandatory, so I've starred the ones which are the most important to understand and to get into your dataset.

- **ANTENNA** Antenna polarization information
- **ARRAY_GEOMETRY** Time system information and antenna coordinates
- FLAG Flagged data
- **FREQUENCY** Frequency setups
- GAIN_CURVE Antenna gain curves
- INTERFEROMETER_MODEL Correlator model
- PHASE-CAL Phase cal measurements
- **SOURCE** Sources observed
- SYSTEM_TEMPERATURE System and antenna temperatures
- **UV_DATA** Visibility data

These tables are new from the FITS IDI update, and are all optional:

- BANDPASS Bandpass functions
- BASELINE Baseline-specific gain factors
- CALIBRATION Gains as a function of time
- WEATHER Meteorological data

5.1.2 Module listing

pyFitsidi.**make_antenna**(*config='config.xml'*, *num_rows=1*)
Creates a vanilla ANTENNA table HDU

Parameters **config: string** :

filename of xml configuration file, defaults to 'config,xml'

num_rows: int :

number of rows to generate. Rows will be filled with numpy zeros.

Notes

CASA didn't like importing POLCALA and POLCALB, so they are currently commented out.

Table is built with the following columns:

•TIME: Difference of antenna table time interval centre time and RDATE 0 hours

•TIME_INTERVAL: Antenna table time interval width

•ANNAME: Antenna name, should match value in ARRAY_GEOMETRY

•ANTENNA_NO: Antemma ID number for station

•ARRAY: Array ID number

•FREQID: Frequency setup ID number

•NO_LEVELS:Number of digitiser levels

•POLYTYA: Feed A polarisation direction

•POLAA: Feed A polarisation (degrees)

•POLCALA: Feed A polarisation parameters

•POLYTYB: As above, for feed B

•POLAB: As above, for feed B

•POLCALB: As above, for feed B

pyFitsidi.**make_array_geometry**(*config='config.xml'*, *num_rows=1*)
Creates a vanilla ARRAY_GEOMETRY table HDU.

One row is required for each antenna in the array (num_rows)

Parameters **config: string** :

filename of xml configuration file, defaults to 'config,xml'

num_rows: int :

number of rows to generate. Rows will be filled with numpy zeros.

Notes

Table is built with the following columns:

•ANNAME: Antenna name

•STABXYZ: Antenna relative position vector ECI components, in meters

•DERXYZ: Antenna velocity vector components, in meters/sec

•ORBPARM: Orbital parameters

•NOSTA: Antenna ID number for station

•MNTSTA: Antenna mount type (0 is alt-azimuth)

•STAXOF: Antenna axis offset, in meters

•DIAMETER: Antenna diameter (optional)

pyFitsidi.**make_bandpass**(*config='config.xml'*, *num_rows=1*)

Creates a vanilla BANDPASS table HDU

Parameters **config: string** :

filename of xml configuration file, defaults to 'config,xml'

num_rows: int :

number of rows to generate. Rows will be filled with numpy zeros.

Notes

Table is built with the following columns:

•TIME: Difference of bandpass table time int center time and RDATE 0 hours

•TIME_INTERVAL: Bandpass table time interval width

•SOURCE_ID: Source ID number

•ANTENNA_NO: Antenna ID number for station

•ARRAY: Array ID number

•FREQID: Frequency setup ID number (should match ANTENNA)

•BANDWIDTH: Frequency band width described by bandpass

•BAND_FREQ: Frequency band base offset

•REFANT_1: Reference antenna ID number

•BREAL_1: Bandpass response real componet

•BIMAG_1: Bandpass response imaginary component

pyFitsidi.**make_flag**(*config='config.xml'*, *num_rows=1*)

Creates a vanilla FLAG table HDU

Parameters **config: string** :

filename of xml configuration file, defaults to 'config,xml'

num_rows: int :

number of rows to generate. Rows will be filled with numpy zeros.

Notes

This table is optional.

Table is built with the following columns:

- SOURCE_ID Source ID number
- ARRAY Array number
- ANTS Antenna numbers
- FREQID Frequency setup number
- TIMERANG Time range
- BANDS Band flags
- CHANS Channel range
- PFLAGS Polarization flags
- REASON Reason for flag
- SEVERITY Severity code

pyFitsidi.**make_frequency**(*config='config.xml'*, *num_rows=1*)

Creates a vanilla FREQUENCY table HDU

Parameters **config: string** :

filename of xml configuration file, defaults to 'config,xml'

num_rows: int :

number of rows to generate. Rows will be filled with numpy zeros.

Notes

Table is built with the following columns:

- FREQID: Frequency setup ID number
- BANDFREQ: Frequency band base offset (Hz)
- CH_WIDTH: Frequency channel width (Hz)
- TOTAL_BANDWIDTH: Frequency bandwidth (Hz)
- SIDEBAND: Sideband flag (1 indicates upper sideband)
- BB_CHAN: ?

pyFitsidi.**make_gain_curve**(*config='config.xml'*, *num_rows=1*)

Creates a vanilla GAIN_CURVE table HDU

Parameters **config: string** :

filename of xml configuration file, defaults to 'config,xml'

num_rows: int :

number of rows to generate. Rows will be filled with numpy zeros.

Notes

This is an optional table (we will not include it). Todo: Add switch to allow dual polarisation.

Table is built with the following columns:

- ANTENNA_NO Antenna number

- •ARRAY Array number
- •FREQID Frequency setup number
- •TYPE_1 Gain curve types for polarization 1
- •NTERM_1 Numbers of terms or entries for polarization 1
- •X_TYP_1 x value types for polarization 1
- •Y_TYP_1 y value types for polarization 1
- •X_VAL_1 x values for polarization 1
- •Y_VAL_1 y values for polarization 1
- •GAIN_1 Relative gain values for polarization 1
- •SENS_1 Sensitivities for polarization 1

pyFitsidi.**make_interferometer_model**(*config='config.xml'*, *num_rows=1*)

Creates a vanilla INTERFEROMETER_MODEL table HDU.

Parameters **config: string** :

filename of xml configuration file, defaults to 'config,xml'

num_rows: int :

number of rows to generate. Rows will be filled with numpy zeros.

Notes

This table is optional, and is not included in Medicina FITS IDI

Table is built with the following columns:

- •TIME Starting time of interval
- •TIME_INTERVAL Duration of interval
- •SOURCE_ID Source ID number
- •ANTENNA_NO Antenna number
- •ARRAY Array number
- •FREQID Frequency setup number
- •I.FAR.ROT Ionospheric Faraday rotation
- •FREQ_VAR Time-variable frequency offsets
- •PDELAY_1 Phase delay polynomials for polarization 1
- •GDELAY_1 Group delay polynomials for polarization 1
- •PRATE_1 Phase delay rate polynomials for polarization 1
- •GRATE_1 Group rate polynomials for polarization 1
- •DISP_1 Dispersive delay for polarization 1 at 1m wavelength
- •DDISP_1 Rate of change of dispersive del for pol 1 at 1m

pyFitsidi.**make_phase_cal**(*config='config.xml'*, *num_rows=1*)

Creates a vanilla PHASE-CAL table HDU

Parameters **config: string** :

filename of xml configuration file, defaults to 'config,xml'

num_rows: int :

number of rows to generate. Rows will be filled with numpy zeros.

Notes

This is an optional table (we will not include it)

Table is built with the following columns:

- TIME Central time of interval covered
- TIME_INTERVAL Duration of interval
- SOURCE_ID Source ID number
- ANTENNA_NO Antenna number
- ARRAY Array number
- FREQID Frequency setup number
- CABLE_CAL Cable calibration measurement
- STATE_1 State counts for polarization 1
- PC_FREQ_1 Phase cal tone frequencies for polarization 1
- PC_REAL_1 real parts of phase-cal measurements for pol 1
- PC_IMAG_1 imaginary parts of phasecal measurements for pol 1
- PC_RATE_1 phase-cal rates for polarization 1

pyFitsidi.**make_primary**(*config='config.xml'*)

Creates the primary header data unit (HDU).

This function generates header keywords from the file headers/primary.tpl

Parameters **config: string** :

filename of xml configuration file, defaults to 'config,xml'

pyFitsidi.**make_source**(*config='config.xml'*, *num_rows=1*)

Creates a vanilla SOURCE table HDU

Parameters **config: string** :

filename of xml configuration file, defaults to 'config,xml'

num_rows: int :

number of rows to generate. Rows will be filled with numpy zeros.

Notes

Table is built with the following columns:

- SOURCE_ID Source ID number
- SOURCE Source name

- •QUAL Source qualifier number.
- •CALCODE Source calibrator code.
- •FREQID Source frequency ID
- •IFLUX Source I flux density
- •QFLUX Source Q flux density
- •UFLUX Source U flux density
- •VFLUX SourceV flux density
- •ALPHA Source spectral index
- •FREQOFF Source frequency offset
- •RAEPO Source J2000 equatorial position RA coordinate
- •DECPO Source J2000 equatorial position DEC coordinate
- •EQUINOX Mean Equinox
- •RAAPP Source apparent equatorial position RA coordinate
- •DECAPP Source apparent equatorial position DEC coordinate
- •SYSVEL Systematic velocity.
- •VELTYP Systematic velocity reference frame.
- •VELDEF Systematic velocity convention.
- •RESTFREQ Line rest frequency.
- •PMRA Source proper motion RA coordinate
- •PMDEC Source proper motion DEC coordinate
- •PARALLAX Source parallax.

pyFitsidi.**make_system_temperature**(*config='config.xml'*, *num_rows=1*)

Creates a vanilla SYSTEM_TEMPERATURE table HDU

Parameters **config: string** :

filename of xml configuration file, defaults to 'config,xml'

num_rows: int :

number of rows to generate. Rows will be filled with numpy zeros.

Notes

This is an optional table. Todo: add dual pol support.

Table is built with the following columns:

- •TIME Central time of interval covered
- •TIME_INTERVAL Duration of interval
- •SOURCE_ID Source ID number
- •ANTENNA_NO Antenna number
- •ARRAY Array number

- FREQID Frequency setup number
- TSYS_1 System temperatures for polarization 1
- TANT_1 Antenna temperatures for polarization 1

pyFitsidi.**make_uv_data**(*config='config.xml'*, *num_rows=1*)

Creates a vanilla UV_DATA table HDU

Parameters **config: string** :

filename of xml configuration file, defaults to 'config,xml'

num_rows: int :

number of rows to generate. Rows will be filled with numpy zeros.

Notes

Table is built with the following columns:

- UU Baseline vector U coordinate (seconds)
- VV Baseline vector V coordinate
- WW Baseline vector W coordinate
- DATE UTC Julian day value for time 00:00:00 on the day of the observation
- TIME Fraction of Julian day from midnight to timestamp on day of observation.
- BASELINE Antenna baseline pair ID.
- FILTER VLBA filter ID
- SOURCE Data source ID
- FREQID Data frequency setup ID
- INTTIM Data integration time
- WEIGHT Data weights (one element for each freq channel)
- GATEID VLBA gate ID
- FLUX UV visibility data matrix

pyFitsidi.**parseConfig**(*tagname*, *config='config.xml'*)

Finds tagname, in elementTree x, parses and returns dictionary of values This is a helper function, and is not usually called directly

5.2 astroCoords.py

Some useful functions for converting in between Right Ascension (hours,mins,secs) and Declination (degrees,mins,secs) to degrees and radians. These functions are all named in the same format: a2b(), where a is the original data, and b is the new data format.

All of this type of basic stuff can be done with pyEphem or similar, but I wanted something super simple.

5.2.1 Module listing

astroCoords.**decdms2deg**(*deg*, *min*, *sec*)
Converts degrees, minutes seconds into degrees i.e. a floating point number in range (0,360)

Parameters **deg: int** :

degrees to be converted (from 0 to 360)

min: int :

minutes to be converted (from 0 to 60)

sec: float :

seconds to be converted (from 0.0 to 60.0)

astroCoords.**decdms2rad**(*deg*, *min*, *sec*)
Converts degrees, minutes seconds into radians i.e. a floating point number in range (0,2pi)

Parameters **deg: int** :

degrees to be converted (from 0 to 360)

min: int :

minutes to be converted (from 0 to 60)

sec: float :

seconds to be converted (from 0.0 to 60.0)

astroCoords.**deg2decdms**(*degrees*)
Convert number in degrees to declination degs, arcmins, arcsecs Returns in list (hours,mins,secs)

Parameters **degrees: float** :

degrees to be converted (from 0 to 360)

astroCoords.**deg2rad**(*degrees*)
Converts from degrees to radians. This is just a numpy call...

Parameters **degrees: float** :

degrees to be converted (from 0 to 360)

astroCoords.**deg2rahms**(*degrees*)
Convert number in degrees to right ascension HH,MM,SS.SS Returns in list (hours,mins,secs)

Parameters **degrees: float** :

degrees to be converted (from 0 to 360)

astroCoords.**rad2decdms**(*radians*)
Convert number in radians to declination degs, arcmins, arcsecs Returns in list (hours,mins,secs)

Parameters **radians: float** :

radians to be converted (from 0 to 2 pi)

astroCoords.**rad2deg**(*radians*)
Converts from radians to degrees. This is just a numpy call...

Parameters **radians: float** :

radians to be converted (from 0 to 2 pi)

astroCoords.**rad2rahms**(*radians*)
Convert number in radians to right ascension H,M,SS.SS Returns tuple (hours,mins,secs)

Parameters **radians: float** :

radians to be converted (from 0 to 2 pi)

astroCoords.**rahms2deg**(*h*, *m*, *s*)
Converts list (hours,minutes,seconds) to degrees Value returned in range (0,360)

Parameters **h: int** :

hour angle to be converted (from 0 to 24)

h: int :

minutes to be converted (from 0 to 60)

s: float :

seconds to be converted (from 0.0 to 60.0)

astroCoords.**rahms2rad**(*h*, *m*, *s*)
Converts list (hours,minutes,seconds) to radians Value returned in range (0,2pi)

Parameters **h: int** :

hour angle to be converted (from 0 to 24)

h: int :

minutes to be converted (from 0 to 60)

s: float :

seconds to be converted (from 0.0 to 60.0)

PYTHON MODULE INDEX

INDEX