Memo: UVCal FITS Format (*.calfits*)

Zaki Ali, Bryna Hazelton, Adam Beardsley, Paul La Plante, and the pyuvdata team

July 3, 2017
Revised May 7, 2020

1 Introduction

This memo introduces a new FITS-based file format for storing calibration solutions to use with pyuvdata[1], a python package which provides a software interface for interferometry. We describe the required and optional parameters of a UVCal FITS—hereafter *calfits*—file and pyuvdata's interface for reading and writing these files. For usage examples please refer to the pyuvdata tutorial: `http://pyuvdata.readthedocs.io/en/latest/tutorial.html`.

2 Overview

Calfits is an adaptation of the FITS file format to enable the storage of calibration information for radio interferometric arrays. Because it builds on the existing FITS file format, all *calfits* files should be properly formatted according to the FITS standard which is widely available. Accordingly, required standard FITS keywords (e.g., SIMPLE, BITPIX, etc.) will not be discussed in this document, even though they may still be required for a valid *calfits* file.

Any valid *calfits* file corresponds directly to a UVCal object within pyuvdata. As such, every new HDU keyword introduced here has a one-to-one correspondence with UVCal object parameters. In this memo, each new keyword is followed by its corresponding UVCal parameter, in parentheses. For more information about the UVCal class and its parameters, please refer to pyuvdata's documentation: `http://pyuvdata.readthedocs.io/en/latest/uvcal.html`.

2.1 The FITS file

A UVCal object stores calibration solutions as either a "gain" type solution, or as a "delay" type solution. These types represent two distinct but mathematically equivalent conven-

[1] `https://github.com/RadioAstronomySoftwareGroup/pyuvdata`

tions, both widely used in radio astronomy. Depending on the calibration type (gain or delay), the *calfits* format may consist of up to 4 HDUs. In either case, the primary header is the same and consists of relevant meta-information for a UVCal object to be instantiated. The second HDU is also the same in either case and is the ANTENNAS HDU. This HDU is a BinaryTable and consists of ANTNAME, ANTINDEX, and ANTARR, corresponding to antenna_names, antenna_numbers, and ant_array in the above list, respectively.

When the calibration type is "gain", the essential data contains only these 2 HDUs. In this case, the image data in the primary HDU consists is a 6 dimensional array, where each dimension corresponds to (Nants, Nspws, Nfreqs, Ntimes, Njones, Number of arrays in image array), respectively. In other words, the primary data HDU contains the 5 axes of the data given in the list above, and then a sixth axis corresponding to the individual quantities being saved. For instance, if there is an input_flag_array the image array consists of [real(gain_array), imag(gain_array), flag_array, input_flag_array, quality_array], which is concatenated along the last axis and so the last dimension is equal to 5. However, if no input_flag_array is given, the input_flag_array is left out of the above array and a the last dimension is equal to 4.

When the calibration type is "delay", there are 3 data HDUs. The image data in the primary HDU is still a 5 dimensional array as before (dimensions are Nants, Nfreqs, Ntimes, Njones, number of arrays in image array), but with Nfreqs = 1 as a placeholder axis. This axis is added to keep the data arrays the same size between the delay-type and gains-type formats. In this case the image data is [delay_array, quality_array], concatenated along the last axis. The flag arrays are stored in the third HDU (ImageHDU) which has the flag_array and may have an input_flag_array.

For both delay-types, there is also an optional total_quality_array HDU, which contains information about the overall χ^2 value of the whole array. The size of the array is (Nspws, Nfreqs, Ntimes, Njones). For delay-type calibrations, Nfreqs = 1 as above. If present, there will be 3 total HDUs for gain-type files, and 4 total HDUs for delay-type. Note that self-consistency checks are run when reading and writing calfits files to ensure that arrays have the proper size across different HDUs.

3 Primary Header

The following are required keywords in the primary header of a *calfits* file. For a more detailed explanation of what these keywords mean, see the descriptions on pyuvdata's ReadTheDocs uvcal_parameters page. The uvcal parameter corresponding to each keyword is noted in parentheses. As with all FITS files, **HISTORY** and **COMMENT** cards are optional and allowed.

- **GNCONVEN**: *string* Gain convention: the convention for applying the calibration solutions to data. Values are "divide" or "multiply", indicating whether one should divide or multiply uncalibrated data by gains. Mathematically this indicates the alpha exponent in the equation: (calibrated data) = (gain$^{\alpha}$) $\times$ (uncalibrated data). A value of “divide” represents $\alpha = -1$ and “multiply” represents $\alpha = 1$. (gain_convention).

- **CALTYPE**: *string* Calibration type parameter. Possible values are “delay”, “gain”, or “unknown”. (cal_type).
- **CALSTYLE**: *string* Style of calibration. Possible values are “sky” or “redundant”. (cal_style).
- **FIELD**: *string* (Required if CALSTYLE = “sky”.) A short string describing the field center or dominant source. (sky_field).
- **CATALOG**: *string* (Required if CALSTYLE = “sky”.) Name of the calibration catalog. (sky_catalog).
- **REFANT**: *string* (Required if CALSTYLE = “sky”.) Phase reference antenna. (ref_antenna_name)
- **Nants_data**: Number of antennas that have data associated with them (i.e. number of unique entries in ant_array). May be smaller than the number of antennas in the telescope’. (Nants_data)
- **Nants_telescope**: Number of antennas in the array. May be larger than the number of antennas with data. (Nants_telescope)
- **Nfreqs**: Number of frequency channels. (Nfreqs)
- **Njones**: Number of Jones calibration parameters (Number of jones matrix elements calculated in calibration). (Njones)
- **Nspws**: Number of spectral windows (ie non-contiguous spectral chunks). More than one spectral window is not currently supported. (Nspws)
- **Ntimes**: Number of times with different calibrations calculated (if a calibration is calculated over a range of integrations, this gives the number of separate calibrations along the time axis). (Ntimes)
- **ant_array**: Array of antenna indices for data arrays, shape (Nants_data). type = int, 0 indexed. (ant_array)
- **antenna_names**: List of antenna names, shape (Nants_telescope), with numbers given by antenna_numbers (which can be matched to ant_array). There must be one entry here for each unique entry in ant_array, but there may be extras as well. (antenna_names)

- **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_array, but there may be extras as well. (antenna_numbers)
- **channel_width**: Channel width of of a frequency bin. Units Hz. (channel_width)
- **flag_array**: Array of flags to be applied to calibrated data (logical OR of input and flag generated by calibration). True is flagged. Shape: (Nants_data, Nspws, Nfreqs, Ntimes, Njones), type = bool. (flag_array)
- **freq_array**: Array of frequencies, shape (Nspws, Nfreqs), units Hz. (freq_array)
- **history**: String of history. (history)
- **integration_time**: Integration time of a time bin, units seconds. (integration_time)
- **jones_array**: Array of antenna polarization integers, shape (Njones). linear pols -5:-8 (jxx, jyy, jxy, jyx).circular pols -1:-4 (jrr, jll. jrl, jlr). (jones_array)
- **quality_array**: Array of qualities of calibration solutions. The shape depends on cal_type, if the cal_type is "gain" or "unknown" the shape is: (Nants_data, Nspws, Nfreqs, Ntimes, Njones), if the cal_type is "delay" the shape is: (Nants_data, Nspws, 1, Ntimes, Njones), type = float. (quality_array)
- **spw_array**: Array of spectral window numbers, shape (Nspws). (spw_array)
- **telescope_name**: Name of telescope. e.g. HERA. String. (telescope_name)
- **time_array**: Array of calibration solution times, center of integration, shape (Ntimes), units Julian Date. (time_array)
- **time_range**: Time range (in JD) that gain solutions are valid for. list: [start_time, end_time] in JD. (time_range)
- **x_orientation**: Orientation of the physical dipole corresponding to what is labelled as the x polarization. Values are east (east/west orientation), north (north/south orientation) or unknown. (x_orientation)

There are also some optionally required parameters that depend on the calibration type. These parameters include.

- **delay_array**: Required if cal_type = "delay". Array of delays with units of seconds. Shape: (Nants_data, Nspws, 1, Ntimes, Njones), type = float. (delay_array)
- **gain_array**: Required if cal_type = "gain". Array of gains, shape: (Nants_data, Nspws, Nfreqs, Ntimes, Njones), type = complex float. (gain_array)

- **freq_range**: Required if cal_type = “delay”. Frequency range that solutions are valid for. list: [start_frequency, end_frequency] in Hz. (freq_range)

In addition to the required parameters, there are a number of truly optional parameters that may be passed in. These include:

- **git_origin_cal**: Origin (on github for e.g) of calibration software. Url and branch. (git_origin_cal)
- **git_hash_cal**: Commit hash of calibration software (from git_origin_cal) used to generate solutions. (git_hash_cal)
- **input_flag_array**: Array of input flags, True is flagged. shape: (Nants_data, Nspws, Nfreqs, Ntimes, Njones), type = bool. (input_flag_array)
- **total_quality_array**: Array of qualities of the calibration solution for the entire array. The shape depends on cal_type, if the cal_type is ”gain” or ”unknown”, the shape is: (Nspws, Nfreqs, Ntimes, Njones), if the cal_type is ”delay”, the shape is (Nspws, 1, Ntimes, Njones), type = float. (total_quality_array)

Once these parameters are set in the UVCal object, a *calfits* file may be written out.

4 Reading and Writing a *calfits* File

Writing out the UVCal object to a file is very simple: just run UVCal.write_calfits(filename). That will write a fits file called “filename”. Note that a filename check will be done and a new file will not be written with the same name. You can override this functionality with the clobber key word.

Reading in a calfits file is also straightforward. First instantiate the UVCal object and then run UVCal.read_calfits(filename). This updates the UVCal object with all the parameters from the the fits file.

There are examples of working with pyuvdata UVCal objects and *calfits* files in the tutorial (`http://pyuvdata.readthedocs.io/en/latest/tutorial.html`).