Pulsar Data Toolbox Documentation

Release 0.1.2

Jeffrey S. Hazboun

Oct 14, 2020

Contents

Contents:

CHAPTER 1

PulsarDataToolbox

Python package for dealing with PSRFITS files.

- Free software: MIT License
- Documentation: https://PulsarDataToolbox.readthedocs.io.

1.1 Features

The Pulsar Data Toolbox has tools built for building pulsar data formats. Only the *PSRFITS* standard is currently supported, but if you are interested in another format submit an issue or a pull request!

1.2 Credits

This package was created with Cookiecutter and the audreyr/cookiecutter-pypackage project template.

CHAPTER 2

Installation

2.1 Stable release

To install PulsarDataToolbox, run this command in your terminal:

```
$ pip install pdat
```

This is the preferred method to install PulsarDataToolbox, as it will always install the most recent stable release.

If you don't have pip installed, this Python installation guide can guide you through the process.

2.2 From sources

The sources for PulsarDataToolbox can be downloaded from the Github repo.

You can either clone the public repository:

```
$ git clone git://github.com/Hazboun6/PulsarDataToolbox
```

Or download the tarball:

```
$ curl  -OL https://github.com/hazboun6/PulsarDataToolbox/tarball/master
```

Once you have a copy of the source, you can install it with:

```
$ python setup.py install
```

CHAPTER 3

Usage

To use PulsarDataToolbox in a project:

```
import pdat

fits_path = 'path/to/psrfits/file'
my_psrfits = pdat.psrfits(fits_path)
```

Note: This tutorial was generated from a Jupyter notebook that can be downloaded here.

CHAPTER 4

Pulsar Data Toolbox:

4.1 `psrfits` class

The `psrfits` class allows easy access to the specialized FITS files used in the Pulsar/Radio Astronomy community know as PSRFITS files. The standard can be found on the CSIRO Pulsar Group website. In the current version of `pdat` this class is based on the Python package `fitsio` which is a wrapper for the c-library `cfitsio`. In the future we plan to also make a version that uses the `astropy.io.fits` package, however the `c` library is fast, efficient, allows appending and accessing of BinTables without loading the whole file to memory. Since PSRFITS files carry large BinTables these types of efficiencies are very useful.

4.2 Loading and Appending

```
import pdat
import os
```

```
pFits1 = '../../../templates/search_scratch.fits'
pFits2 = '../../../templates/search_template.fits'
```

4.3 Check file sizes

```
a=os.path.getsize(pFits1)
b=os.path.getsize(pFits2)
print('Size of 1st file:',a)
print('Size of 2nd file:',b)
```

```
Size of 1st file: 5302080
Size of 2nd file: 5302080
```

4.4 Load files

```
psrf1 = pdat.psrfits(pFits1)
```

```
Loading PSRFITS file from path:
'../../../templates/search_scratch.fits'.
```

4.5 Append the Secondary BinTables to an existing PSRFITS

The `append_from_file` method appends all of the secondary BinTables of a PSRFITS, given as a file path, to the already loaded PSRFITS. The secondary BinTables include `SUBINT`,`POLYCO`, `HISTORY` and `PARAM`. This is only possible between identical `mode` files (`SEARCH`, `PSR` or `CAL`). By default the order of the tables is assumed identical. If the BinTables are in different orders there is an optional `table` flag to provide a list of the order of the original BinTables. Alternatively, you may only select a subset of BinTables to append.

```
psrf1.append_from_file(pFits2)
```

```
os.path.getsize(pFits1)
```

```
5302080
```

Checking the size we see it has grown, but not doubled. That is because the `PRIMARY` header was not changed.

The `psrfits` class comes with all of the functionality built into `fitsio`. The class represents a list of HDUs. The header information is accesible through the `read_header` method.

```
psrf1[1].read_header()
```

```
XTENSION= 'BINTABLE'           / * Subintegration data  *
BITPIX  =                    8 / N/A
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =               264268 / width of table in bytes
NAXIS2  =                   20 / Number of rows in table (NSUBINT)
PCOUNT  =                    0 / size of special data area
GCOUNT  =                    1 / one data group (required keyword)
TFIELDS =                   17 / Number of fields per row
TTYPE1  = 'TSUBINT '           / Length of subintegration
TFORM1  = '1D      '           / Double
TTYPE2  = 'OFFS_SUB'           / Offset from Start of subint centre
TFORM2  = '1D      '           / Double
TTYPE3  = 'LST_SUB '           / LST at subint centre
TFORM3  = '1D      '           / Double
TTYPE4  = 'RA_SUB  '           / RA (J2000) at subint centre
TFORM4  = '1D      '           / Double
TTYPE5  = 'DEC_SUB '           / Dec (J2000) at subint centre
TFORM5  = '1D      '           / Double
TTYPE6  = 'GLON_SUB'           / [deg] Gal longitude at subint centre
TFORM6  = '1D      '           / Double
TTYPE7  = 'GLAT_SUB'           / [deg] Gal latitude at subint centre
TFORM7  = '1D      '           / Double
TTYPE8  = 'FD_ANG  '           / [deg] Feed angle at subint centre
TFORM8  = '1E      '           / Float
```

```
TTYPE9  = 'POS_ANG '           / [deg] Position angle of feed at subint centre
TFORM9  = '1E      '           / Float
TTYPE10 = 'PAR_ANG '           / [deg] Parallactic angle at subint centre
TFORM10 = '1E      '           / Float
TTYPE11 = 'TEL_AZ  '           / [deg] Telescope azimuth at subint centre
TFORM11 = '1E      '           / Float
TTYPE12 = 'TEL_ZEN '           / [deg] Telescope zenith angle at subint centre
TFORM12 = '1E      '           / Float
TTYPE13 = 'DAT_FREQ'           / [MHz] Centre frequency for each channel
TFORM13 = '    128E'           / NCHAN floats
TTYPE14 = 'DAT_WTS '           / Weights for each channel
TFORM14 = '    128E'           / NCHAN floats
TTYPE15 = 'DAT_OFFS'           / Data offset for each channel
TFORM15 = '    128E'           / NCHAN*NPOL floats
TTYPE16 = 'DAT_SCL '           / Data scale factor for each channel
TFORM16 = '    128E'           / NCHAN*NPOL floats
TTYPE17 = 'DATA    '           / Subint data table
TFORM17 = '  262144B'          / NBIN*NCHAN*NPOL*NSBLK int, byte(B) or bit(X)
TDIM17  = '(1, 128, 1, 2048)'  / Dimensions (NBITS or NBIN,NCHAN,NPOL,NSBLK)
INT_TYPE= 'TIME    '           / Time axis (TIME, BINPHSPERI, BINLNGASC, etc)
INT_UNIT= 'SEC     '           / Unit of time axis (SEC, PHS (0-1), DEG)
SCALE   = 'FluxDen '           / Intensity units (FluxDen/RefFlux/Jansky)
NPOL    =                    1 / Nr of polarisations
POL_TYPE= 'IQUV    '           / Polarisation identifier (e.g., AABBCRCI,␣
↪AA+BB)
TBIN    =    2.04833984375E-05 / [s] Time per bin or sample
NBIN    =                    1 / Nr of bins (PSR/CAL mode; else 1)
NBIN_PRD=                    0 / Nr of bins/pulse period (for gated data)
PHS_OFFS=                   0. / Phase offset of bin 0 for gated data
NBITS   =                    8 / Nr of bits/datum (SEARCH mode 'X' data, else␣
↪1)
NSUBOFFS=                    0 / Subint offset (Contiguous SEARCH-mode files)
NCHAN   =                  128 / Number of channels/sub-bands in this file
CHAN_BW =               1.5625 / [MHz] Channel/sub-band width
NCHNOFFS=                    0 / Channel/sub-band offset for split files
NSBLK   =                 2048 / Samples/row (SEARCH mode, else 1)
EXTNAME = 'SUBINT  '           / name of this binary table extension
TUNIT1  = 's       '           / Units of field
TUNIT2  = 's       '           / Units of field
TUNIT3  = 's       '           / Units of field
TUNIT4  = 'deg     '           / Units of field
TUNIT5  = 'deg     '           / Units of field
TUNIT6  = 'deg     '           / Units of field
TUNIT7  = 'deg     '           / Units of field
TUNIT8  = 'deg     '           / Units of field
TUNIT9  = 'deg     '           / Units of field
TUNIT10 = 'deg     '           / Units of field
TUNIT11 = 'deg     '           / Units of field
TUNIT12 = 'deg     '           / Units of field
TUNIT13 = 'MHz     '           / Units of field
TUNIT17 = 'Jy      '           / Units of subint data
EXTVER  =                    1 / auto assigned by template parser
```

The data in a `PSRFITS` is found in the `SUBINT` BinTable.

```
psrf1
```

```
file: ../../../templates/search_scratch.fits
mode: READWRITE
extnum hdutype         hduname[v]
0      IMAGE_HDU
1      BINARY_TBL      SUBINT[1]
```

Here SUBINT is the 2nd HDU. The data is accesible as a numpy.recarray with NSUBINT rows. Think of a recarray as a spreadsheet where the individual entries can be strings, floats or whole arrays.

```
data=psrf1[1].read()
print(data.shape)
data.dtype.descr
```

```
(20,)
```

```
[('TSUBINT', '>f8'),
 ('OFFS_SUB', '>f8'),
 ('LST_SUB', '>f8'),
 ('RA_SUB', '>f8'),
 ('DEC_SUB', '>f8'),
 ('GLON_SUB', '>f8'),
 ('GLAT_SUB', '>f8'),
 ('FD_ANG', '>f4'),
 ('POS_ANG', '>f4'),
 ('PAR_ANG', '>f4'),
 ('TEL_AZ', '>f4'),
 ('TEL_ZEN', '>f4'),
 ('DAT_FREQ', '>f4', (128,)),
 ('DAT_WTS', '>f4', (128,)),
 ('DAT_OFFS', '>f4', (128,)),
 ('DAT_SCL', '>f4', (128,)),
 ('DATA', '|u1', (2048, 1, 128, 1))]
```

While the DATA array above is 4 dimensional (this is the case in SEARCH files, it is 3 dimensional in PSR and CAL files). However there are NSUBINT of those arrays. To access the data one uses the name of the column, DATA, then a single entry square bracket denoting the row. This gives one of the NSUBINT arrays in the BinTable.

```
data['DATA'][0].shape
```

```
(2048, 1, 128, 1)
```

This object is then a normal numpy array that can be accessed with numpy array slice notation. Access a single entry by choosing four integers in the range of dimensions.

```
data['DATA'][0][1000,0,3,0]
```

```
7
```

Other arrays are accessed similarly, but without as many indices. There are NSUBINT rows of 1-dimensional arrays for each of the DAT_X parameters and NSUBINT floats of the other entries.

```
print(data['DAT_OFFS'].shape)
data['DAT_OFFS'][2]
```

```
(20, 128)
```

```
array([ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,
        0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.], dtype=float32)
```

```
print(data['GLON_SUB'].shape)
data['GLON_SUB'][2]
```

```
(20,)
```

```
97.721010667684681
```

One can clear the file from memory using the `close` method.

```
psrf1.close()
```

4.6 `PSR` and `CAL` files

The PSRFITS standard actually has many BinTable extensions, and many files come with more than two HDUs. The `psrfits` class will generically build a Python version of any of these file types. In this package there are three template types, corresponding to the three most common file types used by the NANOGrav Pulsar Timing array. If you would like another template included please start an issue on our GitHub page.

A `PSR` mode file is data from an observation where the data is folded at the frequency of the pulsar to build up signal-to-noise ratio in real time. A `CAL` file has the same set of HDUs but is not folded. It is data take of a calibration source. Here we access the `PSR` template file and look at a different BinTable extension.

```
pFits3 = '../../../templates/psr_template.fits'
psrf2 = pdat.psrfits(pFits3)
```

```
Loading PSRFITS file from path:
'/Users/jeffrey/PSS/guppi_57691_J1909-3744_0004_0001.fits'.
```

```
psrf2
```

```
file: /Users/jeffrey/PSS/guppi_57691_J1909-3744_0004_0001.fits
mode: READWRITE
extnum hdutype         hduname[v]
0      IMAGE_HDU
1      BINARY_TBL      HISTORY[1]
2      BINARY_TBL      PSRPARAM[1]
3      BINARY_TBL      POLYCO[1]
4      BINARY_TBL      SUBINT[1]
```

```
psrf2[3].read_header()
```

```
XTENSION= 'BINTABLE'           / * Polyco history *
BITPIX  =                    8 / N/A
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                  222 / width of table in bytes
NAXIS2  =                    1 / number of rows in table
PCOUNT  =                    0 / size of special data area
GCOUNT  =                    1 / one data group (required keyword)
TFIELDS =                   13 / Number of fields per row
TTYPE1  = 'DATE_PRO'           / Polyco creation date and time (UTC)
TFORM1  = '24A     '           / 24-char string
TTYPE2  = 'POLYVER '           / Polyco version ID
TFORM2  = '16A     '           / 16-char string
TTYPE3  = 'NSPAN   '           / Span of polyco block in min
TFORM3  = '1I      '           / Integer
TTYPE4  = 'NCOEF   '           / Nr of coefficients (<=15)
TFORM4  = '1I      '           / Integer
TTYPE5  = 'NPBLK   '           / Nr of blocks (rows) for this polyco
TFORM5  = '1I      '           / Integer
TTYPE6  = 'NSITE   '           / Observatory code
TFORM6  = '8A      '           / 8-char string
TTYPE7  = 'REF_FREQ'           / Reference frequency for phase
TFORM7  = '1D      '           / Double
TTYPE8  = 'PRED_PHS'           / Predicted pulse phase at observation start
TFORM8  = '1D      '           / Double
TTYPE9  = 'REF_MJD '           / Reference MJD
TFORM9  = '1D      '           / Double
TTYPE10 = 'REF_PHS '           / Reference phase
TFORM10 = '1D      '           / Double
TTYPE11 = 'REF_F0  '           / Zero-order pulsar frequency
TFORM11 = '1D      '           / Double
TTYPE12 = 'LGFITERR'           / Log_10 of polynomial fit rms error in periods
TFORM12 = '1D      '           / Double
TTYPE13 = 'COEFF   '           / Polyco coefficients
TFORM13 = '15D     '           / NCOEF doubles
EXTNAME = 'POLYCO  '           / name of this binary table extension
TUNIT7  = 'MHz     '           / Units of field
TUNIT11 = 'Hz      '           / Units of field
EXTVER  =                    1 / auto assigned by template parser
```

```
psrf2[3]['COEFF'][:]
```

```
array([[  6.37061369e-07,  -3.84007940e-01,   1.63071384e-03,
         -1.91944367e-06,   1.07255013e-09,   6.72218368e-12,
         -8.60574070e-12,   1.25507648e-13,   1.71341258e-14,
         -2.97308173e-16,  -1.79229301e-17,   2.50414099e-19,
          9.50130849e-21,  -7.26854989e-23,  -2.02121757e-24]])
```

```
psrf2[2]['PARAM'][:]
```

```
array([ b'PSRJ              1909-3744
↪                                                                    ',
```

(continues on next page)

(continued from previous page)

```
      b'RAJ                19:09:47.4380095699897                                 ␣
↪                                                                                ',
      b'DECJ              -37:44:14.3162347000103                                 ␣
↪                                                                                ',
      b'PEPOCH             53000.0000000000000000                                 ␣
↪                                                                                ',
      b'F                  3.3931569275871846D+02                                 ␣
↪                                                                                ',
      b'F1                -1.6150815823660001D+00                                 ␣
↪                                                                                ',
      b'PMDEC             -3.6776299999999999D+01                                 ␣
↪                                                                                ',
      b'PMRA              -9.5500000000000007D+00                                 ␣
↪                                                                                ',
      b'POSEPOCH           53000.0000000000000000                                 ␣
↪                                                                                ',
      b'PX                 1.3517999999999999D+00                                 ␣
↪                                                                                ',
      b'DM                 1.0394679999999999D+01                                 ␣
↪                                                                                ',
      b'START              53219.2149999999965075                                 ␣
↪                                                                                ',
      b'FINISH             54614.2710000000006403                                 ␣
↪                                                                                ',
      b'CLK                UTC(NIST)                                              ␣
↪                                                                                ',
      b'EPHEM              DE405                                                  ␣
↪                                                                                ',
      b'TZRMJD             53293.02028990324198077                                ␣
↪                                                                                ',
      b'TZRFRQ             8.4256500000000005D+02                                 ␣
↪                                                                                ',
      b'TZRSITE            1                                                      ␣
↪                                                                                ',
      b'BINARY             ELL1                                                   ␣
↪                                                                                ',
      b'A1                 1.8979909859999999D+00                                 ␣
↪                                                                                ',
      b'PB                 1.5334494510779999D+00                                 ␣
↪                                                                                ',
      b'SINI               9.9727800000000000D-01                                 ␣
↪                                                                                ',
      b'M2                 2.2327900000000001D-01                                 ␣
↪                                                                                ',
      b'EPS1               3.7300000000000003D-08                                 ␣
↪                                                                                ',
      b'EPS2               1.1340000000000000D-07                                 ␣
↪                                                                                ',
      b'TASC               53113.9505872139998246                                 ␣
↪                                                                                ',
      b'TRES               4.2999999999999999D-01                                 ␣
↪                                                                                ',
      b'NTOA               746                                                    ␣
↪                                                                                '],
     dtype='|S128')
```

4.7 Glossary:

BinTable: A table of binary data.

HDU: Header Unit. The main division of a FITS file.

ImageHDU: An HDU that either holds a 2-d data array, usually represnting an image, of the primary HDU, acting as the main header file for the FITS file.

SUBINT HDU: The BinTable extension (HDU) that holds the data from a pulsar/radio observation. In a `PSR` (folded) mode PSRFITS file these are actually subintegrations of folded pulsar data.

HISTORY HDU: The BinTable extension (HDU) that has some information about the history of the observation and what may have been done to the data in the file.

FITS Card: The header information in FITS files is held in a FITS card. In Python these are usually held as dictionary-type variables. There is a `card string` which hold the information that appears when you call the header. One of the dictionary entries is the actual value called when accesing the data.

POLYCO HDU: The BinTable extension (HDU) that has a list of the Chebyshev polynomial coefficients used for a short timescale timing model when using the backend of a telescope in 'PSR' (folding) mode.

PARAM HDU: The BinTable extensino (HDU) that hols the parameters of the pulsar. Most often these are text lines taken from a `.par` (parameter) file.

Note: This tutorial was generated from a Jupyter notebook that can be downloaded here.

CHAPTER 5

Writing PSRFITS

```
import numpy as np
import pdat
```

5.1 Instantiate a draft file from a template

```
template = '../../../templates/search_template.fits'
new_psrfits = 'my_psrfits.fits'
```

```
psrfits1=pdat.psrfits(new_psrfits,from_template=template)
```

```
Making new SEARCH mode PSRFITS file using template from path:
'../../../templates/search_template.fits'.
Writing to path 'my_psrfits.fits'.
The Binary Table HDU headers will be written as they are added
     to the PSRFITS file.
```

The `pdat` package is very helpful for making PSRFITS file from drafts. Many of the parameters in the `PRIMARY` and `SUBINT` HDUs are codependent on one another, and the program keeps track of these dependencies for the user. When you instantiate a `psrfits` with a template a set of draft headers is made from the template so that you can edit them before writing to disk. This "template-to-write" scheme exists because many pieces of software that will analyze `PSRFITS` will baulk at files without a complete set of header information.

The template fits file is accessible as `fits_template`.

```
def set_primary_header(psrfits_object,prim_dict):
    """
    prim_dict = dictionary of primary header changes
    """
    PF_obj = psrfits_object
    for key in prim_dict.keys():
```

(continues on next page)

(continued from previous page)

```
        PF_obj.replace_FITS_Record('PRIMARY',key,prim_dict[key])

def set_subint_header(psrfits_object,subint_dict):
    """
    prim_dict = dictionary of primary header changes
    """
    PF_obj = psrfits_object
    for key in subint_dict.keys():
        PF_obj.replace_FITS_Record('SUBINT',key,subint_dict[key])
```

```
def make_subint_BinTable(self):
    subint_draft = self.make_HDU_rec_array(self.nsubint, self.subint_dtype)
    return subint_draft
```

```
psrfits1.fits_template[0].read_header()['OBSERVER']
```

```
'GALILEOGALILEI'
```

The draft headers are editable, and can be changed until you write the file to disk. The ImageHDU that conatins the primary header is names 'PRIMARY'. The others all go by the name of the `EXTNAME`. ['PRIMARY','SUBINT','POLYCO','HISTORY','PARAM']

```
psrfits1.draft_hdrs['SUBINT']
```

```
XTENSION= 'BINTABLE'           / * Subintegration data  *
BITPIX  =                    8 / N/A
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =             33636428 / width of table in bytes
NAXIS2  =                    4 / Number of rows in table (NSUBINT)
PCOUNT  =                    0 / size of special data area
GCOUNT  =                    1 / one data group (required keyword)
TFIELDS =                   17 / Number of fields per row
TTYPE1  = 'TSUBINT '           / Length of subintegration
TFORM1  = '1D      '           / Double
TTYPE2  = 'OFFS_SUB'           / Offset from Start of subint centre
TFORM2  = '1D      '           / Double
TTYPE3  = 'LST_SUB '           / LST at subint centre
TFORM3  = '1D      '           / Double
TTYPE4  = 'RA_SUB  '           / RA (J2000) at subint centre
TFORM4  = '1D      '           / Double
TTYPE5  = 'DEC_SUB '           / Dec (J2000) at subint centre
TFORM5  = '1D      '           / Double
TTYPE6  = 'GLON_SUB'           / [deg] Gal longitude at subint centre
TFORM6  = '1D      '           / Double
TTYPE7  = 'GLAT_SUB'           / [deg] Gal latitude at subint centre
TFORM7  = '1D      '           / Double
TTYPE8  = 'FD_ANG  '           / [deg] Feed angle at subint centre
TFORM8  = '1E      '           / Float
TTYPE9  = 'POS_ANG '           / [deg] Position angle of feed at subint centre
TFORM9  = '1E      '           / Float
TTYPE10 = 'PAR_ANG '           / [deg] Parallactic angle at subint centre
TFORM10 = '1E      '           / Float
TTYPE11 = 'TEL_AZ  '           / [deg] Telescope azimuth at subint centre
```

```
TFORM11 = '1E      '           / Float
TTYPE12 = 'TEL_ZEN '           / [deg] Telescope zenith angle at subint centre
TFORM12 = '1E      '           / Float
TTYPE13 = 'DAT_FREQ'           / [MHz] Centre frequency for each channel
TFORM13 = '2048E   '           / NCHAN floats
TTYPE14 = 'DAT_WTS '           / Weights for each channel
TFORM14 = '2048E   '           / NCHAN floats
TTYPE15 = 'DAT_OFFS'           / Data offset for each channel
TFORM15 = '8192E   '           / NCHAN*NPOL floats
TTYPE16 = 'DAT_SCL '           / Data scale factor for each channel
TFORM16 = '8192E   '           / NCHAN*NPOL floats
TTYPE17 = 'DATA    '           / Subint data table
TFORM17 = '33554432B'          / NBIN*NCHAN*NPOL*NSBLK int, byte(B) or bit(X)
INT_TYPE= 'TIME    '           / Time axis (TIME, BINPHSPERI, BINLNGASC, etc)
INT_UNIT= 'SEC     '           / Unit of time axis (SEC, PHS (0-1), DEG)
SCALE   = 'FluxDen '           / Intensity units (FluxDen/RefFlux/Jansky)
NPOL    =                    4 / Nr of polarisations
POL_TYPE= 'IQUV    '           / Polarisation identifier (e.g., AABBCRCI,␣
↪AA+BB)
TBIN    = 0.000770559999999999 / [s] Time per bin or sample
NBIN    =                    1 / Nr of bins (PSR/CAL mode; else 1)
NBIN_PRD=                    0 / Nr of bins/pulse period (for gated data)
PHS_OFFS=                   0. / Phase offset of bin 0 for gated data
NBITS   =                    8 / Nr of bits/datum (SEARCH mode 'X' data, else␣
↪1)
NSUBOFFS=                    0 / Subint offset (Contiguous SEARCH-mode files)
NCHAN   =                 2048 / Number of channels/sub-bands in this file
CHAN_BW =            -0.390625 / [MHz] Channel/sub-band width
NCHNOFFS=                    0 / Channel/sub-band offset for split files
NSBLK   =                 4096 / Samples/row (SEARCH mode, else 1)
EXTNAME = 'SUBINT  '           / name of this binary table extension
TUNIT1  = 's       '           / Units of field
TUNIT2  = 's       '           / Units of field
TUNIT3  = 's       '           / Units of field
TUNIT4  = 'deg     '           / Units of field
TUNIT5  = 'deg     '           / Units of field
TUNIT6  = 'deg     '           / Units of field
TUNIT7  = 'deg     '           / Units of field
TUNIT8  = 'deg     '           / Units of field
TUNIT9  = 'deg     '           / Units of field
TUNIT10 = 'deg     '           / Units of field
TUNIT11 = 'deg     '           / Units of field
TUNIT12 = 'deg     '           / Units of field
TUNIT13 = 'MHz     '           / Units of field
TDIM17  = '(1,2048,4,4096)'    / Dimensions (NBITS or NBIN,NCHAN,NPOL,NSBLK)
TUNIT17 = 'Jy      '           / Units of subint data
EXTVER  =                    1 / auto assigned by template parser
```

In order to set the dimensions of the data arrays within the SUBINT HDU there is a convenience function called `set_subint_dims`. By setting the dimensions using this function the dependencies on these dimensions, incldiung memory allocation, will be propagated through the headers correctly.

First lets choose some dimensions for the data.

```
sample_size = 20.48e-3 # in milliseconds
ROWS = 30
N_Time_Bins = 2048*ROWS
Total_time = round(N_Time_Bins*sample_size)
dt = Total_time/N_Time_Bins
subband =1.5625
BW=200
N_freq = int(BW/subband)
Npols = 4
print('Total_time',Total_time/1e3)
print('N_freq',N_freq)
```

```
Total_time 1.258
N_freq 128
```

And then call the `set_subint_dims` method.

```
psrfits1.set_subint_dims(nsblk=2048,nchan=N_freq,nsubint=ROWS,npol=Npols)
```

Once we have set the `SUBINT` dimensions a `subint_dtype` list is made which we can then use to make a recarray to hold the data. Here `nsubint` is the same as above, and has been made an attribute.

```
subint_draft = psrfits1.make_HDU_rec_array(psrfits1.nsubint, psrfits1.subint_dtype)
```

All of the header cards can bet set by assigning them to the appropriate member of the draft header.

```
npol = psrfits1.draft_hdrs['SUBINT']['NPOL']
```

Here we set the time per subintegration (time length of an NSBLK) and the offsets, which are the times at the center of each subintegration from the beginning of the observation.

```
tsubint = data.shape[-1]*dt*1e-3 #in seconds
offs_sub_init = tsubint/2
offs_sub = np.zeros((ROWS))

for jj in range(ROWS):
    offs_sub[jj] = offs_sub_init + (jj * tsubint)
```

Here we just use the values from the template file.

```
lst_sub = psrfits1.fits_template[1]['LST_SUB'].read()[0]
ra_sub = psrfits1.fits_template[1]['RA_SUB'].read()[0]
dec_sub = psrfits1.fits_template[1]['DEC_SUB'].read()[0]
glon_sub = psrfits1.fits_template[1]['GLON_SUB'].read()[0]
glat_sub = psrfits1.fits_template[1]['GLAT_SUB'].read()[0]
fd_ang = psrfits1.fits_template[1]['FD_ANG'].read()[0]
pos_ang = psrfits1.fits_template[1]['POS_ANG'].read()[0]
par_ang = psrfits1.fits_template[1]['PAR_ANG'].read()[0]
tel_az = psrfits1.fits_template[1]['TEL_AZ'].read()[0]
tel_zen = psrfits1.fits_template[1]['TEL_ZEN'].read()[0]

ones = np.ones((ROWS))
#And assign them using arrays of the appropriate sizes
subint_draft['TSUBINT'] = tsubint * ones
subint_draft['OFFS_SUB'] = offs_sub
subint_draft['LST_SUB'] = lst_sub * ones
subint_draft['RA_SUB'] = ra_sub * ones
```

(continues on next page)

(continued from previous page)

```
subint_draft['DEC_SUB'] = dec_sub * ones
subint_draft['GLON_SUB'] = glon_sub * ones
subint_draft['GLAT_SUB'] = glat_sub * ones
subint_draft['FD_ANG'] = fd_ang * ones
subint_draft['POS_ANG'] = pos_ang * ones
subint_draft['PAR_ANG'] = par_ang * ones
subint_draft['TEL_AZ'] = tel_az * ones
subint_draft['TEL_ZEN'] = tel_zen * ones
```

Here we'll just make some data of the correct shape.

```
data = np.random.randn(ROWS,1,N_freq,Npols,2048)
```

And now we can assign the data arrays

```
for ii in range(subint_draft.size):
    subint_draft[ii]['DATA'] = data[ii,:,:,:,:]
    subint_draft[ii]['DAT_SCL'] = np.ones(N_freq*npol)
    subint_draft[ii]['DAT_OFFS'] = np.zeros(N_freq*npol)
    subint_draft[ii]['DAT_FREQ'] = np.linspace(1300,1500,N_freq)
    subint_draft[ii]['DAT_WTS'] = np.ones(N_freq)
```

```
subint_hdr=psrfits1.draft_hdrs['SUBINT']
```

```
from decimal import *
getcontext().prec=12
a=Decimal(S1.TimeBinSize*1e-3)
a.to_eng_string()
```

```
'0.000020475260416666664749435824988132992530154297128319740295410156 25'
```

```
b='{0:1.18f}'.format(Decimal(a.to_eng_string()))
b
```

```
'0.000020475260416667'
```

```
pri_dic= {'OBSERVER':'GALILEOGALILEI','OBSFREQ':S1.f0,'OBSBW':S1.bw,'OBSNCHAN':S1.Nf}
subint_dic = {'TBIN':b,'CHAN_BW':S1.freqBinSize}
```

```
subint_dic['TBIN']
```

```
'0.000020475260416667'
```

```
psrfits1.make_FITS_card(subint_hdr,'TBIN',subint_dic['TBIN'])
```

```
{'card_string': 'TBIN    = 0.000020475260416667 / [s] Time per bin or sample',
 'class': 150,
 'comment': '[s] Time per bin or sample',
 'dtype': 'F',
 'name': 'TBIN',
 'value': 2.0475260416667e-05,
 'value_orig': 2.0475260416667e-05}
```

```
psrfits1.draft_hdrs['SUBINT'].records()[65]
```

```
{'card_string': "TUNIT8  = 'deg     '           / Units of field",
 'class': 70,
 'comment': 'Units of field',
 'dtype': 'C',
 'name': 'TUNIT8',
 'value': 'deg     ',
 'value_orig': 'deg     '}
```

```
set_primary_header(psrfits1,pri_dic)
```

```
set_subint_header(psrfits1,subint_dic)
```

```
psrfits1.draft_hdrs['SUBINT'].records()[47]['value'] = '0.000020483398437500'
psrfits1.draft_hdrs['SUBINT'].records()[47]['value_orig'] = '0.000020483398437500'
psrfits1.draft_hdrs['SUBINT'].records()[47]
```

```
{'card_string': 'TBIN    = 0.000020475260416667 / [s] Time per bin or sample',
 'class': 150,
 'comment': '[s] Time per bin or sample',
 'dtype': 'F',
 'name': 'TBIN',
 'value': '0.000020483398437500',
 'value_orig': '0.000020483398437500'}
```

```
psrfits1.HDU_drafts['SUBINT'] = subint_draft
```

```
psrfits1.write_psrfits()
```

```
psrfits1.close()
```

CHAPTER 6

pdat

6.1 pdat package

Top-level package for PulsarDataToolbox.

6.1.1 Subpackages

pdat.templates package

__init__.py file to make templates a module

Submodules

pdat.templates.template module

Functions to get the built-in template file

pdat.templates.template.**get_template**(*template_name*)

Function to get the psrfits template from the templates module dir

template_name: str The name of the built-in template.

Full path to the template file. If the request file does not exists, it will return None.

6.1.2 Submodules

6.1.3 pdat.pdat module

Main module.

class pdat.pdat.**psrfits**(*psrfits_path*, *mode=u'rw'*, *from_template=False*, *obs_mode=None*, *verbose=True*)

Bases: fitsio.fitslib.FITS

Class which inherits fitsio.FITS() (Python wrapper for cfitsio) class's functionality, and add's new functionality to easily manipulate and make PSRFITS files.

from_template [bool, str] Either a boolean which dictates if a copy would like to be made from a template, or a string which is the path to a user chosen template.

psrfits_path [str] Either the path to an existing PSRFITS file or the name for a new file.

obs_mode [Same as OBS_MODE in a standard PSRFITS, either SEARCH, PSR or] CAL for search mode, fold mode or calibration mode respectively.

mode [str, {'r', 'rw, 'READONLY' or 'READWRITE'}] Read/Write mode.

append_from_file(*path*, *table=u'all'*)

Method to append more subintegrations to a PSRFITS file from other PSRFITS files. Note: Tables are appended directly to the original file. Make a copy

before copying if you are unsure about appending. The array must match the columns (in the numpy.recarray sense) of the existing PSRFITS file.

path [str] Path to the new PSRFITS file to be appended.

table [list]

List of BinTable HDU headers to append from file. Defaults to appending all secondary BinTables. ['HISTORY','PSRPARAM','POLYCO','SUBINT']

close()

Override of fitsio close method. Adds more variables to set to none. Close the fits file and set relevant metadata to None

copy_template_BinTable(*ext_name*, *cols=u'all'*, *dtypes=None*)

Method to copy PSRFITS binary tables exactly. This is especially useful when using real PSRFITS files to make simulated data, i.e. if you would just like to replace the DATA arrays in the file with your simulated data, but keep the ancillary telescope information. This copies the BinTable as a numpy.recarray into the *HDU_drafts* dictionary.

ext_name [str, {'PRIMARY','SUBINT','HISTORY','PSRPARAM','POLYCO'}] Binary Extension name to copy.

cols [str or list] Columns of the given BinTable to copy.

dtypes [list of tuples] Data types for numpy.recarray that will be the draft for the BinTable.

get_FITS_card_dict(*hdr*, *name*)

Make a FITS card compatible dictionary from a template FITS header that matches the input name key in a standard FITS card/record. It is necessary to make a new FITS card/record to change values in the header. This function outputs a writeable dictionary which can then be used to change the value in the header using the hdr.add_record() method.

hdr [fitsio.fitslib.FITSHDR object] Template for the card.

name [str] The name key in the FITS record you wish to make.

get_HDU_dtypes(*HDU*)

Returns a list of data types and array sizes needed to make a recarray. HDU = A FITS HDU.

get_colnames()
Returns the names of all of the columns of data needed for a PSRFITS file.

make_FITS_card(*hdr*, *name*, *new_value*)
Make a new FITS card/record using a FITS header as a template. This function makes a new card by finding the card/record in the template with the same name and replacing the value with new_value. Note: fitsio will set the dtype dependent on the form of the new_value for numbers.

hdr [fitsio.fitslib.FITSHDR] A fitsio.fitslib.FITSHDR object, which acts as the template.

name [str] A string that matches the name key in the FITS record you wish to make.

new_value [str, float] The new value you would like to replace.

make_HDU_rec_array(*nrows*, *HDU_dtype_list*)
Makes a rec array with the set number of rows and data structure dictated by the dtype list.

replace_FITS_Record(*hdr*, *name*, *new_value*)
Replace a Fits record with a new value in a fitsio.fitslib.FITSHDR object.

hdr [str or FITSHDR object] Header name.

name [FITS Record/Car] FITS Record/Card name to replace.

new_value [float, str] The new value of the parameter.

set_HDU_array_shape_and_dtype(*HDU_dtype_list*, *name*, *new_array_shape=None*, *new_dtype=None*)
Takes a list of data types (output of get_HDU_dtypes()) and returns new list with the named element's array shape and/or data type edited.

HDU_dtype_list : dtype list for making recarray (output of get_HDU_dtypes()).

name [str] Name of parameter to edit.

new_array_shape [tuple] New array shape. Note 1-d arrays are of type (n,) in FITS files.

new_dtype : New data type. See PSRFITS and fitsio documentation for recognized names.

set_draft_header(*ext_name*, *hdr_dict*)
Set draft header entries for the new PSRFITS file from a dictionary.

psrfits_object [pdat.psrfits] Pulsar Data Toolbox PSRFITS object.

ext_name [str, {'PRIMARY','SUBINT','HISTORY','PSRPARAM','POLYCO'}] Name of the header to replace the header entries.

hdr_dict [dict] Dictionary of header changes to be made to the template header. Template header entries are kept, unless replaced by this function.

set_hdr_from_draft(*hdr*)
Sets a header of the PSRFITS file using the draft header derived from template.

set_subint_dims(*nbin=1*, *nchan=2048*, *npol=4*, *nsblk=4096*, *nsubint=4*, *obs_mode=None*, *data_dtype=u'\u1'*)
Method to set the appropriate parameters for the SUBINT BinTable of a PSRFITS file of the given dimensions.

The parameters above are defined in the PSRFITS literature. The method automatically changes all the header information in the

template dependent on these values. The header template is set to these values.

A list version of a dtype array is made which has all the info needed to make a SUBINT recarray. This can then be written to a PSRFITS file, using the command write_prsfits().

nbin [int] NBIN, number of bins. 1 for SEARCH mode data.

nchan [int] NCHAN, number of frequency channels.

npol [int] NPOL, number of polarization channels.

nsblk [int] NSBLK, size of the data chunks for search mode data. Set to 1 for PSR and CAL mode.

nsubint [int] NSUBINT or NAXIS2 . This is the number of rows or subintegrations in the PSRFITS file.

obs_mode [str , {'SEARCH', 'PSR', 'CAL'}] Observation mode.

data_type [str] Data type of the DATA array ('lu1'=int8 or 'lu2'=int16).

write_PrimaryHDU_info_dict(*ImHDU_template*, *new_ImHDU*)
Writes the information dictionary for a primary header Image HDU (new_ImHDU) using ImHDU_template as the template. Both are FITS HDUs.

ImHDU_template : Template header.

new_ImHDU : Header where template is copied.

write_psrfits(*HDUs=None*, *hdr_from_draft=True*)

Function that takes the template headers and a dictionary of recarrays to make into PSRFITS HDU's. These should only include BinTable HDU Extensions, not the PRIMARY header (an ImageHDU). PRIMARY is dealt with a bit differently.

HDUs [dict, optional] Dictionary of recarrays to make into HDUs. Default is set to HDU_drafts

pdat.pdat.**convert2asciii**(*dictionary*)
Changes all keys (i.e. assumes they are strings) to ASCII and values that are strings to ASCII. Specific to dictionaries.

pdat.pdat.**list_arg**(*list_name*, *string*)
Returns the index of a particular string in a list of strings.

6.1.4 pdat.pypsrfits module

class pdat.pypsrfits.**PyPSRFITS**(*fname=None*)
A version of Paul Demorest's pypsrfits routines added into the Pulsar Data Toolbox framework for ease of installation and documenting. I have replaced PSRFITS with PyPSRFITS to avoid confusion between this and the psrfits class, which is more generic.

See https://github.com/demorest/pypsrfits for more details.

This is a very simple python module for reading search-mode PSRFITS data into python. It requires the fitsio python module (and numpy of course).

Example usage:

Import the module, open a file import pypsrfits f = pdat.PyPSRFITS('my_file.fits')

A full fitsio object for the file is available: f.fits

The main header and SUBINT header are also accessible: f.hdr f.subhdr

Read all data from row 13 d = f.get_data(13)

Read all data in entire file, downsampling in time by # a factor of 256 d = f.get_data(0,-1,downsamp=256)

get_data(*start_row=0*, *end_row=None*, *downsamp=1*, *fdownsamp=1*, *apply_scales=True*, *get_ft=False*, *squeeze=False*)

Read the data from the specified rows and return it as a single array. Dimensions are [time, poln, chan]. options:

start_row: first subint read (0-based index) end_row: final subint to read. None implies end_row=start_row.

Negative values imply offset from the end, i.e. get_data(0,-1) would read the entire file. (Don't forget that PSRFITS files are often huge so this might be a bad idea).

downsamp: downsample the data in time as they are being read in. The downsample factor should evenly divide the number of spectra per row. downsamp=0 means integrate each row completely.

fdownsamp: downsample the data in freq as they are being read in. The downsample factor should evenly divide the number of channels.

apply_scales: set to False to avoid applying the scale/offset data stored in the file.

get_ft: if True return time and freq arrays as well. squeeze: if True, "squeeze" the data array (remove len-1

dimensions).

Notes:

- Only 8, 16, and 32 bit data are currently understood

get_freqs(*row=0*)

Return the frequency array from the specified subint.

open(*fname*)

Open the specified PSRFITS file. A fitsio object is created and stored as self.fits. For convenience, the main header is stored as self.hdr, and the SUBINT header as self.subhdr.

CHAPTER 7

Contributing

Contributions are welcome, and they are greatly appreciated! Every little bit helps, and credit will always be given.

You can contribute in many ways:

7.1 Types of Contributions

7.1.1 Report Bugs

Report bugs at https://github.com/Hazboun6/PulsarDataToolbox/issues.

If you are reporting a bug, please include:

- Your operating system name and version.
- Any details about your local setup that might be helpful in troubleshooting.
- Detailed steps to reproduce the bug.

7.1.2 Fix Bugs

Look through the GitHub issues for bugs. Anything tagged with "bug" and "help wanted" is open to whoever wants to implement it.

7.1.3 Implement Features

Look through the GitHub issues for features. Anything tagged with "enhancement" and "help wanted" is open to whoever wants to implement it.

7.1.4 Write Documentation

PulsarDataToolbox could always use more documentation, whether as part of the official PulsarDataToolbox docs, in docstrings, or even on the web in blog posts, articles, and such.

7.1.5 Submit Feedback

The best way to send feedback is to file an issue at https://github.com/hazboun6/PulsarDataToolbox/issues.

If you are proposing a feature:

- Explain in detail how it would work.
- Keep the scope as narrow as possible, to make it easier to implement.
- Remember that this is a volunteer-driven project, and that contributions are welcome :)

7.2 Get Started!

Ready to contribute? Here's how to set up *PulsarDataToolbox* for local development.

1. Fork the *PulsarDataToolbox* repo on GitHub.
2. Clone your fork locally:

```
$ git clone git@github.com:your_name_here/PulsarDataToolbox.git
```

3. Install your local copy into a virtualenv. Assuming you have virtualenvwrapper installed, this is how you set up your fork for local development:

```
$ mkvirtualenv PulsarDataToolbox
$ cd PulsarDataToolbox/
$ python setup.py develop
```

4. Create a branch for local development:

```
$ git checkout -b name-of-your-bugfix-or-feature
```

   Now you can make your changes locally.

5. When you're done making changes, check that your changes pass flake8 and the tests, including testing other Python versions with tox:

```
$ flake8 pdat tests
$ python setup.py test or py.test
$ tox
```

   To get flake8 and tox, just pip install them into your virtualenv.

6. Commit your changes and push your branch to GitHub:

```
$ git add .
$ git commit -m "Your detailed description of your changes."
$ git push origin name-of-your-bugfix-or-feature
```

7. Submit a pull request through the GitHub website.

7.3 Pull Request Guidelines

Before you submit a pull request, check that it meets these guidelines:

1. The pull request should include tests.
2. If the pull request adds functionality, the docs should be updated. Put your new functionality into a function with a docstring, and add the feature to the list in README.rst.
3. The pull request should work for Python 2.6, 2.7, 3.3, 3.4 and 3.5, and for PyPy. Check https://travis-ci.org/hazboun6/PulsarDataToolbox/pull_requests and make sure that the tests pass for all supported Python versions.

7.4 Tips

To run a subset of tests:

```
$ python -m unittest tests.test_pdat
```

CHAPTER 8

Credits

8.1 Development Lead

- Jeffrey S Hazboun <jeffrey.hazboun@gmail.com>

8.2 Contributors

Paul Demorest (GitHub: demorest)

CHAPTER 9

History

9.1 0.2.1 (2018-12-17)

- Fixed Python 2 string bug.

9.2 0.2.0 (2018-12-17)

- Second release on PyPI.
- Added ability to use deal effectively with PSR mode PSRFITS files.
- Fixed issue with ASCII text.

9.3 0.1.0 (2017-08-21)

- First release on PyPI.

CHAPTER 10

Indices and tables

- genindex
- modindex
- search

Python Module Index

e

p

Index