Gamma Astro Data Specs Documentation

Release 0.1

Gamma-ray astronomy community

Nov 17, 2017

Contents

A place to propose and share data format descriptions for gamma-ray astronomy.

- Repository: https://github.com/open-gamma-ray-astro/gamma-astro-data-formats
- Docs: https://gamma-astro-data-formats.readthedocs.io/
- Mailing list: https://lists.nasa.gov/mailman/listinfo/open-gamma-ray-astro

CHAPTER 1

Table of contents

1.1 General

This section contains general information and basic definitions.

It's purpose is two-fold:

1. Users and developers can learn or look up the nitty-gritty details how coordinates, times, ... are defined and basic information about file and storage formats (e.g. how axis-information for multi-dimensional arrays can be stored in FITS files).
2. Data format specifications can refer to the definitions in this section, e.g. we don't have to repeat that the azimuth angle is measured east of north in each format specification where azimuth is used.

1.1.1 About

What is this?

This is a grassroots effort to describe data formats that are in use, but are not specified anywhere else. Examples range from FITS format instrument response functions for imaging atmospheric Cherenkov telescopes (IACTs) to YAML format spectral and spatial model specifications, to high-level analysis results like spectra and light curves.

The formats described here and the whole effort are not official in any way, i.e. not supported or ratified by any institutes, collaborations or committees. In some cases the formats described here will likely be adopted or superceded by more official formats in the coming years.

Everyone is welcome to adopt these formats or even contribute, development and discussion is done openly on Github. Especially if you are a software library or tool developer, we encourage you to support the formats described here instead of inventing your own.

How to contribute?

The documentation is written in restructured text (RST) and rendered to HTML and PDF with Sphinx and hosted at Readthedocs.

Everyone can contribute by making a pull request with a change or addition to https://github.com/open-gamma-ray-astro/gamma-astro-data-formats or by sending comments and feedback via the Github issue tracker, or, for high-level and important things, to https://lists.nasa.gov/mailman/listinfo/open-gamma-ray-astro .

We use the Sphinx Readthedocs theme as described in this Sphinx RTD theme FAQ entry, i.e. to build the HTML docs locally you have to `pip install sphinx_rtd_theme` before `make html`.

References

Existing FITS specs and recommendations:

- http://fits.gsfc.nasa.gov/fits_home.html
- http://fits.gsfc.nasa.gov/registry/grouping.html

Existing HEASARC specs and recommendations:

- https://heasarc.gsfc.nasa.gov/docs/heasarc/ofwg/ofwg_recomm.html
- http://heasarc.gsfc.nasa.gov/docs/heasarc/caldb/caldb_doc.html

1.1.2 Time

This page gives background information on times in gamma-ray astronomy.

It's not a format specification, rather a summary of the status quo:

- How times are stored in files.
- How times are represented in science tool codes
- How times are input by users and output to users from these codes.

Introduction

Times are used in many places in high-level analysis, e.g.

- Observations have start and end times and sometimes are split up into "good time intervals" GTIs when hardware issues occur or clouds pass the field of view.
- Gamma-ray events are observed at given times, and those times are needed to convert the reconstructed AltAz position to RaDec, or to select events in a given GTI.
- Some gamma-ray sources are variable, e.g. AGNs can flare on timescales of seconds or minutes, or pulsars emit a periodic signal on timescales of seconds or milli-seconds.

This page contains specifications and recommendations how to work with times for high-level gamma-ray astronomy, i.e. how to store times in files (e.g. event lists, GTI extension, observation tables) and take times as input and output in analysis tools.

Reference documents and tools

Basically we follow Time in Fermi data analysis, so this is the number one reference.

The SOFA Time Scale and Calendar Tools document provides a detailed description of times in the high-precision IAU SOFA library, which is the gold standard for times in astronomy. The SOFA time routines are available via the Astropy time Python package, which makes it easy to convert between different **time scales** (`utc`, `tt` and `mjd` in this example)

```
>>> from astropy.time import Time
>>> time = Time('2011-01-01 00:00:00', scale='utc', format='iso')
>>> time
<Time object: scale='utc' format='iso' value=2011-01-01 00:00:00.000>
>>> time.tt
<Time object: scale='tt' format='iso' value=2011-01-01 00:01:06.184>
>>> time.mjd
55562.0
```

as well as different **time formats** (`iso`, `isot` and `fits` in this example)

```
>>> time.iso
'2011-01-01 00:00:00.000'
>>> time.isot
'2011-01-01T00:00:00.000'
>>> time.fits
'2011-01-01T00:00:00.000(UTC)'
```

If you don't want to install SOFA or Astropy (or to double-check), you can use the xTime time conversion utility provided by HEASARC as a web tool.

Finally, the "Representation of Time Coordinates in FITS" standard (2015A%26A...574A..36R) explains in detail how times should be stored in FITS files.

Precision

Depending on the use case and required precision, times are stored as strings or as one or several integer or floating point numbers. Tools usually use one 64-bit or two 64-bit floating point numbers for time calculations.

For high-level gamma-ray astronomy, the situation can be summarised like this (see sub-section computation below for details):

- **Do use single 64-bit floats for times.** The resulting precision will be about 0.1 micro-seconds or better, which is sufficient for any high-level analysis (including milli-second pulsars).
- **Do not use 32-bit floats for times.** If you do, times will be incorrect at the 1 to 100 second level.

For data acquisition and low-level analysis (event triggering, traces, ...), IACTs require nanosecond precision or better. There, the simple advice to use 64-bit floats representing seconds wrt. a single reference time doesn't work! One either needs to have several reference times (e.g. per-observation) or two integer or float values. This is not covered by this spec.

Computation

The time precision obtained with a single 32-bit or 64-bit float can be computed with this function:

```
def time_precision(time_range, float_precision):
    """Compute time precision (seconds) in float computations.

    For a given `time_range` and `float_precision`, the `time_precision`
    is computed as the smallest time difference corresponding to the
    float precision.

    time_range -- (IN) Time range of application (years)
    float_precision -- (IN) {32, 64} Floating point precision
    time_precision -- (OUT) Time precision (seconds)
    """
    import numpy as np
    YEAR_TO_SEC = 315576000

    dtype = {32: np.float32, 64: np.float64}[float_precision]
    t1 = dtype(YEAR_TO_SEC * time_range)
    t2 = np.nextafter(t1, np.finfo(dtype).max)
    print('Time range: {} years, float precision: {} bit => time precision: {:.3g}␣
↪seconds.'
          ''.format(time_range, float_precision, t2-t1))
```

```
>>> time_precision(10, 32)
Time range: 10 years, float precision: 32 bit => time precision: 256 seconds.
>>> time_precision(10, 64)
Time range: 10 years, float precision: 64 bit => time precision: 4.77e-07 seconds.
```

Files

Here's a summary of how times are stored in files:

- *EVENTS extension*:
  - Table column `TIME`, `float64`, MET
  - Header keywords `MJDREFI`, `MJDREFF` – Reference time
  - Header keywords `TSTART`, `TSTOP` – MET
  - `TSTART_STR`, `TSTOP_STR` – UTC or TT str -> TIMESYS.
  - `TIMESYS`, `TIMEREF` – need it?
- *GTI extension*:
  - Table columns: `TSTART`, `TSTOP`, MET
  - Header keywords: `MJDREFI`, `MJDREFF` – Reference time
- *Observation index table*
  - Column `TSTART`, `TSTOP`, `TMID` – MET
  - Column `TSTART_STR`, `TSTOP_STR`, `TMID_STR` – UTC string
  - Header keywords `MJDREFI`, `MJDREFF` – Reference time
  - Header keywords `TIMEUNIT`, `TIMESYS`

Tools

Here's a summary of how gamma-ray science tool codes handle times.

Fermi Science Tools

The Fermi Science tools (e.g. gtselect) support only Fermi-LAT MET for user input / output (and probably also just use MET internally). Some info on other time scales and formats is given on a docs page at Time in Fermi data analysis, converting to MET is left up to the user. Note that no leap second table is in the code, i.e. MET – UTC conversions are not supported (one can use Astropy for this though).

- TODO: Is this correct? How does the software store times internally?

TODO: We should also document what time scales and formats are supported by the Fermi-LAT data selection tool:

- http://fermi.gsfc.nasa.gov/cgi-bin/ssc/LAT/LATDataQuery.cgi
- http://fermi.gsfc.nasa.gov/ssc/LATDataQuery_help.html#observationDates
- http://fermi.gsfc.nasa.gov/ssc/LATDataQuery_help.html#timeSystem

This is the equivalent of our *Observation index table* format and observation selection tools and unless there's a good reason not to we should just adopt whatever Fermi-LAT does here.

Gammalib / ctools

The ctools (e.g. ctselect) use MET for user input (the reference time is taken from the event list header). Internally a time is represented as a `GTime` object, which has a time scale (supports JD, MJD, TT, UTC, leap second table in the library code) and supports different formats (including parsing ISO and ISOT strings). Internally times are stored as 64-bit float METs wrt. a single reference time defined by Gammalib. See Times in Gammalib.

TODO: this means that TIME columns in event lists are converted to that reference time on file read or attribute access?

Astropy / Gammapy

As already mentioned above, the Astropy time package contains the `Time` class, which supports all common scales and formats. Internally times are stored as two 64-bit floats.

TODO: describe how MET values from event list `TIME` columns are converted to that internal format on read / write in `gammapy.time`. TODO: where do they store leap seconds / how are those updated?

Examples

TODO: write a set of tests doing equivalent time computations using Gammalib and Astropy time (or possibly Gammapy wrappers where useful).

1.1.3 Coordinates

This section describes the sky coordinates in use by science tools. It is referenced from the description of data formats to explain the exact meaning of the coordinates stored.

We don't have a separate section for world coordinate systems (WCS), pixel coordinates, projections, that is covered here as well (see FITS WCS and WCSLIB for references).

We only discuss 2-dimensional sky and image coordinates here, other coordinates like e.g. time or an energy axis aren't covered here.

Some conventions are adopted from astropy.coordinates, which is a Python wrapper of the IAU SOFA C time and coordinate library, which can be considered the gold standard when it comes to coordinates. In some cases code examples are given using *astropy.coordinates* to obtain a reference value that can be used to check a given software package (in case it's not based on *astropy.coordinates*).

RA / DEC

The most common way to give sky coordinates is as right ascension (RA) and declination (DEC) in the equatorial coordinate system.

Actually there are several equatorial coordinate systems in use, the most common ones being FK4, FK5 and ICRS. If you're interested to learn more about these and other astronomical coordinate systems, the references in the see also section for astropy.coordinates are a good starting point.

But in practice it's pretty simple: when someone gives or talks about RA / DEC coordinates, they mean either ICRS or FK5 J2000 coordinates. The difference between those two is at the sub-arcsecond level for the whole sky, i.e. irrelevant for gamma-ray astronomy.

We recommend you by default assume RA / DEC is in the ICRS frame, which is the default in astropy.coordinates.SkyCoord and also the current standard celestial reference system adopted by the IAU (see Wikipedia - ICRS).

Galactic

The Galactic coordinate system is often used by Galactic astronomers.

Unfortunately there are slightly different variants in use (usually with differences at the arcsecond level), and there are no standard names for these slightly different Galactic coordinate frames. See here for an open discussion which Galactic coordinates to support and what to call them in Astropy.

We recommend you use ICRS RA / DEC for precision coordinate computations. If you do use Galactic coordinates, we recommend you compute them like Astropy does (which I think is the most frame in use in the literature and in existing astronomy software).

Both ICRS and Galactic coordinates don't need the specification of an epoch or equinox.

To check your software, you can use the `(l, b) = (0, 0)` position:

```
>>> from astropy.coordinates import SkyCoord
>>> SkyCoord(0, 0, unit='deg', frame='galactic')
<SkyCoord (Galactic): (l, b) in deg (0.0, 0.0)>
>>> SkyCoord(0, 0, unit='deg', frame='galactic').icrs
<SkyCoord (ICRS): (ra, dec) in deg (266.40498829, -28.93617776)>
```

Alt / Az

The horizontal coordinate system is the one connected to an observer at a given location on earth and point in time.

- Azimuth is oriented east of north (i.e. north is at 0 deg, east at 90 deg, south at 180 deg and west at 270 deg). This is the convention used by astropy.coordinates.AltAz and quoted as the most common convention in astronomy on Wikipedia (see horizontal coordinate system).

- The zenith angle is defined as the angular separation from the zenith, which is the direction defined by the line connecting the Earth's center and the observer. Altitude and elevation are the same thing, and are defined as 90 degree minus the zenith angle. The reason to define altitude like this instead of the angle above the horizon is that usually Earth models aren't perfect spheres, but ellipsoids, so the zenith angle as defined here isn't perfectly perpendicular with the horizon plane.
- Unless explicitly specified, Alt / Az should be assumed to not include any refraction corrections, i.e. be valid assuming no refraction. Usually this can be achived in coordinate codes by setting the atmospheric pressure to zero, i.e. turning the atmosphere off.

Here's some Astropy coordinates code that shows how to convert back and forth between ICRS and AltAz coordinates (the default pressure is set to zero in Astropy, i.e. this is without refraction corrections):

```
import astropy.units as u
from astropy.time import Time
from astropy.coordinates import Angle, SkyCoord, EarthLocation, AltAz

# Take any ICRS sky coordinate
icrs = SkyCoord.from_name('crab')
print('RA = {pos.ra.deg:10.5f}, DEC = {pos.dec.deg:10.5f}'.format(pos=icrs))
# RA =   83.63308, DEC =   22.01450

# Convert to AltAz for some random observation time and location
# This assumes pressure is zero, i.e. no refraction
time = Time('2010-04-26', scale='tt')
location = EarthLocation(lon=42 * u.deg, lat=42 * u.deg, height=42 * u.meter)
altaz_frame = AltAz(obstime=time, location=location)
altaz = icrs.transform_to(altaz_frame)
print('AZ = {pos.az.deg:10.5f}, ALT = {pos.alt.deg:10.5f}'.format(pos=altaz))
# AZ =  351.88232, ALT =  -25.56281

# Convert back to ICRS to make sure round-tripping is OK
icrs2 = altaz.transform_to('icrs')
print('RA = {pos.ra.deg:10.5f}, DEC = {pos.dec.deg:10.5f}'.format(pos=icrs2))
# RA =   83.63308, DEC =   22.01450
```

Field of view

In Gamma-ray astronomy, sometimes field of view (FOV) coordinates are used. Specifically some *background models* are in the FOV coordinate system and FOV coordinates can also be used for other IRFs.

The basic idea is to have a coordinate system that is centered on the array pointing position. We define FOV coordinates here to be spherical coordinates, there is no projection or WCS, only a spherical rotation.

Two versions of FOV coordinates are defined:

1. **`(LON, LAT)` with the pointing position on the equator at `(0, 0)`**
    - `LON` range -180 deg to + 180 deg
    - `LAT` range -90 deg to + 90 deg
2. **`(THETA, PHI)` with the pointing position at the pole `THETA=0`**
    - `THETA` range 0 deg to +180 deg
    - `PHI` range 0 deg to 360 deg
    - `THETA` is the angular separation wrt. the pointing position.
    - TODO: define PHI orientation

- TODO: give example with numbers to make PHI orientation clear

Also, there are two versions of FOV coordinates defined:

1. Aligned with `ALTAZ`
2. Aligned with `RADEC`

To summarise, the following coordinates are defined:

Field	Description
FOV_ALTAZ_LON	Longitude in ALTAZ FOV system
FOV_ALTAZ_LAT	Latitude in ALTAZ FOV system
FOV_ALTAZ_THETA	Offset in ALTAZ FOV system
FOV_ALTAZ_PHI	Position angle in ALTAZ FOV system
FOV_RADEC_LON	Longitude in RADEC FOV system
FOV_RADEC_LAT	Latitude in RADEC FOV system
FOV_RADEC_THETA	Offset in RADEC FOV system
FOV_RADEC_PHI	Position angle in RADEC FOV system

1.1.4 FITS Multidimensional datasets

As described e.g. here or here or in the FITS Standard, there are several ways to serialise multi-dimensional arrays and corresponding axis information in FITS files.

Here we describe the schemes in use in gamma-ray astronomy and give examples.

IMAGE HDU

- Data array is stored in an IMAGE HDU.
- Axis information is either stored in the IMAGE HDU header or in extra BINTABLE HDUs, sometimes a mix.
- Advantage: IMAGE HDUs can be opened up in image viewers like ds9.
- Disadvantage: axis information is not self contained, an extra HDU is needed.

Example

E.g. the Fermi-LAT counts cubes or diffuse model spectral cubes are stored in an IMAGE HDU, with the information about the two celestial axes in WCS header keywords, and the information about the energy axis in ENERGIES (for spectral cube) or EBOUNDS (for counts cube) BINTABLE HDU extensions.

```
$ ftlist gll_iem_v02.fit H

        Name                 Type       Dimensions
        ----                 ----       ----------
HDU 1   Primary Array        Image      Real4(720x360x30)
HDU 2   ENERGIES             BinTable     1 cols x 30 rows
```

Let's have a look at the header of the primary IMAGE HDU.

As you can see, there's three axes.

The first two are Galactic longitude and latitude and the pixel to sky coordinate mapping is specified by header keywords according to the FITS WCS standard.

I think the energy axis isn't a valid FITS WCS axis specification. ds9 uses the *C????3* keys to infer a WCS mapping of pixels to energies, but it is incorrect. Software that's supposed to work with this axis needs to know to look at the *ENERGIES* table instead.

```
$ ftlist gll_iem_v02.fit K
SIMPLE  =                    T / Written by IDL:  Tue Jul  7 15:25:03 2009
BITPIX  =                  -32 /
NAXIS   =                    3 / number of data axes
NAXIS1  =                  720 / length of data axis 1
NAXIS2  =                  360 / length of data axis 2
NAXIS3  =                   30 / length of data axis 3
EXTEND  =                    T / FITS dataset may contain extensions
COMMENT   FITS (Flexible Image Transport System) format is defined in 'Astronomy
COMMENT   and Astrophysics', volume 376, page 359; bibcode: 2001A&A...376..359H
FLUX    =        8.29632317174 /
CRVAL1  =                   0. / Value of longitude in pixel CRPIX1
CDELT1  =                  0.5 / Step size in longitude
CRPIX1  =                360.5 / Pixel that has value CRVAL1
CTYPE1  = 'GLON-CAR'           / The type of parameter 1 (Galactic longitude in
CUNIT1  = 'deg     '           / The unit of parameter 1
CRVAL2  =                   0. / Value of latitude in pixel CRPIX2
CDELT2  =                  0.5 / Step size in latitude
CRPIX2  =                180.5 / Pixel that has value CRVAL2
CTYPE2  = 'GLAT-CAR'           / The type of parameter 2 (Galactic latitude in C
CUNIT2  = 'deg     '           / The unit of parameter 2
CRVAL3  =                  50. / Energy of pixel CRPIX3
CDELT3  =    0.113828620540137 / log10 of step size in energy (if it is logarith
CRPIX3  =                   1. / Pixel that has value CRVAL3
CTYPE3  = 'photon energy'      / Axis 3 is the spectra
CUNIT3  = 'MeV     '           / The unit of axis 3
CHECKSUM= '3fdO3caL3caL3caL'   / HDU checksum updated 2009-07-07T22:31:18
DATASUM = '2184619035'         / data unit checksum updated 2009-07-07T22:31:18
HISTORY From Ring/Hybrid fit with GALPROP 54_87Xexph7S extrapolation
HISTORY Integrated flux (m^-2 s^-1) over all sky and energies:    8.30
HISTORY Written by rings_gll.pro
DATE    = '2009-07-07'         /
FILENAME= '$TEMPDIR/diffuse/gll_iem_v02.fit' /File name with version number
TELESCOP= 'GLAST   '           /
INSTRUME= 'LAT     '           /
ORIGIN  = 'LISOC   '           /LAT team product delivered from the LISOC
OBSERVER= 'MICHELSON'          /Instrument PI
END
```

BINTABLE HDU

- Data array and axis information is stored in a BINTABLE HDU with one row.
- This is called the "multidimensional array" convention in appendix B of 1995A%26AS..113..159C.
- The OGIP Calibration Memo CAL/GEN/92-003 has a section use of multi-dimensional datasets that describes this format in greater detail.
- Advantage: everything is contained in one HDU. (as many axes and data arrays as you like)
- Disadvantage: format is a bit unintuitive / header is quite complex / can't be opened directly in ds9.

Example

Let's look at an example file in this format, the `aeff_P6_v1_diff_back.fits` which represents the Fermi-LAT effective area (an old version) as a function of energy and offset.

It follows the OGIP effective area format.

The data array and axis information are stored in one BINTABLE HDU called "EFFECTIVE AREA", with 5 columns and one row:

```
$ ftlist aeff_P6_v1_diff_back.fits H

        Name                 Type       Dimensions
        ----                 ----       ----------
HDU 1   Primary Array        Null Array
HDU 2   EFFECTIVE AREA       BinTable     5 cols x 1 rows
```

There five columns contain arrays of different length that represent:

- First axis is energy (*ENERG_LO* and *ENERG_HI* columns) with 60 bins.
- Second axis is cosine of theta (*CTHETA_LO* and *CTHETA_HI* columns) with 32 bins.
- First and only data array is effective area (*EFFAREA*) at the given energy and cosine theta values.

```
$ ftlist aeff_P6_v1_diff_back.fits C
HDU 2

  Col  Name                Format[Units](Range)      Comment
    1 ENERG_LO              60E [MeV]
    2 ENERG_HI              60E [MeV]
    3 CTHETA_LO             32E
    4 CTHETA_HI             32E
    5 EFFAREA               1920E [m2]
```

The part that's most difficult to understand / remember is how the relevant information is encoded in the BINTABLE FITS header.

But note the `HDUDOC = 'CAL/GEN/92-019'` key. If you Google *CAL/GEN/92-019* you will find that it points to the OGIP effective area format document. document, which explains in detail what all the other keys mean.

There's some software (e.g. `fv`) that understands this way of encoding n-dimensional arrays and axis information in FITS BINTABLEs.

```
$ ftlist aeff_P6_v1_diff_back.fits[1] K
XTENSION= 'BINTABLE'           / binary table extension
BITPIX  =                    8 / 8-bit bytes
NAXIS   =                    2 / 2-dimensional binary table
NAXIS1  =                 8416 / 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 =                    5 / number of fields in each row
TTYPE1  = 'ENERG_LO'           /
TFORM1  = '60E     '
TTYPE2  = 'ENERG_HI'           /
TFORM2  = '60E     '
TTYPE3  = 'CTHETA_LO'          /
TFORM3  = '32E     '           /
TTYPE4  = 'CTHETA_HI'          /
```

```
TFORM4  = '32E     '           /
TTYPE5  = 'EFFAREA '           /
TFORM5  = '1920E   '
ORIGIN  = 'LISOC   '           / name of organization making this file
DATE    = '2008-05-06T08:56:19.9999' / file creation date (YYYY-MM-DDThh:mm:ss U
EXTNAME = 'EFFECTIVE AREA'     / name of this binary table extension
TUNIT1  = 'MeV     '           /
TUNIT2  = 'MeV     '           /
TUNIT3  = '        '
TUNIT4  = '        '
TUNIT5  = 'm2      '           /
TDIM5   = '(60, 32)'
TELESCOP= 'GLAST   '           /
INSTRUME= 'LAT     '           /
DETNAM  = 'BACK    '
HDUCLASS= 'OGIP    '           /
HDUDOC  = 'CAL/GEN/92-019'     /
HDUCLAS1= 'RESPONSE'           /
HDUCLAS2= 'EFF_AREA'           /
HDUVERS = '1.0.0   '           /
EARVERSN= '1992a   '           /
1CTYP5  = 'ENERGY  '           / Always use log(ENERGY) for interpolation
2CTYP5  = 'COSTHETA'           / Off-axis angle cosine
CREF5   = '(ENERG_LO:ENERG_HI,CTHETA_LO:CTHETA_HI)' /
CSYSNAME= 'XMA_POL '           /
CCLS0001= 'BCF     '           /
CDTP0001= 'DATA    '           /
CCNM0001= 'EFF_AREA'           /
CBD10001= 'VERSION(P6_v1_diff)'
CBD20001= 'CLASS(P6_v1_diff_back)'
CBD30001= 'ENERG(18-560000)MeV'
CBD40001= 'CTHETA(0.2-1)'
CBD50001= 'PHI(0-360)deg'
CBD60001= 'NONE    '
CBD70001= 'NONE    '
CBD80001= 'NONE    '
CBD90001= 'NONE    '
CVSD0001= '2007-01-17'         / Dataset validity start date (UTC)
CVST0001= '00:00:00'           /
CDES0001= 'GLAST LAT effective area' /
EXTVER  =                    1 / auto assigned by template parser
CHECKSUM= 'IpAMIo5LIoALIo5L'   / HDU checksum updated 2008-05-06T08:56:20
DATASUM = '340004495'          / data unit checksum updated 2008-05-06T08:56:20
END
```

1.1.5 HDU classes

Following NASA's recommendation (see HFWG Recommendation R8), a hierarchical classification is applied to each HDU within DL3 FITS files, using the `HDUCLASS` and `HDUCLASn` keywords.

Some useful links from other projects:

- http://cxc.harvard.edu/contrib/arots/fits/content.txt
- https://heasarc.gsfc.nasa.gov/docs/heasarc/ofwg/docs/ofwg_recomm/r8.html
- https://heasarc.gsfc.nasa.gov/docs/heasarc/ofwg/docs/ofwg_recomm/hduclas.html

- http://www.starlink.rl.ac.uk/docs/sun167.htx/sun167se3.html
- https://confluence.slac.stanford.edu/display/ST/LAT+Photons

The different HDUs defined in the current specifications are listed here:

- `EVENTS` : Table containing the event lists. See *EVENTS extension*.
- `GTI` : Table containing the Good Time Intervals ('GTIs') for the event list. See *GTI extension*.
- `POINTING` : Table containing the pointing direction of the telescopes for a number of time stamps. See *POINTING extension*.
- `RESPONSE` : Table containing any of the different instrument response function components defined in the specs. See *IACT IRF*.

The current HDU class scheme used is the following:

- `HDUCLASS` : General identifier of the data format. Recommended value: "GADF" (for "gamma-astro-data-formats")
- `HDUDOC` : Link to the DL3 specifications documentation
- `HDUVERS` : Version of the DL3 specification format
- `HDUCLAS1` : General type of HDU, currently: `EVENTS`, `GTI` or `RESPONSE`
- `HDUCLAS2` : In case of `RESPONSE` type, refers to the IRF components stored within the HDU: `EFF_AREA`, `BKG`, `EDISP` or `RPSF`
- `HDUCLAS3` : In case of `RESPONSE` type, refers to the way the IRF component was produced (`POINT-LIKE` or `FULL-ENCLOSURE`)
- `HDUCLAS4` : In case of `RESPONSE` type, refers to the name of the specific format

HDUCLAS1	HDUCLAS2	HDUCLAS3	HDUCLAS4
EVENTS			
GTI			
POINTING			
RESPONSE	EFF_AREA	POINT-LIKE	AEFF_2D
			AEFF_2D_RECO
		FULL-ENCLOSURE	AEFF_2D
			AEFF_2D_RECO
	EDISP	POINT-LIKE	EDISP_2D
		FULL-ENCLOSURE	EDISP_2D
	RPSF	FULL-ENCLOSURE	PSF_TABLE
			PSF_3GAUSS
			PSF_KING
			PSF_GTPSF
	BKG	POINT-LIKE	BKG_2D
			BKG_3D
		FULL-ENCLOSURE	BKG_2D
			BKG_3D

1.1.6 Notes

Here we collect miscellaneous notes that are helpful when reading or working with the specs.

FITS BINTABLE TFORM data type codes

The valid FITS BINTABLE TFORM data type codes are given in this table in the FITS standard paper in table 18

Information on how to use it correctly via CFITSIO is here

For astropy.io.fits, there's these dicts to translate FITS BINTABLE TFORM codes to Numpy dtype codes:

```
>>> from astropy.io.fits.column import FITS2NUMPY, NUMPY2FITS
>>> FITS2NUMPY
{'J': 'i4', 'I': 'i2', 'L': 'i1', 'E': 'f4', 'M': 'c16', 'B': 'u1', 'K': 'i8', 'C':
↪'c8', 'D': 'f8', 'A': 'a'}
>>> NUMPY2FITS
{'i1': 'L', 'c16': 'M', 'i4': 'J', 'f2': 'E', 'i2': 'I', 'b1': 'L', 'i8': 'K', 'u8':
↪'K', 'u1': 'B', 'u4': 'J', 'u2': 'I', 'c8': 'C', 'f8': 'D', 'f4': 'E', 'a': 'A'}
```

But normally you never should have to manually handle these dtypes from Python. `astropy.io.fits` or `astropy.table.Table` will read and write the TFORM FITS header keys correctly for you.

1.1.7 Glossary

FITS

Flexible Image Transport System http://fits.gsfc.nasa.gov/

HEASARC

High Energy Astrophysics Science Archive Research Centre. http://heasarc.gsfc.nasa.gov/

OGIP FITS Standards

The FITS Working Group in the Office of Guest Investigators Program has established conventions for FITS files for high-energy astrophysics projects. http://hesperia.gsfc.nasa.gov/rhessidatacenter/software/ogip/ogip.html

CALDB

The HEASARC's calibration database (CALDB) system stores and indexes datasets associated with the calibration of high energy astronomical instrumentation. http://heasarc.gsfc.nasa.gov/docs/heasarc/caldb/caldb_intro.html

IACT

IACT = imaging atmospheric Cherenkov telescope (see wikipedia article).

Observation = Run

For IACTs observations are usually conducted by pointing the array (or a sub-array) for a period of time (typically half an hour for current IACTs) at a fixed location in celestial coordinates (i.e. the telescopes slew in horizontal Alt/Az coordinates to keep the pointing position RA/DEC in the center of the field of view).

For current IACTs the term "run" is more common than "observation", but for CTA probably the term "observation" will be used. So it's recommended to use observation in these format specs.

Off Observation

The term “off observation” or “off run” refers to observations where most of the field of view contains no gamma-ray emission (apart from a possible diffuse extragalactic isotropic component, which is supposed to be very weak at TeV energies).

AGN observations are sometimes also considered “off observations”, because the fraction of the field of view containing their gamma-ray emission is often very small, and most of the field of view is empty.

For further info on background modeling see Berge (2007)

1.2 IACT event lists

This document describes the IACT DL3 event list format.

Event lists and associated information are stored in FITS files that contain the following extensions (HDUs):

1.2.1 `EVENTS` extension

The `EVENTS` extension is a binary FITS table that contains an event list. Each row of the table provides information that characterises one event. The mandatory and optional columns of the table are listed below. In addition, a list of header keywords providing metadata is specified. Also here there are mandatory and optional keywords. The recommended extension name of the binary table is `EVENTS`.

Mandatory columns

- **`EVENT_ID` tform: `1K`**
  - Event identification number at the DL3 level (lower data levels could be different, see note below).
- **`TIME` tform: `1D`, unit: s**
  - Time stamp of event in instrument specific MJD time reference. See the header keywords `MJDREFI` and `MJDREFF` for the zero point of the reference time. See also the OGIP event list standard.
- **`RA` tform: `1E`, unit: deg**
  - Reconstructed event Right Ascension (see *RA / DEC*). See also HFWG Recommendation R3 for the OGIP standard.
- **`DEC` tform: `1E`, unit: deg**
  - Reconstructed event Declination (see *RA / DEC*). See also HFWG Recommendation R3 for the OGIP standard.
- **`ENERGY` tform: `1E`, unit: TeV**
  - Reconstructed event energy. See also the OGIP event list standard.

Optional columns

Note: None of the following columns is required to be part of an `EVENTS` extension. Any software **using** these columns should first check whether the columns exist, and warn in case of their absence. Any software **ignoring** these columns should make sure that their presence does not detoriate the functioning of the software.

- **`EVENT_TYPE` tform: `32X`**
  - Event quality partition.
- **`GLON` tform: `1E`, unit: deg**
  - Reconstructed event Galactic longitude (see *Galactic*).
- **`GLAT` tform: `1E`, unit: deg**
  - Reconstructed event Galactic latitude (see *Galactic*).
- **`ALT` tform: `1E`, unit: deg**
  - Reconstructed altitude coordinate of event (horizon system, see *Alt / Az*)
- **`AZ` tform: `1E`, unit: deg**
  - Reconstructed azimuth coordinate of event (horizon system, see *Alt / Az*)
- **`DETX` tform: `1E`, unit: deg**
  - Reconstructed event X-coordinate in detector system (nominal system, see *Field of view*). See also the OGIP event list standard.
- **`DETY` tform: `1E`, unit: deg**
  - Reconstructed event Y-coordinate in detector system (nominal system, see *Field of view*). See also the OGIP event list standard.
- **`THETA` tform: `1E`, unit: deg**
  - Reconstructed offset from the observation pointing position
- **`PHI` tform: `1E`, unit: deg**
  - Reconstructed position angle from the observation pointing position (position angles are counted counterclockwise from celestial North)
- **`MULTIP` tform: `1I`**
  - Telescope multiplicity. Number of telescopes that have seen the event
- **`DIR_ERR` tform: `1E`, unit: deg**
  - Direction error of reconstruction
- **`ENERGY_ERR` tform: `1E`, unit: TeV**
  - Error on reconstructed event energy
- **`COREX` tform: `1E`, unit: m**
  - Reconstructed core position X of shower
- **`COREY` tform: `1E`, unit: m**
  - Reconstructed core position Y of shower
- **`CORE_ERR` tform: `1E`, unit: m**
  - Error on reconstructed core position of shower
- **`XMAX` tform: `1E`, unit: radiation lengths**
  - First interaction depth
- **`XMAX_ERR` tform: `1E`, unit: radiation lengths**
  - Error on first interaction depth

- **`HIL_MSW` tform: `1E`**
  - Hillas mean scaled width
- **`HIL_MSW_ERR` tform: `1E`**
  - Hillas mean scaled width error
- **`HIL_MSL` tform: `1E`**
  - Hillas mean scaled length
- **`HIL_MSL_ERR` tform: `1E`**
  - Hillas mean scaled length error

Mandatory header keywords

- **`HDUCLASS` type: string**
  - Signal conformance with HEASARC/OGIP conventions (option: 'OGIP'). See *HDU classes*.
- **`HDUDOC` type: string**
  - Reference to documentation where data format is documented. See *HDU classes*.
- **`HDUVERS` type: string**
  - Version of the format (e.g. '1.0.0'). See *HDU classes*.
- **`HDUCLAS1` type: string**
  - Primary extension class (option: 'EVENTS'). See *HDU classes*.
- **`ORIGIN` type: string**
  - Organisation that created the FITS file.
- **`CREATOR` type: string**
  - Software that created the file. When appropriate, the value of the `CREATOR` keyword should also reference the specific version of the program that created the FITS file. It is intented that this keyword should refer to the program that originally defined the FITS file structure and wrote the contents. If a FITS file is subsequently copied largely intact into a new FITS by another program, then the value of the `CREATOR` keyword should still refer to the original program. `HISTORY` keywords should be used instead to document any further processing that is performed on the file after it is created. For more reading on the OGIP standard, see here.
- **`TELESCOP` type: string**
  - Telescope (e.g. 'CTA', 'HESS', 'VERITAS', 'MAGIC')
- **`INSTRUME` type: string**
  - Instrument used to aquire the data contained in the file (e.g. 'North', 'South')
- **`OBSERVER` type: string**
  - Name of observer (e.g. 'Joe Public'). This could be for example the PI of a proposal.
- **`OBJECT` type: string**
  - Observed object (e.g. 'Crab')
- **`OBS_MODE` type: string**

  - Observation mode (e.g. 'WOBBLE', 'SCAN', 'SLEW', or any mode that is supported by `TELESCOP`; string should be upper case)
- **`OBS_ID` type: int**
  - Unique observation identifier (Run number)
- **`DATE-OBS` type: string**
  - Start date of observation in ISO standard date representation "ccyy-mm-ddThh:mm:ss" (UTC)
- **`DATE-END` type: string**
  - End date of observation in ISO standard date representation "ccyy-mm-ddThh:mm:ss" (UTC)
- **`TSTART` type: float, unit: s**
  - Start time of observation (given in instrument specific time reference, see below)
- **`TSTOP` type: float, unit: s**
  - End time of observation (given in instrument specific time reference, see below)
- **`MJDREFI` type: int, unit: days**
  - Integer part of instrument specific MJD time reference
- **`MJDREFF` type: float, unit: days**
  - Float part of instrument specific MJD time reference
- **`TIMEUNIT` type: string**
  - Time unit (e.g. 's')
- **`TIMESYS` type: string**
  - Time system, also referred as time scale (e.g. 'UT', 'UTC', 'TT', 'TAI')
- **`TIMEREF` type: string**
  - Time reference frame, used for example for barycentric corrections (options: 'LOCAL', 'SOLARSYSTEM', 'HELIOCENTRIC', 'GEOCENTRIC')
- **`EQUINOX` type: float**
  - Equinox in years for the celestial coordinate system in which positions given in either the header or data are expressed (options: 2000.0). See also HFWG Recommendation R3 for the OGIP standard.
- **`RADECSYS` type: string**
  - Stellar reference frame used for the celestial coordinate system in which positions given in either the header or data are expressed. (options: 'ICRS', 'FK5'). See also HFWG Recommendation R3 for the OGIP standard.
- **`TELAPSE` type: float, unit: s**
  - Time interval between start and stop time (`TELAPSE=TSTOP-TSTART`). Any gaps due to bad weather, or high background counts and/or other anomalies, are included.
- **`ONTIME` type: float, unit: s**
  - Total *good time* (sum of length of all Good Time Intervals). If a Good Time Interval (GTI) table is provided, `ONTIME` should be calculated as the sum of those intervals. Corrections for instrumental *dead time* effects are **NOT** included.
- **`LIVETIME` type: float, unit: s**
  - Total time (in seconds) on source, corrected for the *total* instrumental dead time effect.

- **DEADC type: float**
  - Dead time correction, defined by `LIVETIME/ONTIME`. Is comprised in [0,1]. Defined to be 0 if `ONTIME=0`.
- **EV_CLASS type: str**
  - Event class (the 'cut' that has been used, e.g. 'STD', 'HARD', 'SOFT')'.

Optional header keywords

- **CREATED type: string**
  - Time when file was created in ISO standard date representation "ccyy-mm-ddThh:mm:ss" (UTC)
- **RA_OBJ type: float, unit: deg**
  - Right ascension of `OBJECT`
- **DEC_OBJ type: float, unit: deg**
  - Declination of `OBJECT`

Warning: Keywords below seem to be pretty low-level and eventually instrument specific. It needs to be discussed whether a recommendation on these keywords should be made, or whether the definition should be left to the respective consortia.

- **HDUCLAS2 type: string**
  - Secondary extension class (option: 'ACCEPTED'). See *HDU classes*.
- **TELLIST type: string**
  - Telescope IDs in observation (e.g. '1,2,3,4')
- **N_TELS type: int**
  - Number of observing telescopes
- **TASSIGN type: string**
  - Place of time reference ('Namibia')
- **DST_VER type: string**
  - Version of DST/Data production
- **ANA_VER type: string**
  - Reconstruction software version
- **CAL_VER type: string**
  - Calibration software version
- **CONV_DEP type: float**
  - convergence depth (0 for parallel pointing)
- **CONV_RA type: float, unit: deg**
  - Convergence Right Ascension
- **CONV_DEC type: float, unit: deg**
  - Convergence Declination

- **`TRGRATE` type: float, unit: Hz**
  - Mean system trigger rate
- **`ZTRGRATE` type: float, unit: Hz**
  - Zenith equivalent mean system trigger rate
- **`MUONEFF` type: float**
  - Mean muon efficiency
  - TODO: define how muon efficiency is defined (it's very tricky to get a comparable number in HESS from HD and PA calibration)
- **`BROKPIX` type: float**
  - Fraction of broken pixels (0.15 means 15% broken pixels)
- **`AIRTEMP` type: float, unit: deg C**
  - Mean air temperature at ground during the observation.
- **`PRESSURE` type: float, unit: hPa**
  - Mean air pressure at ground during the observation.
- **`NSBLEVEL` type: float, unit: a.u.**
  - Measure for NSB level
  - TODO: how is this defined? at least leave a comment if it doesn't have a clear definition and can only be used in one chain.
- **`RELHUM` type: float**
  - Relative humidity
  - TODO: link to definition ... wikipedia?

Notes

EVENT_ID column

This paragraph contains some explanatory notes concerning the requirements and recommendations on `EVENT_ID`.

Most analyses with high-level science tools don't need `EVENT_ID` information. But being able to uniquely identify every event is important, e.g. when comparing the high-level reconstructed event parameters (`RA`, `DEC`, `ENERGY`) for different calibrations, reconstructions or gamma-hadron separations.

Assigning a unique `EVENT_ID` during data taking can be difficult or impossible. E.g. in H.E.S.S. we have two numbers `BUNCH_ID_HESS` and `EVENT_ID_HESS` that only together uniquely identify an event within a given run (i.e. `OBS_ID`). Probably the scheme to uniquely identify events at the DL0 level for CTA will be even more complicated, because of the much larger number of telescopes and events.

So given that data taking and event identification is different for every IACT at low data levels and is already fixed for existing IACTs, we propose here to have an `EVENT_ID` that is simpler and works the same for all IACTs at the DL3 level.

As an example: for H.E.S.S. we achieve this by using an INT64 for `EVENT_ID` and to store `EVENT_ID = (BUNCH_ID_HESS << 32) | (EVENT_ID_HESS)`, i.e. use the upper bits to contain the low-level bunch ID and the lower bits to contains the low-level event ID. This encoding is unique and reversible, i.e. it's easy to go back to `BUNCH_ID_HESS` and `EVENT_ID_HESS` for a given `EVENT_ID`, and to low-level checks (e.g. look at the shower images for a given event that behaves strangely in reconstructed high-level parameters).

1.2.2 `GTI` extension

The `GTI` extension is a binary FITS table that contains the Good Time Intervals ('GTIs') for the event list. A general description of GTIs can be found in the OGIP GTI standard.

This HDU contains two mandatory columns named `START` and `STOP`. At least one row is containing the start and end time of the observation must be present. The values are in units of seconds with respect to the reference time defined in the header (keywords MJDREFI and MJDREFF). This extension allows for a detailed handling of good time intervals (i.e. excluding periods with cloud cover or lightning during one observation).

High-level Science tools could modify the GTIs according to user parameter. See e.g. gtmktime for an application example from the Fermi Science Tools.

Mandatory columns

- **`START` tform: `1D`, unit: s**
  - Start time of good time interval (given in instrument specific time reference, see below)
- **`STOP` tform: `1D`, unit: s**
  - End time of good time interval (given in instrument specific time reference, see below)

Mandatory header keywords

- **`MJDREFI` type: int, unit: days**
  - Integer part of instrument specific MJD time reference
- **`MJDREFF` type: float, unit: days**
  - Float part of instrument specific MJD time reference
- **`TIMEUNIT` type: string**
  - Time unit (e.g. 's')
- **`TIMESYS` type: string**
  - Time system, also referred as time scale (e.g. 'UT', 'UTC', 'TT', 'TAI')
- **`TIMEREF` type: string**
  - Time reference frame, used for example for barycentric corrections (options: 'LOCAL', 'SOLARSYSTEM', 'HELIOCENTRIC', 'GEOCENTRIC')

Warning: This is a first draft proposal of a pointing table. Please note that the format is likely subject to change.

1.2.3 `POINTING` extension

The `POINTING` extension is a binary FITS table that contains for a number of time stamps the pointing direction of the telescopes. A *pointing* is here defined as the centre of the field of view (or centre of the camera coordinates). In reality, all telescopes may point to different positions (for example for divergent pointing mode). The main purpose of the `POINTING` extension is to provide time dependent information on how to transform between celestial and terrestial coordinates.

See also HFWG Recommendation R3 for the OGIP standard.

Mandatory columns

- **`TIME` type: float64, unit: s**
  - Time stamp of pointing.
- **`RA_PNT` type: float, unit: deg**
  - Pointing Right Ascension (see *RA / DEC*).
- **`DEC_PNT` type: float, unit: deg**
  - Pointing declination (see *RA / DEC*).
- **`ALT_PNT` type: float, unit: deg**
  - Pointing altitude (see *Alt / Az*).
- **`AZ_PNT` type: float, unit: deg**
  - Pointing azimuth (see *Alt / Az*).

Mandatory header keywords

- **`MJDREFI` type: int, unit: days**
  - Integer part of instrument specific MJD time reference
- **`MJDREFF` type: float, unit: days**
  - Float part of instrument specific MJD time reference
- **`TIMEUNIT` type: string**
  - Time unit (e.g. 's')
- **`TIMESYS` type: string**
  - Time system (e.g. 'TT', 'MJD', 'JD', 'TJD')
- **`TIMEREF` type: string**
  - Time reference frame, used for example for barycentric corrections (options: 'LOCAL', 'SOLARSYSTEM', 'HELIOCENTRIC', 'GEOCENTRIC')
- **`GEOLON` type: float, unit: deg**
  - Geographic longitude of array centre
- **`GEOLAT` type: float, unit: deg**
  - Geographic latitude of array centre
- **`GEOALT` type: float, unit: m**
  - Altitude of array center above sea level

1.3 IACT IRF

The instrument response function (IRF) format currently in use for imaging atmospheric Cherenkov telescopes (IACTs) are stored in FITS binary tables using the "multidimensional array" convention (binary tables with a single row and array columns) described at *BINTABLE HDU*. This format has been used for calibration data and IRF of X-ray instruments, as well as for the IRFs that are distributed with the Fermi-LAT science tools.

Two different approaches are used to store the IRF of IACTs:

- Full-enclosure IRF: all IRF components are stored as a function of the offset with respect to the source position.
- Point-like IRF: IRF components are calculated after applying a cut in direction offset. This format has been used by the current generation of IACTs to perform spectral analysis and light curves.

At the moment (November 2015), this format is used by H.E.S.S., MAGIC and VERITAS and supported by Gammapy and Gammalib and is being proposed for DL3 IRF (i.e. the format distributed to end users and used by the science tools for CTA).

1.3.1 IRF components

The IRF is made up of several components, described here:

Effective Area

Within the IACT community, the effective area has been expressed following two different methods: as a function of the true energy, and as a function of the reconstructed energy. Both have been widely used and documented, and each of them bring certain advantages and disadvantages:

- Using the effective area as a function of true energy, together with the energy dispersion, is the most precise alternative, although it requires enough MC statistics for the energy dispersion noise to be acceptable. In addition, the likelihood fits performed by the science tools are also slower following this approach.
- In the case of using the effective area as a function of reconstructed energy, results may be less precise, but require less MC statistics and allows to perform faster likelihood fits by science tools.

The proposed effective area format, used for both cases mentioned above, follows mostly the OGIP effective area format document.

For the moment, the format for the effective area works to a satisfactory level. Nevertheless, for instance the energy threshold variation across the FoV is not taken into account. However, since the threshold definitions are currently non-unified an inclusion of this variation is still arbitrary and subject to analysis chain. In addition, this feature is currently not supported in current open source tools. We therefore keep the optional opportunity to add an individual extension listing the energy threshold varying across the FoV. This will likely be included in future releases.

Energy Dispersion

The energy dispersion information is stored in a FITS file with one required extensions (HDU). The stored quantity is a PDF for the **energy migration**

$$\mu = \frac{E_{\mathrm{reco}}}{E_{\mathrm{true}}}$$

as a function of true energy and offset. It should be normalized to unity. The migration range covered in the file must be large enough to make this possible (Suggestion: $1/3 < \mu < 3$)

Transformation

For the purpose of some analysis, for example when extracting an *RMF file*, it is necessary to calculate the detector response $R(I, J)$, i.e. the probability to find an energy from within a given true energy bin I of width ΔE_{true} within a certain reconstructed energy bin J of width ΔE_{reco}. In order to do so, the following integration has to be performed (for a fixed offset).

$$R(I,J) = \frac{\int_{\Delta E_{\mathrm{true}}} R(I, E_{\mathrm{true}})\, dE_{\mathrm{true}}}{\Delta E_{\mathrm{true}}},$$

where

$$R(I, E_{\mathrm{true}}) = \int_{\mu(\Delta E_{\mathrm{reco}})} \mathrm{PDF}(E_{\mathrm{true}}, \mu)\, d\mu$$

is the probability to find a given true energy E_{true} in the reconstructed energy band J.

Point spread function

Introduction

The point spread function (PSF) (Wikipedia - PSF) represents the spatial probability distribution of reconstructed event positions for a point source. So far we're only considering radially symmetric PSFs here.

Probability distributions

- $dP/d\Omega(r)$, where dP is the probability to find an event in a solid angle $d\Omega$ at an offset r from the point source. This is the canonical form we use and the values we store in files.
- Often, when comparing observered event distributions with a PSF model, the dP/dr^2 distributions in equal-width bins in r^2 is used. The relation is $d\Omega = \pi dr^2$, i.e. $dP/dr^2 = (1/\pi)(dP/d\Omega)$.
- Sometimes, the distribution $dP/dr(r)$ is used. The relation is $dP/dr = 2\pi r dP/d\Omega$.

TODO: explain "encircled energy" = "encircled counts" = "cumulative" representation of PSF and define containment fraction and containment radius.

Normalisation

PSFs must be normalised to integrate to total probability 1, i.e.

$$\int_0^\infty 2\pi r dP/dr(r) dr = 1, where dP/dr = 2\pi r dP/d\Omega$$

This implies that the PSF producer is responsible for choosing the Theta range and normalising. I.e. it's OK to choose a theta range that contains only 95% of the PSF, and then the integral will be 0.95.

We recommend everyone store PSFs so that truncation is completely negligible, i.e. the containment should be 99% or better for all of parameter space.

Comments

- Usually the PSF is derived from Monte Carlo simulations, but in principle it can be estimated from bright point sources (AGN) as well.
- Tools should assume the PSF is well-sampled and noise-free. I.e. if limited event statistics in the PSF computation is an issue, it is up to the PSF producer to denoise it to an acceptable level.

Background

One method of background modeling for IACTs is to construct spatial and / or spectral model templates of the irreducible cosmic ray background for a given reconstruction and gamma-hadron separation from *Off Observation*. These templates can then be used as an ingredient to model the background in observations that contain gamma-ray emission of interest, or to compute the sensitivity for that set of cuts.

Note: Generating background models requires the construction of several intermediate products (counts and livetime histograms, both filled by cutting out exclusion regions around sources like AGN) to arrive at the models containing an absolute rate described here. At this time we don't specify a format for those intermediate formats.

Note: Background models are sometimes considered an instrument response function (IRF) and sometimes not (e.g. when the background is estimated from different parts of the field of view for the same observation).

Here we have the background format specifications listed under IRFs, simply because the storage format is very similar to the other IRFs (e.g. effective area) and we didn't want to introduce a new top-level section besides IRFs.

1.3.2 IRF axes

Most IRFs are dependent on parameters, and the 1-dimensional parameter arrays are stored in columns. The following names are recommended:

- For energy grids, see here for basic recommendations. Column names should be `ENERGY` or `ENERG_LO`, `ENERG_HI` because that is used (consistently I think) for OGIP and Fermi-LAT. For separate HDUs, the extension names should be `ENERGIES` or `EBOUNDS` (used by Fermi-LAT consistently).
- Sky coordinates should be called `RA`, `DEC`, `GLON`, `GLAT`, `ALT`, `AZ`.
- Field of view coordinates `DETX`, `DETY` or `THETA`, `PHI` for offset and azimuth angle in the field of view.
- Offset wrt. the source position should be called `RAD` (this is what the OGIP PSF formats use).

In the specific case of point-like IRFs:

- The energy-dependent radius of the selected region of interest should be `RAD_MAX`

The IRF format specifications mention a recommended axis format and axis units. But tools should not depend on this and instead:

- Use the axis order specified by the `CREF` header keyword (see *BINTABLE HDU*)
- Use the axis unit specified by the `CUNIT` header keywords (see *BINTABLE HDU*)

1.3.3 Full-enclosure IRFs

Full-enclosure IRF format has been used for calibration data and IRF of X-ray instruments, as well as for the IRFs that are distributed with the Fermi-LAT science tools.

Any full-enclosure IRF component should contain the header keyword:

- `HDU_CLAS3 = FULL-ENCLOSURE`

From here on, the specific format of each IRF component:

Effective area format

Here we specify the format to store the effective area (see *Effective Area*) of a full-enclosure IRF. It is possible to store as a function of the true energy or as a function of the reconstructed energy.

`aeff_2d`

Effective Area vs true energy

The effective area as a function of the true energy and offset angle is saved as a *BINTABLE HDU* with required columns listed below.

Columns:

- **`ENERG_LO`, `ENERG_HI` – ndim: 1, unit: TeV**
    - True energy axis
- **`THETA_LO`, `THETA_HI` – ndim: 1**
    - Field of view offset axis
- **`EFFAREA` – ndim: 2**
    - Effective area value as a function of true energy

Recommended axis order: `ENERGY`, `THETA`

Header keywords:

In addition to the standard header keywords, the recommended energy range for the observation corresponding to the effective area file is stored in two additional header keywords. A hierarchical `HDUCLASS` keyword is used to declare the effective area type contained within the HDU.

- **`OBS_ID` type: int**
    - Observation ID, run number
- **`LO_THRES` type: float, unit: TeV**
    - Low energy threshold
- **`HI_THRES` type: float, unit: TeV**
    - High energy threshold

As explained in *HDU classes*, the following header keyword should be used to declare the type of HDU:

- `HDUDOC` = 'https://github.com/open-gamma-ray-astro/gamma-astro-data-formats'
- `HDUVERS` = '0.2'
- `HDUCLASS` = 'GADF'
- `HDUCLAS1` = 'RESPONSE'
- `HDUCLAS2` = 'EFF_AREA'
- `HDUCLAS3` = 'FULL-ENCLOSURE'
- `HDUCLAS4` = 'AEFF_2D'

The recommended `EXTNAME` keyword is "EFFECTIVE AREA".

Effective Area vs reconstructed energy

The effective area as a function of the reconstructed energy, may be stored as an additional HDU within the FITS file. Note in this case, the `ENERG_LO` and `ENERG_HI` columns contain the reconstructed energy instead of the true energy.

The format is analog to the one described in `aeff_2d`, except for the `HDUCLAS4` keyword:

- `HDUCLAS4` = 'AEFF_2D_RECO'

The `EXTNAME` keyword is recommended to be "EFFECTIVE AREA (RECO)".

Example data file: `here`.

Energy dispersion format

The format to store full-enclosure energy dispersion (see *Energy Dispersion*) is the following:

edisp_2d

The energy dispersion information is saved as a *BINTABLE HDU* with the following required columns.

Columns:

- **`ENERG_LO`, `ENERG_HI` – ndim: 1, unit: TeV**
  - True energy axis
- **`MIGRA_LO`, `MIGRA_HI` – ndim: 1, unit: dimensionless**
  - Energy migration axis (defined above)
- **`THETA_LO`, `THETA_HI` – ndim: 1, unit: deg**
  - Field of view offset axis
- **`MATRIX` – ndim: 3, unit: dimensionless**
  - Energy dispersion $dP/d\mu$, see *Energy Dispersion*.

Recommended axis order: `ENERGY`, `MIGRA`, `THETA`

Header keywords:

As explained in *HDU classes*, the following header keyword should be used to declare the type of HDU:

- `HDUDOC` = 'https://github.com/open-gamma-ray-astro/gamma-astro-data-formats'
- `HDUVERS` = '0.2'
- `HDUCLASS` = 'GADF'
- `HDUCLAS1` = 'RESPONSE'
- `HDUCLAS2` = 'EDISP'
- `HDUCLAS3` = 'FULL-ENCLOSURE'
- `HDUCLAS4` = 'EDISP_2D'

Example data file: TODO

PSF format

IACTs PSF does not always have a Gaussian shape. Its shape is highly dependent on the analysis and each specific instrument. For this reason, several parameterizations are allowed to store the PSF:

`psf_table`

This is a PSF FITS format we agree on for IACTs. This file contains the offset- and energy-dependent table distribution of the PSF.

This format is almost identical to the OGIP radial PSF format. The differences are that we don't have the dependency on azimuthal field of view position, the units are different and the recommended axis order is different (to have uniformity across axis order in the IACT DL3 IRFs).

Columns:

- **`ENERG_LO`, `ENERG_HI` – ndim: 1, unit: TeV**
  - True energy axis
- **`THETA_LO`, `THETA_HI` – ndim: 1, unit: deg**
  - Field of view offset axis
- **`RAD_LO`, `RAD_HI` – ndim: 1, unit: deg**
  - Offset angle from source position
- **`RPSF` – ndim: 3, unit: sr^-1**
  - Point spread function value $dP/d\Omega$, see *Probability distributions*.

Recommended axis order: `ENERGY`, `THETA`, `RAD`.

Header keywords:

As explained in *HDU classes*, the following header keyword should be used to declare the type of HDU:

- `HDUDOC` = 'https://github.com/open-gamma-ray-astro/gamma-astro-data-formats'
- `HDUVERS` = '0.2'
- `HDUCLASS` = 'GADF'
- `HDUCLAS1` = 'RESPONSE'
- `HDUCLAS2` = 'PSF'
- `HDUCLAS3` = 'FULL-ENCLOSURE'
- `HDUCLAS4` = 'PSF_TABLE'

Example data file: TODO

`psf_3gauss`

Multi-Gauss mixture models are a common way to model distributions (for source intensity profiles, PSFs, anything really), see e.g. 2013PASP..125..719H. For H.E.S.S., radial PSFs have been modeled as 1, 2 or 3 two-dimensional Gaussians $dP/d\Omega$.

Note: A two-dimensional Gaussian distribution $dP/d\Omega = dP/(dxdy)$ is equivalent to an exponential distribution in dP/x, where $x = r^2$ and a Rayleigh distribution in dP/dr.

In this format, the triple-Gauss distribution is parameterised as follows:

$$dP/d\Omega(r, S, \sigma_1, A_2, \sigma_2, A_3, \sigma_3) = \frac{S}{\pi}\left[\exp\left(-\frac{r^2}{2\sigma_1^2}\right) + A_2\exp\left(-\frac{r^2}{2\sigma_2^2}\right) + A_3\exp\left(-\frac{r^2}{2\sigma_3^2}\right)\right],$$

where S is `SCALE`, σ_i is `SIGMA_i` and A_i is `AMPL_i` (see columns listed below).

TODO: give analytical formula for the integral, so that it's easy to check if the PSF is normalised for a given set of parameters.

TODO: give test case value and Python function for easy checking?

Note: By setting the amplitudes of the 3rd (and 2nd) Gaussians to 0 one can implement double (or single) Gaussian models as well.

Columns:

- **`ENERG_LO`, `ENERG_HI` – ndim: 1, unit: TeV**
  - True energy axis
- **`THETA_LO`, `THETA_HI` – ndim: 1, unit: deg**
  - Field of view offset axis
- **`SCALE` – ndim: 2, unit: none**
  - Absolute scale of the 1st Gaussian
- **`SIGMA_1`, `SIGMA_2`, `SIGMA_3` – ndim: 2, unit: deg**
  - Model parameter (see formula above)
- **`AMPL_2`, `AMPL_3` – ndim: 2, unit: none**
  - Model parameter (see formula above)

Recommended axis order: `ENERGY`, `THETA`

Header keywords:

As explained in *HDU classes*, the following header keyword should be used to declare the type of HDU:

- `HDUDOC` = 'https://github.com/open-gamma-ray-astro/gamma-astro-data-formats'
- `HDUVERS` = '0.2'
- `HDUCLASS` = 'GADF'
- `HDUCLAS1` = 'RESPONSE'
- `HDUCLAS2` = 'PSF'
- `HDUCLAS3` = 'FULL-ENCLOSURE'
- `HDUCLAS4` = 'PSF_3GAUSS'

Example data file: TODO

psf_king

The King function parametrisations for PSFs has been in use in astronomy as an analytical PSF model for many instruments, for example by the Fermi-LAT (see 2013ApJ...765...54A).

The distribution has two parameters `GAMMA` γ and `SIGMA` σ and is given by the following formula:

$$dP/d\Omega(r,\sigma,\gamma) = \frac{1}{2\pi\sigma^2}\left(1-\frac{1}{\gamma}\right)\left(1+\frac{r^2}{2\gamma\sigma^2}\right)^{-\gamma}$$

This formula integrates to 1 (see *Introduction*).

Columns:

- **`ENERG_LO`, `ENERG_HI` – ndim: 1, unit: TeV**
  - True energy axis
- **`THETA_LO`, `THETA_HI` – ndim: 1, unit: deg**
  - Field of view offset axis
- **`GAMMA` – ndim: 2, unit: none**
  - Model parameter (see formula above)
- **`SIGMA` – ndim: 2, unit: deg**
  - Model parameter (see formula above)

Recommended axis order: `ENERGY`, `THETA`

Header keywords:

As explained in *HDU classes*, the following header keyword should be used to declare the type of HDU:

- `HDUDOC` = 'https://github.com/open-gamma-ray-astro/gamma-astro-data-formats'
- `HDUVERS` = '0.2'
- `HDUCLASS` = 'GADF'
- `HDUCLAS1` = 'RESPONSE'
- `HDUCLAS2` = 'PSF'
- `HDUCLAS3` = 'FULL-ENCLOSURE'
- `HDUCLAS4` = 'PSF_KING'

Example data file: TODO

psf_gtpsf

The FITS file has the following BinTable HDUs / columns:

- PRIMARY HDU – empty
- **PSF HDU**
  - Energy – 1D (MeV)
  - Exposure – 1D (cm^2 s)
  - Psf – 2D (sr^-1), shape = (len(Energy) x len(Theta)) Point spread function value $dP/d\Omega$, see *Probability distributions*.

- **THETA HDU**
  - Theta – 1D (deg)

Header keywords:

As explained in *HDU classes*, the following header keyword should be used to declare the type of HDU:

- `HDUDOC` = 'https://github.com/open-gamma-ray-astro/gamma-astro-data-formats'
- `HDUVERS` = '0.2'
- `HDUCLASS` = 'GADF'
- `HDUCLAS1` = 'RESPONSE'
- `HDUCLAS2` = 'PSF'
- `HDUCLAS3` = 'FULL-ENCLOSURE'
- `HDUCLAS4` = 'GTPSF'

Example data file: `psf-fermi.fits`

Background format

Here we specify two formats for the background template models (see *Background*) of a full-enclosure IRF:

- `bkg_2d` models depend on `ENERGY` and `THETA`, i.e. are radially symmetric.
- `bkg_3d` models depend on `ENERGY` and field of view coordinates `DETX` and `DETY`.

`bkg_2d`

The `bkg_2d` format contains a 2-dimensional array of post-select background rate, stored in the *BINTABLE HDU* format.

Required columns:

- **`ENERG_LO`, `ENERG_HI` – ndim: 1, unit: TeV**
  - Reconstructed energy axis
- **`THETA_LO`, `THETA_HI` – ndim: 1, unit: deg**
  - Field of view offset axis
- **`BKG` – ndim: 2, unit: s^-1 MeV^-1 sr^-1**
  - Absolute post-select background rate (expected background per time, energy and solid angle).

Recommended axis order: `ENERGY`, `THETA`

Header keywords:

As explained in *HDU classes*, the following header keyword should be used to declare the type of HDU:

- `HDUDOC` = 'https://github.com/open-gamma-ray-astro/gamma-astro-data-formats'
- `HDUVERS` = '0.2'
- `HDUCLASS` = 'GADF'
- `HDUCLAS1` = 'RESPONSE'
- `HDUCLAS2` = 'BKG'

- HDUCLAS3 = 'FULL-ENCLOSURE'
- HDUCLAS4 = 'BKG_2D'

Example data file: here.

bkg_3d

The bkg_3d format contains a 3-dimensional array of post-select background rate, stored in the *BINTABLE HDU* format.

Required columns:

- **ENERG_LO, ENERG_HI – ndim: 1, unit: TeV**
  - Reconstructed energy axis
- **DETX_LO, DETX_HI, DETY_LO, DETY_HI – ndim: 1, unit: deg**
  - Field of view coordinates binning, see *Field of view*
- **BKG – ndim: 3, unit: s^-1 MeV^-1 sr^-1**
  - Absolute post-select background rate (expected background per time, energy and solid angle).

Recommended axis order: ENERGY, DETX, DETY

Header keywords:

As explained in *HDU classes*, the following header keyword should be used to declare the type of HDU:

- HDUDOC = 'https://github.com/open-gamma-ray-astro/gamma-astro-data-formats'
- HDUVERS = '0.2'
- HDUCLASS = 'GADF'
- HDUCLAS1 = 'RESPONSE'
- HDUCLAS2 = 'BKG'
- HDUCLAS3 = 'FULL-ENCLOSURE'
- HDUCLAS4 = 'BKG_3D'

Example data file: here.

Notes

The background rate is not a "flux" or "surface brightness". It is already a count rate, it doesn't need to be multiplied with effective area to obtain predicted counts, like gamma-ray flux and surface brightness models do. The rate is computed per observation time (without any dead-time correction, don't use livetime when computing or using the background rate).

1.3.4 Point-like IRFs

Point-like IRFs has been classically used within the IACT community. Each IRF component is calculated from the events surviving an energy dependent directional cut around the assumed source position.

The format of each point-like IRF component is analog to the ones already described within the full enclosure IRF specifications (see *Full-enclosure IRFs*), with certain differences listed in this section.

Any point-like IRF component should contain the header keyword:

- `HDU_CLAS3 = POINT-LIKE`

RAD_MAX

In addition to the IRFs, the actual directional cut applied to the data needs to be stored. This cut is allowed to be constant or variable along several axes, with a different format.

In case the angular cut is constant along the energy and FoV, an additional header keyword may be added to the IRF HDU:

Header keyword:

- **`RAD_MAX` type: float, unit: deg**
  - Radius of the directional cut applied to calculate the IRF, in degrees.

If this keyword is present, the science tools should assume the directional cut of this point-like IRF is constant over all axes. In case the angular cut is variable along any axis (reconstructed energy or FoV), an additional HDU is required to store these values. Note any DL3 file with a point-like IRF (with `HDU_CLAS3` = POINT-LIKE) that has no `RAD_MAX` keyword within the HDU metadata should have this additional HDU.

In case the directional cut is variable with energy or the FoV, point-like IRFs require an additional binary table. It stores the values of `RAD_MAX` as a function of the reconstructed energy and FoV following the *BINTABLE HDU* format.

rad_max_2d

The `rad_max_2d` format contains a 2-dimensional array of directional cut values, stored in the *BINTABLE HDU* format.

Required columns:

- **`ENERG_LO`, `ENERG_HI` – ndim: 1, unit: TeV**
  - Reconstructed energy axis
- **`THETA_LO`, `THETA_HI` – ndim: 1, unit: deg**
  - Field of view offset axis
- **`RAD_MAX` – ndim: 2, unit: deg**
  - Radius of the directional cut applied to calculate the IRF, in degrees.

Recommended axis order: `ENERGY`, `THETA`, `RAD_MAX`

Header keywords:

As explained in *HDU classes*, the following header keyword should be used to declare the type of HDU:

- `HDUDOC` = ‘https://github.com/open-gamma-ray-astro/gamma-astro-data-formats‘
- `HDUVERS` = ‘0.2’
- `HDUCLASS` = ‘GADF’
- `HDUCLAS1` = ‘RESPONSE’
- `HDUCLAS2` = ‘RAD_MAX’
- `HDUCLAS3` = ‘POINT-LIKE’

- `HDUCLAS4` = 'RAD_MAX_2D'

Example data file: `here`.

1.4 IACT data storage

Here we give an overview of how the data storage for IACTs should look like. In general, IACT data is divided into "runs" of a few tens of minutes of data taking. We have per-run IRFs and per run event lists. The challenge is that in the data structure, we have to accomodate user choices on various levels:

- Reconstruction chain (e.g. paris)
- Version of FITS exporter (e.g. prod01)
- Version of internal data storage (e.g. model_deconvoluted_prod26)
- Cut configuration (e.g. mpp_std)

We here propose a two-level index file scheme to allow for arbitrary folder structures. For each directory tree containing the files of the cut configuration, two files should be present:

1.4.1 Observation index table

The observation index table is stored in a FITS file as a BINTABLE HDU:

- Suggested filename: `obs-index.fits.gz`
- Suggested HDU name: `OBS_INDEX`

It contains one row per observation (a.k.a. run) and lists parameters that are commonly used for observation selection, grouping and analysis.

Required columns

- **`OBS_ID` type: int**
  - Unique observation identifier (Run number)
- **`RA_PNT` type: float, unit: deg**
  - Obsevation pointing right ascension (see *RA / DEC*)
- **`DEC_PNT` type: float, unit: deg**
  - Observation pointing declination (see *RA / DEC*)
- **`ZEN_PNT` type: float, unit: deg**
  - Observation pointing zenith angle at observation mid-time `TMID` (see *Alt / Az*)
- **`ALT_PNT` float, deg**
  - Observation pointing altitude at observation mid-time `TMID` (see *Alt / Az*)
- **`AZ_PNT` type: float, unit: deg**
  - Observation pointing azimuth at observation mid-time `TMID` (see *Alt / Az*)
- **`ONTIME` type: float, unit: s**
  - Total observation time (including dead time).

  - Equals `TSTOP` - `TSTART`
- `LIVETIME` **type: float, unit: s**
  - Total livetime (observation minus dead time)
- `DEADC` **type: float**
  - Dead time correction.
  - It is defined such that `LIVETIME` = `DEADC` * `ONTIME` i.e. the fraction of time the telescope was actually able to take data.
- `TSTART` **type: float, unit: s**
  - Start time of observation relative to the reference time
- `TSTOP` **type: float, unit: s**
  - End time of observation relative to the reference time
- `DATE_OBS` **type: string**
  - Start date of observation in UTC string format: "YYYY-MM-DD"
- `TIME_OBS` **type: string**
  - Start time of observation in UTC string format: "HH:MM:SS"
- `DATE_END` **type: string**
  - End date of observation in UTC string format: "YYYY-MM-DD"
- `TIME_END` **type: string**
  - End time of observation in UTC string format: "HH:MM:SS"
- `N_TELS` **type: int**
  - Number of participating telescopes
- `TELLIST` **type: string**
  - Telescope IDs (e.g. '1,2,3,4')
- `QUALITY` **type: int**
  - **Observation data quality. The following levels of data quality are defined:**
    * -1 = unknown quality
    * 0 = best quality, suitable for spectral analysis.
    * 1 = medium quality, suitable for detection, but not spectra (typically if the atmosphere was hazy).
    * 2 = bad quality, usually not to be used for analysis.

Optional columns

The following columns are optional. They are sometimes used for observation selection or data quality checks or analysis, but aren't needed for most users.

- `OBJECT` **type: string**
  - Primary target of the observation
  - **Recommendations:**

    * Use a name that can be resolved by SESAME
    * For survey observations, use "survey".
    * For *Off Observation*, use "off observation"

- **`GLON_PNT` type: float, unit: deg**
  - Observation pointing Galactic longitude (see *Galactic*).
- **`GLAT_PNT` type: float, unit: deg**
  - Observation pointing Galactic latitude (see *Galactic*).
- **`RA_OBJ` type: float, unit: deg**
  - Right ascension of `OBJECT`
- **`DEC_OBJ` type: float, unit: deg**
  - Declination of `OBJECT`
- **`TMID` type: float, unit: s**
  - Mid time of observation relative to the reference time
- **`TMID_STR` type: string**
  - Mid time of observation in UTC string format: "YYYY-MM-DD HH:MM:SS"
- **`EVENT_COUNT` type: int**
  - Number of events in run
- **`EVENT_RA_MEDIAN` type: float, unit: deg**
  - Median right ascension of events
- **`EVENT_DEC_MEDIAN` type: float, unit: deg**
  - Median declination of events
- **`EVENT_ENERGY_MEDIAN` type: float, unit: deg**
  - Median energy of events
- **`EVENT_TIME_MIN` type: double, unit: s**
  - First event time
- **`EVENT_TIME_MAX` type: double, unit: s**
  - Last event time
- **`BKG_SCALE` type: float**
  - Observation-dependent background scaling factor. See notes below.
- **`TRGRATE` type: float, unit: Hz**
  - Mean system trigger rate
- **`ZTRGRATE` type: float, unit: Hz**
  - Zenith equivalent mean system trigger rate
  - Some HESS chains export this at the moment and this quantity can be useful for data selection. Comparing values from different chains or other telescopes would require a more specific specification.
- **`MUONEFF` type: float**

- Mean muon efficiency
- Currently use definitions from analysis chain, since creating a unified specification is tricky.

- **BROKPIX type: float**
  - Fraction of broken pixels (0.15 means 15% broken pixels)
- **AIRTEMP type: float, unit: deg C**
  - Mean air temperature at ground during the observation.
- **PRESSURE type: float, unit: hPa**
  - Mean air pressure at ground during the observation.
- **NSBLEVEL type: float, unit: a.u.**
  - Measure for NSB level
  - Some HESS chains export this at the moment and this quantity can be useful for data selection. Comparing values from different chains or other telescopes would require a more specific specification.
- **RELHUM type: float**
  - Relative humidity
  - Definition

Mandatory Header keywords

- **MJDREFI type: int, unit: days**
  - Integer part of instrument specific MJD time reference
- **MJDREFF type: float, unit: days**
  - Float part of instrument specific MJD time reference
- **TIMEUNIT type: string**
  - Time unit (e.g. 's')
- **TIMESYS type: string**
  - Time system (e.g. 'TT', 'UTC')

Notes

- Some of the required columns are redundant. E.g. ONTIME = TSTOP - TSTART. The motivation to declare those columns required is to make it easy for users and tools to browse the observation lists and select observations via cuts on these parameters without having to compute them on the fly.
- Observation runs where the telescopes don't point to a fixed RA / DEC position (e.g. drift scan runs) aren't supported at the moment by this format.
- Purpose / definition of BKG_SCALE: For a 3D likelihood analysis a good estimate of the background is important. The run-by-run varation of the background rate is ~20%. The main reasons are the changing atmospheric conditions. This parameter allows to specify (from separate studies) a scaling factor to the *Background* This factor comes e.g. from the analysis of off runs. The background normalisation usually dependends on e.g. the number of events in a run, the zenith angle and other parameters. This parameter provides the possibility to give the user a better prediction of the background normalisation. For CTA this might be induced from atmospheric monitoring and additional diagnostic input. For HESS we try to find a trend in the off run background normalisations and other parameters such as number of events per unit livetime. The background scale should be around

1.0 if the background model is good. This number should also be set to 1.0 if no dependency analysis has been performed. If the background model normalisation is off by a few orders of magnitude for some reasons, this can also be incorporated here.

1.4.2 HDU index table

The HDU index table is stored in a FITS file as a BINTABLE HDU:

- Suggested filename: `hdu-index.fits.gz`
- Suggested HDU name: `HDU_INDEX`

The HDU index table can be used to locate HDUs. E.g. for a given `OBS_ID` and (`HDU_TYPE` and / or `HDU_CLASS`), the HDU can be located via the information in the `FILE_DIR`, `FILE_NAME` and `HDU_NAME` columns. The path listed in `FILE_DIR` has to be relative to the location of the index file.

TODO: discuss if we want to support a `BASE_DIR` header keyword, to allow the use case where `FILE_DIR` is not relative to the index file location (e.g. in cases where the user creates or modifies the index file and doesn't have write permission in the folder where the data files are.)

Columns

Column Name	Description	Data type	Required?
`OBS_ID`	Observation ID (a.k.a. run number)	int	yes
`HDU_TYPE`	HDU type (see below)	string	yes
`HDU_CLASS`	HDU class (see below)	string	yes
`FILE_DIR`	Directory of file (rel. to this file)	string	yes
`FILE_NAME`	Name of file	string	yes
`HDU_NAME`	Name of HDU in file	string	yes
`SIZE`	File size (bytes)	int	no
`MTIME`	Modification time	double	no
`MD5`	Checksum	string	no

HDU_TYPE and HDU_CLASS

The `HDU_TYPE` and `HDU_CLASS` can be used to select the HDU of interest.

The difference is that `HDU_TYPE` corresponds generally to e.g. PSF, whereas `HDU_CLASS` corresponds to a specific PSF format. Declaring `HDU_CLASS` here means that tools loading these files don't have to do guesswork to infer the format on load.

Valid `HDU_TYPE` values (others optional):

- `events` - Event list
- `gti` - Good time interval
- `aeff` - Effective area
- `psf` - Point spread function
- `edisp` - Energy dispersion
- `bkg` - Background

(can be optional, e.g. if no bkg model is available another approach has to be used)

Valid `HDU_CLASS` values:

- `events` - see format spec: *EVENTS extension*
- `gti` - see format spec: *GTI extension*
- `aeff_2d` - see format spec: *aeff_2d*
- `edisp_2d` - see format spec: *edisp_2d*
- `psf_table` - see format spec: *psf_table*
- `psf_3gauss` - see format spec: *psf_3gauss*
- `psf_king` - see format spec: *psf_king*
- `bkg_2d` - see format spec: *bkg_2d*
- `bkg_3d` - see format spec: *bkg_3d*

We recommend that HDU names are chosen to be identical to either the `HDU_TYPE` or the `HDU_CLASS` names mentioned above. This is not a requirement, usually end users will access data via HDU index files and the HDU names don't matter. Or, if HDUs are accessed directly, the package or tool should be flexible to allow loading the HDU with any name.

The observation index provides information of meta data about each observation run. E.g. pointing in the sky, duration, number of events, etc. The hdu index file provides a list of all available HDUs and in what files they are located. Science tools can make use of this index files to build filenames of required files according to some user parameters.

In addition, we have an index of all available index files to simply allow a quick look on what configurations are available. This file also provides the locations of the hdu index and observation index files.

1.4.3 Master index file

Warning: We are currently in a prototyping phase. This format is under development.

The idea is to have an index file containing and listing the locations to all further hdu index files. To allow for human-readable and human-editable files, we use a `JSON` format here

- Required filename: `master.json`

The user copies this file from the server along with selected data. The Science tools that access this file just ignore chains/configs that are not present on the users' machine. Ideally, the Science tools provide the possibility to inspect the local master index file and print the users' options on the screen. Since all paths must be relative to the location of the master index file, the user doesn't have to edit and maintain the master index file. The Science tools naturally will allow the analysis of a certain chain/config or not. Of course the user can always add own FITS productions etc simply by hand (or locally change names of configs for convenience). The `JSON` table should contain an array named `datasets`. Each dataset is specified by the following required keys:

Required keys

- **`name` type: string**
  - Unique name describing the present FITS production, e.g."hess-hap-hd-prod01-std_zeta_fullEnclosure".
- **`hduindx` type: string**
  - Location of corresponding hdu index file. This path must be relative to the location of the master index file

- **`obsindx` type: string**
  - Location of corresponding observation index file. This path must be relative to the location of the master index file

Of course any optional and additional information can be added, e.g. the telescope name, analysis chain, cut configuration, etc. The Science tools should be able to show these information to the user to simplify the choice for a preferred FITS production.

Here is an example of the master index file:

```
{
    "datasets": [
        {
            "name": "fits-prod1-stdcuts",
                "obsindx": "relative/path/to/prod1-std/obs-index.fits.gz",
            "hduindx": "relative/path/to/prod1-std/hdu-index.fits.gz",
                "comment": "First test version",
                "drawback": "Not all data available"
        },
        {
                "name": "fits-prod2-hardcuts",
                "obsindx": "relative/path/to/prod2-hard/obs-index.fits.gz",
            "hduindx": "relative/path/to/prod2-hard/hdu-index.fits.gz",
                "recommendation:": "use for science publications"
        }
    ]
}
```

Note that the keywords “comment”, “drawback” and “recommendation” are arbitray and can be added from the user or maintainer of the master index file. The Science tools can display them for the user to get more details about each FITS dataset on the users’ machine.

1.5 OGIP 1D spectrum data formats

The *EVENTS extension* and 2D *IACT IRF* can be transformed into a 1D counts vector and 1D IRFs that can serve as input to general X-ray spectral analysis packages such as Sherpa. For an introduction to this so-called OGIP data format please refer to the official Documentation on *HEASARC*.

The following section only highlight differences and modifications made to the OGIP standard in order to meet the requirements of gamma-astronomy.

1.5.1 PHA file

The OGIP standard PHA file format is described here.

TODO: Should an EBOUNDS extension be added to the PHA file (channels -> energy)? In OGIP this info has to be extraced from the RMF file.

The values of following header keywords need some attention when using them for *IACT* analysis.

- **`BACKSCAL`**
  - For now it is assumed that exposure ration between signal and background counts does not depend on energy, thus this keyword must be set

   - The BACKSCAL keywords in the PHA and the BKG file must be set so that

$$\alpha = \frac{\mathrm{PHA}_{\mathrm{backscal}}}{\mathrm{BKG}_{\mathrm{backscal}}}$$

   - It is recommended to set the `BACKSCAL` keyword to 1 in the PHA file and to $1/\alpha$ in the BKG file

Additional header keyword that can be stored in the PHA header for *IACT* analysis are listed below.

- **`OFFSET` type: float, unit deg**
   - Distance between observation position and target of a spectral analysis
- **`MUONEFF` type: float**
   - Muon efficiency of the observation
- **`ZENITH` type: tbd, unit: deg**
   - Zenith angle of the observation
- **`ON-RAD` type: float, unit deg**
   - Radius of the spectral extraction region
   - Defines the spectral extraction region together with the standard keywords `RA-OBJ` and `DEC-OBJ`

1.5.2 BKG file

The values of following header keywords need some attention when using them for *IACT* analysis.

- **`BACKSCAL`**
   - It is recommended to set the `BACKSCAL` keyword to $1/\alpha$ in the BKG file (see above)

1.5.3 ARF file

The OGIP standard ARF file format is described here.

Additional header keyword that can be stored in the ARF header for *IACT* analysis are listed below.

- **`LO_THRES` type: tbd, unit: TeV**
   - Low energy threshold of the analysis
- **`HI_THRES` type: tbd, unit: TeV**
   - High energy threshold of the analysis

1.5.4 RMF file

The OGIP standard RMF file format is described here.

How an RMF file can be extracted from a IACT 2D energy dispersion file is explained in *Energy Dispersion*.

1.6 Sky Maps

This page describes data format conventions for FITS binned data and model representations pixelized with the HEALPix algorithm.

1.6.1 HEALPix Formats

This section describes a proposal for HEALPix format conventions which is based on formats currently used within the Fermi Science Tools (STs) and pointlike. This format is intended for representing maps and cubes of both integral and differential quantities including:

- Photon count maps and cubes (e.g. as generated with with *gtbin*).
- Exposure cubes (e.g. as generated with *gtexpcube2*).
- Source maps – product of exposure with instrument response in spatial dimension (e.g. as generated with *gtsrcmaps*).
- Model maps and cubes (the Fermi IEM and other diffuse emission components).

The format defines a *SKYMAP HDU* for storing a sequence of image slices (*bands*) and a *BANDS HDU* to store the geometry and coordinate mapping for each band. A band can represent any selection on non-spatial coordinates such as energy, time, or FoV angle. The most common use-case is a sequence of bands representing energy bins (for counts maps) or energy nodes (for source or model maps).

There are three primary HEALPix map formats which use different table structures for mapping table entries to HEALPix pixel and band:

- *IMPLICIT Format*: The row to pixel mapping is determined implicitly from the row number. The row number corresponds to the HEALPix pixel number.
- *EXPLICIT Format*: The row to pixel mapping is determined explicitly from the `PIX` column. This can be used to define maps or cubes that only encompass a partial region of the sky.
- *LOCAL Format*: The row to pixel mapping is determined explicitly from the `PIX` column but with a local indexing scheme. This can be used to define maps or cubes that only encompass a partial region of the sky.
- *SPARSE Format*: The row to pixel mapping is determined explicitly from the `PIX` column but with a variable number of pixels in each band. This format can be used to represent maps that have a different spatial geometry in each band and also supports band-dependent pixel size.

Note that there are variations of these primary formats which use different conventions for column, HDU, and header keywords names. The `HPX_CONV` keyword defines a specific mapping between the names used here and in these other formats:

- `FGST_CCUBE`
- `FGST_LTCUBE`
- `FGST_BEXPCUBE`
- `FGST_SRCMAP`
- `FGST_TEMPLATE`
- `FGST_SRCMAP_SPARSE`
- `GALPROP`
- `GALPROP2`

Non-Standard HEALPix Conventions

Sample Files

- All-sky Counts Cube (IMPLICIT Format): `FITS`
- Partial-sky Counts Cube (EXPLICIT Format): `FITS`

- Partial-sky Counts Map (EXPLICIT Format): `FITS`
- Partial-sky Counts Cube (SPARSE Format w/ fixed NSIDE):`FITS`
- Partial-sky Counts Cube (SPARSE Format w/ variable NSIDE):`FITS`

SKYMAP HDU

The SKYMAP table contains the map data and row-to-pixel mapping formatted according to one of three indexing schemes specified by the `INDXSCHM` header keyword: *IMPLICIT Format*, *EXPLICIT Format*, or *SPARSE Format*. By convention if a file contains a single map it is recommended to name the extension `SKYMAP`. For maps with non-spatial dimensions an accompanying BANDS table must also be defined (see *BANDS HDU*).

Header Keywords

This section lists the keywords that should be written to the SKYMAP BINTABLE header. These keywords define the pixel size and ordering scheme that was used to construct the HEALPix map.

- **`PIXTYPE`, type: string**
  - Should be set to `HEALPIX`.
- **`INDXSCHM`, type: string**
  - Indexing scheme. Can be one of `IMPLICIT`, `EXPLICIT`, `SPARSE`. If this keyword is not provided then the `IMPLICIT` indexing scheme will be assumed.
- **`ORDERING`, type: string**
  - HEALPix ordering scheme. Can be `NESTED` or `RING`.
- **`COORDSYS`, type: string**
  - Map coordinate system. Can be `CEL` (celestial coordinates) or `GAL` (galactic coordinates).
- **`ORDER`, type: int**
  - Healpix order. `ORDER` = log2(`NSIDE`) if `NSIDE` is a power of 2 and -1 otherwise. If the `BANDS` table is defined this keyword is superseded by the `NSIDE` column.
- **`NSIDE`, type: int**
  - Number of healpix pixels per side. If the `BANDS` table is defined this keyword is superseded by the `NSIDE` column.
- **`FIRSTPIX`, type: int**
  - Index of first pixel in the map.
- **`LASTPIX`, type: int**
  - Index of last pixel in the map.
- **`HPX_REG`, type: string, optional**
  - Region string for the geometric selection that was used to construct a partial-sky map. See *HEALPix Region String* for additional details.
- **`HPX_CONV`, type: string, optional**
  - Convention for HEALPix format. See *Non-Standard HEALPix Conventions* for additional details.
- **`BANDSHDU`, type: string, optional**

    – Name of HDU containing the BANDS table. If undefined the extension name should be `EBOUNDS` or `ENERGIES`. See *BANDS HDU* for additional details.

HEALPix Region String

For partial-sky maps that correspond to a particular geometric selection one can optionally specify the selection that was used in constructing the map with the `HPX_REG` header keyword. The following region strings are supported:

- `DISK({LON},{LAT},{RADIUS})`: A circular selection centered on the coordinates (`{LON}`, `{LAT}`) with radius `{RADIUS}` with all quantities given in degrees. A pixel is included in the selection if its center is within `{RADIUS}` deg of coordinates (`{LON}`, `{LAT}`).
- `DISK_INC({LON},{LAT},{RADIUS})`: A circular selection centered on the coordinates (`{LON}`, `{LAT}`) with radius `{RADIUS}` with all quantities given in degrees. A pixel is included in the selection if any part of it is encompassed by the circle.
- `HPX_PIXEL({ORDERING},{ORDER},{PIX})`: A selection encompassing all pixels contained in the HEALPix pixel of the given pixelization where ordering is `{ORDERING}` (i.e. `NESTED` or `RING`), order is `{ORDER}`, and pixel index is `{PIX}`.

IMPLICIT Format

The IMPLICIT format defines a one-to-one mapping between table row and HEALPix pixel index. Each energy plane is represented by a separate column (`CHANNEL0`, `CHANNEL1`, etc.). This format can only be used for all-sky maps.

HEADER

- `INDXSCHM`:`IMPLICIT`

SKYMAP Columns

- **`CHANNEL{BAND_IDX}` – ndim: 1, type: float or int**
    – Dimension: nrows
    – Amplitude in image plane `{BAND_IDX}`. The HEALPix pixel index is determined from the table row.

EXPLICIT Format

The EXPLICIT format uses an additional `PIX` column to explicitly define the pixel number corresponding to each table row. Pixel values for each band are represented by a separate column (`CHANNEL0`, `CHANNEL1`, etc.). This format can be used for both all-sky and partial-sky maps but requires the same pixel size for all bands.

HEADER

- `INDXSCHM`:`EXPLICIT`

SKYMAP Columns

- **`PIX` – ndim: 1, unit: None, type: int**
  - Dimension: nrows
  - HEALPix pixel index. This index is common to all bands.
- **`CHANNEL{BAND_IDX}` – ndim: 1, type: float or int**
  - Dimension: nrows
  - Amplitude in HEALPix pixel `PIX` and band `{BAND_IDX}`.

LOCAL Format

The LOCAL format is identical to the *EXPLICIT Format* with the exception that the `PIX` column contains a local rather than global HEALPix index. The local HEALPix index is a mapping of global indices in a partial-sky geometry to 0, ..., N-1 where N is the total number of pixels in the geometry. For an all-sky geometry the local index is equal to the global index. This format can be used for both all-sky and partial-sky maps as well as maps with a different pixel-size in each band.

HEADER

- `INDXSCHM : LOCAL`

SKYMAP Columns

- **`PIX` – ndim: 1, unit: None, type: int**
  - Dimension: nrows
  - Local HEALPix pixel index. The mapping to global HEALPix index is derived by finding the index of the ith pixel in the geometry where pixels are ordered by their global index values.
- **`CHANNEL{BAND_IDX}` – ndim: 1, type: float or int**
  - Dimension: nrows
  - Amplitude in HEALPix pixel `PIX` and band `{BAND_IDX}`.

SPARSE Format

The SPARSE format allows for an arbitrary set of pixels to be defined in each band. The SKYMAP table contains three columns with the pixel index, band index, and amplitude. Pixel values for each band are continguous and arranged in order of band index. This format supports an different HEALPix pixel size in each band defined by the `NSIDE` column in the `BANDS` table.

Pixels that are undefined but contained within the geometric selection are assumed to be zero while pixels outside the geometric selection are undefined. For counts data this allows for maps that only contain pixels with at least one count.

HEADER

- `INDXSCHM : SPARSE`

SKYMAP Columns

- **`PIX` – ndim: 1, unit: None, type: int**
  - Dimension: nrows
  - HEALPix pixel index. Pixels are ordered by band number. The row to band mapping is defined by the `CHANNEL` column. The column type may be either 32- or 64-bit depending on the maximum index of the map geometry. A 32-bit column type is sufficient for maps with NSIDE as large as 8192.
- **`CHANNEL` – ndim: 1, unit: None, type: int**
  - Dimension: nrows
  - Band index. This column is optional for maps with a single band. For efficiency it is recommended to represent this column with a 16-bit integer.
- **`VALUE` – ndim: 1, unit: None, type: float or int**
  - Dimension: nrows
  - Amplitude in pixel indexed by `PIX` and `CHANNEL`.

1.6.2 WCS Formats

This page describes data format conventions for images and cubes pixelized with World Coordinate Systems (WCS). WCS is a system for describing transformations between pixel/world coordinates and sky coordinates. The conventions described here are extensions to the FITS WCS standard which allow for serializing more complex data structures such as sparse maps or maps with non-regular geometry (e.g. a different pixel size in each image plane).

The format splits the specification of an image into two HDUs: an *Image HDU* and a *BANDS HDU*. The image HDU contains the image data while the BANDS HDU is an optional extension that stores information about non-spatial dimensions.

There are two supported WCS formats:

- *IMAGE Format*: This is the standard FITS image format whereby data is stored in a FITS ImageHDU. With the exception of handling of non-spatial dimensions images generated with this format follow standard FITS format conventions.
- *SPARSE Format*: This is a sparse data format for FITS images which uses a BinTableHDU to store pixel values.

By default multi-dimensional maps are assumed to have the same projection in each image plane with the WCS projection specified in the header keywords.

Both formats support the specification of non-regular multi-dimensional geometries through the inclusion of `CRPIX`, `CEDLT`, and `CRVAL` columns in the *BANDS HDU*. A non-regular geometry is one in which each image plane may have a different pixelization (e.g. different pixel size or image extent). The `CRPIX`, `CEDLT`, and `CRVAL` override the pixelization of the base WCS projection (defined in the IMAGE HDU header) in each image plane.

Sample Files

- Counts Cube: `FITS`
- Counts Cube (Multi-resolution/non-regular): `FITS`
- Counts Cube (SPARSE Format): `FITS`

Image HDU

The IMAGE HDU contains the map data and can be formatted according to either the *IMAGE Format* or *SPARSE Format* scheme.

WCS FITS Header Keywords

The keywords listed here are those required by the FITS WCS specification.

- **`CRPIX{IDX}`, type: float**
  - Pixel coordinate of reference point for axis `{IDX}`.
- **`CDELT{IDX}`, type: float**
  - Coordinate increment at reference point for axis `{IDX}`.
- **`CTYPE{IDX}`, type: float**
  - Coordinate system and projection code for axis `{IDX}`.
- **`CRVAL{IDX}`, type: float**
  - Coordinate value at reference point for axis `{IDX}`.

Extra Header Keywords

These are extra keywords that are not part of the FITS WCS specification but are required for some features of the GADF format.

- **`WCSSHAPE`, type: string, optional**
  - Comma-separated list with the number of pixels in each dimension in row-major order, e.g. `(4,5,3)` would correspond to a 4x5 image with three image planes. For non-regular geometries the image dimension should be the maximum across all image planes. This keyword is optional for maps in the *IMAGE Format* format.
- **`BANDSHDU`, type: string, optional**
  - Name of HDU containing the BANDS table. If undefined the extension name should be `EBOUNDS` or `ENERGIES`. See *BANDS HDU* for additional details.

IMAGE Format

The IMAGE format uses an ImageHDU to store map values. Dimensions of the image are directly inferred from the data member of the HDU. This is the standard format for WCS-based maps.

SPARSE Format

The SPARSE WCS format uses the same conventions as the *Sparse HEALPix Format*. This format uses a BinTable-HDU to store map values with one row per pixel.

Columns

- **PIX – ndim: 1, unit: None, type: int**
  - Dimension: nrows
  - Flattened pixel index in a given image plane (band). Indices are flattened assuming a column-major ordering for the image. The row to band mapping is defined by the `CHANNEL` column.
- **CHANNEL – ndim: 1, unit: None, type: int**
  - Dimension: nrows
  - Band index. This column is optional for maps with a single band. For efficiency it is recommended to represent this column with a 16-bit integer.
- **VALUE – ndim: 1, unit: None, type: float or int**
  - Dimension: nrows
  - Amplitude in pixel indexed by `PIX` and `CHANNEL`.

This page describes data formats for data structures representing images in celestial coordinates. A sky map may have one or more non-spatial dimensions (e.g. energy or time) such that the data consists of a sequence of image planes. There are two primary pixelization formats which are described in more detail in the following sub-pages:

- *HEALPix Formats*
- *WCS Formats*

Both pixelization formats store the data in a single IMAGE HDU which can be either an ImageHDU or BinTableHDU depending on the image format. A *BANDS HDU* is required for maps with non-spatial dimensions.

1.6.3 BANDS HDU

For maps with non-spatial dimensions, the BANDS table defines the geometry in each band and the band to coordinate mapping for non-spatial dimensions (e.g. energy). The BANDS table is optional for maps with a single band.

The extension name of the BANDS table associated to an image HDU is given by the `BANDSHDU` header keyword of the image HDU. If `BANDSHDU` is undefined the BANDS table should be read from the `EBOUNDS` or `ENERGIES` HDU. The BANDS table extension names `EBOUNDS` and `ENERGIES` are reserved for maps with a third energy dimension and are supported for backward compatibility with existing file format conventions of the Fermi STs. Although each map will have its own IMAGE HDU, a single BANDS table can be associated to multiple maps (if they share the same geometry).

The BANDS table contains 1 row per band with columns that define the mapping of the band to the non-spatial dimensions of the map. For integral quantities (e.g. counts) this should be the lower and upper edge values of the bin. For differential quantities this should be the coordinates at which the map value was evaluated. Some examples of quantities that can be used to define bands are as follows:

- Energy (Integral): `E_MIN`, `E_MAX`
- Energy (Differential): `ENERGY`
- Event Type: `EVENT_TYPE`
- Time: `T_MIN`, `T_MAX`
- FoV Angle: `THETA_MIN`, `THETA_MAX`

The mapping of column to non-spatial dimension is defined with the `AXCOLS{IDX}` header keyword. For maps with multiple non-spatial dimensions the mapping of channel number to band indices follows a column-major ordering.

For instance for a map with a first energy dimension with 3 bins indexed by *i* and a second time dimension with 2 bins indexed by *j* the band index mapping to channel number *k* would be as follows:

```
(i, j) = (0,0) : (k) = (0,)
(i, j) = (1,0) : (k) = (1,)
(i, j) = (2,0) : (k) = (2,)
(i, j) = (0,1) : (k) = (3,)
(i, j) = (1,1) : (k) = (4,)
(i, j) = (2,1) : (k) = (5,)
```

Header Keywords

- **`AXCOLS{IDX}`, type: string, optional**
  - Comma-separated list of columns in the BANDS table corresponding to non-spatial dimension with one-based index `{IDX}`. If there are two elements the columns should be interpreted as the lower and upper edges of each bin. If there is a single element the column should be interpreted as the bin center. For `EBOUNDS` or `ENERGIES` HDUs this keyword is optional.

Columns

- **`CHANNEL`, ndim: 1**
  - Dimension: nbands
  - Unique index of the band. If this column is not defined then the band index should be inferred from the row number indexing from zero. For maps with multiple non-spatial dimensions the index mapping to channel number follows a column-major ordering.
- **`E_MIN`, ndim: 1, unit: keV, optional**
  - Dimension: nbands
  - Lower energy bound for integral quantities.
- **`E_MAX`, ndim: 1, unit: keV, optional**
  - Dimension: nbands
  - Upper energy bound for integral quantities.
- **`ENERGY`, ndim: 1, unit: keV, optional**
  - Dimension: nbands
  - Energy value for differential quantities.
- **`EVENT_TYPE`, ndim: 1, optional**
  - Dimension: nbands
  - Integer key for a sequence of independent data subselections (e.g. FRONT/BACK-converting LAT events).

WCS Columns

This section lists BANDS table columns specific to the *WCS map format*. These are optional columns to specify the pixelization for non-regular geometries.

- **`NPIX`, ndim: 2, type: int, optional**

  - Number of pixels in longitude and latitude in each image plane.
- **`CRPIX`, ndim: 2, type: float, optional**
  - Reference pixel coordinate [deg] in longitude and latitude in each image plane.
- **`CDELT`, ndim: 2, type: float, optional**
  - Pixel size [deg] in longitude and latitude in each image plane.

HPX Columns

This section lists BANDS table columns specific to the *HEALPix map format*.

- **`NSIDE` – ndim: 1,**
  - Dimension: nbands
  - NSIDE of the HEALPix pixelization in this band. If not defined then `NSIDE` should be inferred from the FITS header. Only required for formats that support energy-dependent pixelization (`SPARSE`).

1.7 Spectra

Here we describe data formats for high-level spectral analysis results.

Science tools are encouraged to use these formats for easy interoperability with other codes (e.g. to check results, combine results in one plot, ...).

1.7.1 SED

The SED format is a flexible specification for representing one-dimensional spectra (distributions of amplitude vs. energy). The SED is structured as a table with one row per energy bin/point and columns for the energy, measured normalization, and normalization errors. The format supports both integral and differential representations of the normalization as described in *Normalization Representation*.

The list of supported columns is given in the *Columns* section. All columns are optional by default, and an SED may contain any combination of the allowed columns. *SED Types* are a specification for defining groups of columns that are required to be present in the file. The *Likelihood SED* format is an example of an SED type that contains both measured flux points and the profile likelihoods versus normalization in each energy bin.

The SED format does not enforce a specific set of units but units should be defined in the column metadata for all quantities with physical dimensions. Recommended units are provided in the *Columns* section to indicate the dimensionality of each column. Column UCDs are intended for defining the type of physical quantity associated to each column (e.g. energy, photon flux, etc.). The convention for including UCDs in the column metdata is still under discussion and the UCDs defined in the current format are optional.

The intended serialization format is a FITS BINTABLE with one row per energy bin. However any serialization format that supports tabular data and column metadata could also be supported (e.g. ECSV or HDF5). Because the SED occupies a single HDU multiple SEDs can be written to a single FITS file with an identifier (e.g. source name or observation epoch) used as the HDU name. Sample FITS and ECSV files are provided in *Sample Files*.

Normalization Representation

The SED format supports both differential and integral representations of the source normalization. Integral representations correspond to quantities integrated over an energy bin as defined by the `e_min` and `e_max` columns. Differential representations are quantities evaluated at a discrete energies defined by the `e_ref` column. The supported *Normalization Columns* are:

- `dnde` : Differential photon flux at `e_ref`. Dimensionality: photons / (time * area * energy)
- `e2dnde` : Defined as `(e_ref ^ 2) * dnde`. A commonly published and plotted differential flux quantity. Dimensionality: energy / (time * area)
- `rate` : Photon rate between `e_min` and `e_max`. Dimensionality: photons / time.
- `flux` : Photon flux (integral of `dnde`) between `e_min` and `e_max`. Dimensionality: photons / (time * area)
- `eflux` : Energy flux (integral of E times `dnde`) between `e_min` and `e_max`. Dimensionality: energy / (time * area)
- `npred` : Photon counts between `e_min` and `e_max`. Dimensionality: photons
- `norm` : Normalization in units of the reference model. Dimensionality: unitless

An SED should contain at least one of the normalization representations listed above. Multiple representations (e.g. `flux` and `dnde`) may be included in a single SED.

The `dnde` and `e2dnde` representations are equivalent. We define both here, because both are in common use for publications and plots.

Errors and upper limits on the normalization are defined with the *Error Columns* by appending the appropriate suffix to the normalization column name. For example in the case of photon flux the error and upper limit columns are:

- `flux_err` : Symmetric 1-sigma error.
- `flux_errp` : Asymmtric 1-sigma positive error.
- `flux_errn` : Asymmtric 1-sigma negative error.
- `flux_ul` : Upper limit with confidence level given by `UL_CONF` header keyword.

A row may have a value and any combination of upper limits and errors:

```
e_ref dnde dnde_err dnde_errp dnde_errn dnde_ul
1000.0 1.0 0.1 0.1 0.1 1.16
3000.0 1.0 0.1 0.1 0.1 1.16
```

A `nan` value should be used for empty or missing data such as a bin for which there is an upper limit but no value and vice versa, e.g.

```
e_ref dnde dnde_err dnde_ul
1000.0 1.0 0.1 nan
3000.0 nan nan 1.16
```

The `is_ul` column is an optional boolean flag that can be used to designate whether the measurement in a given row should be interpreted as a two-sided confidence interval or an upper limit:

```
e_ref dnde dnde_err dnde_ul is_ul
1000.0 1.0 0.1 nan False
3000.0 nan nan 1.16 True
```

Setting UL values to `nan` is an implicit way of flagging rows that do not contain an upper limit. When parsing an SED one should first check for the existence of the `is_ul` column and otherwise check for `nan` values in the UL column. The `is_ul` column is only required if you want to explicitly flag ULs when both the UL and two-sided interval may be defined in a row.

Reference Model

The *reference model* of an SED is the global parameterization that was used to extract the normalization in each energy bin. The choice of reference model is relevant when considering models that are rapidly changing across a bin or when energy dispersion is large relative to the bin size. The *Reference Model Columns* define the reference model in different representations. When an SED includes a reference model, the normalizations, errors, and upper limits can be given in the `norm` representation which is expressed in units of the reference model. `norm` columns can be converted to another representation by performing an element-wise multiplication of the column with the `ref` column of the desired representation.

Likelihood Profiles

The *Likelihood Columns* contain values of the model likelihood and likelihood profiles versus normalization. Likelihood profiles provide additional information about the measurement uncertainty in each bin. A more detailed discussion of the motivation for SED likelihood profiles can be found in *Likelihood SED*.

SED Types

By default all columns in the SED format are optional. To facilitate interoperability of files produced by different packages/tools, the SED format defines an *SED Type* string which is set with the `SED_TYPE` header keyword. The SED type defines a minimal set of columns that must be present in the SED. The SED types and their required columns are given in the following list:

- `dnde`: `e_ref`, `dnde`
- `e2dnde`: `e_ref`, `e2dnde`
- `flux`: `e_min`, `e_max`, `flux`
- `eflux`: `e_min`, `e_max`, `eflux`
- `likelihood`: See *Likelihood SED*.

Sample Files

- Differential Flux Points: `FITS` `ECSV` `H5`
- Integral Flux Points: `FITS` `ECSV` `H5`
- Likelihood SED: `FITS` `H5`
- H.E.S.S. 1ES0229 Spectrum: `FITS` `ECSV`

Header Keywords

- **`UL_CONF`, optional**
  - Confidence level of the upper limit (range: 0 to 1) of the value in the `{NORM_REP}_ul` column.
- **`SED_TYPE`, optional**
  - SED type string (see *SED Types* for more details).

Columns

This sections lists the column specifications. Unless otherwise specified the data type of all columns should be float32 or float64.

Energy Columns

- **`e_min` – ndim: 1, unit: MeV**
  - Dimension: nebins
  - ucd : `em.energy`
  - Lower edge of energy bin. This defines the lower integration bound for integral representations of the normalization. Can also define the energy band used to evaluate differential representations (`dnde` or `e2dnde`).
- **`e_max` – ndim: 1, unit: MeV**
  - Dimension: nebins
  - ucd : `em.energy`
  - Upper edge of energy bin. This defines the upper integration bound for integral representations of the normalization. Can also define the energy band used to evaluate differential representations (`dnde` or `e2dnde`).
- **`e_ref` – ndim: 1, unit: MeV**
  - Dimension: nebins
  - ucd : `em.energy`
  - Energy at which differential representations of the normalization are evaluated (e.g. `dnde`). This can be the geometric center of the bin or some weighted average of the energy distribution within the bin.

Normalization Columns

- **`norm` – ndim: 1, unit: None**
  - Dimension: nebins
  - Measured normalization in units of the reference model.
- **`dnde` – ndim: 1, unit: 1 / (cm2 s MeV)**
  - Dimension: nebins
  - ucd : `phot.flux.density`
  - Measured differential photon flux at `e_ref`.
- **`e2dnde` – ndim: 1, unit: MeV / (cm2 s)**
  - Dimension: nebins
  - ucd : `phot.flux.density`
  - Measured differential photon flux at `e_ref`, multiplied with `e_ref ^ 2`.
- **`flux` – ndim: 1, unit: 1 / (cm2 s)**
  - Dimension: nebins

  - ucd : `phot.count`
  - Measured photon flux between `e_min` and `e_max`.
- **`eflux` – ndim: 1, unit: MeV / (cm2 s)**
  - Dimension: nebins
  - ucd : `phot.flux`
  - Measured energy flux between `e_min` and `e_max`.
- **`npred` – ndim: 1**
  - Dimension: nebins
  - Measured counts between `e_min` and `e_max`.

Error Columns

The error columns define the error and upper limit for a given representation of the normalization. In the following column specifications `{NORM_REP}` indicates a placeholder for the name of the normalization column (e.g. `flux_err`).

- **`{NORM_REP}_err` – ndim: 1**
  - Dimension: nebins
  - Symmetric error on the normalization in the representation `{NORM_REP}`.
- **`{NORM_REP}_errp` – ndim: 1**
  - Dimension: nebins
  - Positive error on the normalization in the representation `{NORM_REP}`.
- **`{NORM_REP}_errn` – ndim: 1**
  - Dimension: nebins
  - Negative error on the normalization in the representation `{NORM_REP}`. A negative or NaN value indicates that the negative error is undefined.
- **`{NORM_REP}_ul` – ndim: 1**
  - Dimension: nebins
  - Upper limit on the normalization in the representation `{NORM_REP}`. The upper limit confidence level is specified with the `UL_CONF` header keyword.
- **`is_ul` – ndim: 1, type: bool**
  - Dimension: nebins
  - Boolean flag indicating whether a row should be interpreted as an upper limit. If `True` then one should represent the measurement with the one-sided confidence interval defined by `{NORM_REP}_ul`. If `False` then the measurement should be represented by the two-sided intervals defined by `{NORM_REP}_err` or `{NORM_REP}_errp` and `{NORM_REP}_errn`.

Reference Model Columns

- **`ref_dnde` – ndim: 1, unit: 1 / (MeV cm2 s)**
  - Dimension: nebins

  - Differential flux of reference model at the `e_ref`.
- **`ref_eflux` – ndim: 1, unit: MeV / (cm2 s)**
  - Dimension: nebins
  - Energy flux (integral of E times `dnde`) of reference model from `e_min` to `e_max`.
- **`ref_flux` – ndim: 1, unit: 1 / (cm2 s)**
  - Dimension: nebins
  - Flux (integral of `dnde`) of reference model from `e_min` to `e_max`.
- **`ref_dnde_e_min` – ndim: 1, unit: 1 / (MeV cm2 s)**
  - Dimension: nebins
  - Differential flux of reference model at `e_min`.
- **`ref_dnde_e_max` – ndim: 1, unit: 1 / (MeV cm2 s)**
  - Dimension: nebins
  - Differential flux of reference model at `e_max`.
- **`ref_npred` – ndim: 1, unit: counts**
  - Dimension: nebins
  - Number of photon counts of reference model.

Likelihood Columns

- **`ts` – ndim: 1, unit: none**
  - Dimension: nebins
  - Source test statistic in each energy bin defined as twice the difference between best-fit and null log-likelihood values. In the asymptotic limit this is square of the significance.
- **`loglike` – ndim: 1, unit: none**
  - Dimension: nebins
  - Log-Likelihood value of the best-fit model.
- **`loglike_null` – ndim: 1, unit: none**
  - Dimension: nebins
  - Log-Likelihood value of the zero normalization model.
- **`{NORM_REP}_scan` – ndim: 2, unit: None**
  - Dimension: nebins x nnorms
  - Likelihood scan points in each energy bin in the representation `{NORM_REP}`.
- **`dloglike_scan` – ndim: 2, unit: none**
  - Dimension: nebins x nnorms
  - Scan over delta LogLikelihood value vs. normalization in each energy bin. The Delta-Loglikelihood is evaluated with respect to the zero normalization model (`loglike_null`).

1.7.2 Bin-by-bin Likelihood Profiles

This page describes formats for bin-by-bin likelihood profiles as currently used in some LAT analyses. The bin-by-bin likelihood extends the concept of an SED by providing a representation of the likelihood function in each energy bin. *Likelihood SED* and *Likelihood SED Cube* are two formats for serializing bin-by-bin likelihoods to a FITS file. A Likelihood SED stores the bin-by-bin likelihood for a single source or test source position while a Likelihood SED Cube stores a sequence of bin-by-bin likelihoods (e.g. for a grid of positions or a group of sources).

In the following we describe some advantages and limitations of using bin-by-bin likelihoods. Relative to a traditonal SED, the bin-by-bin likelihood retains more information about the shape of the likelihood function around the maximum. This can be important when working in the low statistics regime where the likelihoods are non-Gaussian and a flux value and one sigma uncertainty is insufficient to describe the shape of the likelihood function. Applications in which bin-by-bin likelihoods may be useful include:

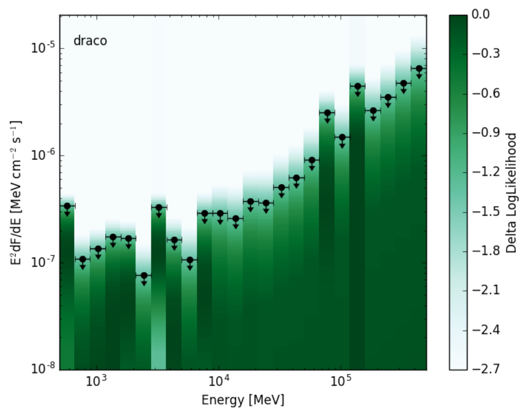
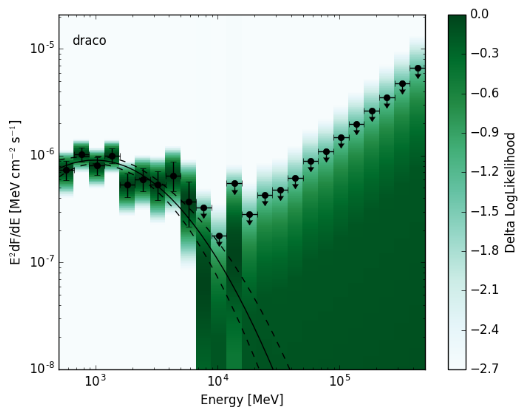
- Deriving upper limits on the global spectral distribution of a source. Likelihood SEDs can be used to construct the likelihood function for arbitrary spectral models without recomputing the experimental likelihood function. This is particularly useful for DM searches in which one tests a large number of spectral models (e.g. for mass and annihilation channel) and recomputing the experimental likelihood function for all models would be very expensive. The bin-by-bin likelihoods are also a convenient way of distributing analysis results in a format that allows other spectral models to be easily tested. The two of the most recent LAT publications on dSph DM searches have publicly released the analysis results in this format (see 2015PhRvL.115w1301A and 2014PhRvD..89d2001A).
- Stacking analyses that combine measurements from multiple sources or multiple epochs of observation of a single source. Forming a joint likelihood from the product of Likelihood SEDs fully preserves information in each data set and is equivalent to doing a joint fit as long as the data sets are independent.
- Analyses combining spectral measurements from multiple experiments. Likelihoods from two or more experiments can be multiplied to derive a joint likelihood function incorporating the measurements of each experiment. As for stacking analyses, the joint likelihood approach avoids merging or averaging data or IRFs. The bin-by-bin likelihoods further allow joint anlayses to be performed without having access to the data sets or tools that produced the original measurement. For an application of this approach in the context of DM searches see 2016JCAP...02..039M.

There are a few important caveats to bin-by-bin likelihoods which may limit their use for certain applications:

- Large correlations between the normalizations of two or more model components (e.g. when the spatial models are partially degenerate) can limit the utility of this approach. Although such correlations can be accounted for by profiling the corresponding nuisance parameters, this may result in unphysical background models with large bin-to-bin fluctuations in the model amplitude. One technique to avoid this issue (see 2015PhRvD..91j2001B and 2016PhRvD..93f2004C) is to apply a Gaussian prior that constrains the spectral distribution of the background components to lie within a certain range of the global spectral model of that source (computed without the test source).
- Because the likelihoods in each energy bin are calculated independently, this technique cannot fully account for bin-to-bin correlations caused by energy dispersion. The effect of energy dispersion can be corrected to first order by scanning the likelihood with a spectral model (e.g. a power-law with index 2) that is close in shape to the spectral models of interest. However in analyses where the energy response matrix is particularly broad or non-diagonal the systematic errors arising from the approximate treatment of energy dispersion may exceed the statistical errors. In LAT analyses energy dispersion can become a significant effect when using data below 100 MeV (see LAT_edisp_usage). However when using an Index=2.0 and considering energies above 100 MeV, the spectral bias is less than 3% for models with indices between 1 and 3.5.

Likelihood SED

The likelihood SED is a representation of spectral energy distribution of a source that contains a likelihood for the source normalization in each energy bin. This format is a special case of the more general *SED* format. Depending on the requirements of the analysis the likelihoods can be evaluated with either profiled or fixed nuisance parameters. The likelihood SED can be used in the same way as a traditional SED but contains additional information about the shape of the likelihood function around the maximum. A 2D visualization of the likelihood functions can be produced by creating a colormap with intensity mapped to the likelihood value:

Low Significance Source	High Significance Source
	

In the following we use *nebins* to designate the number of energy bins and *nnorms* to designate the number of points in the normalization scan. The format is a BINTABLE with one row per energy bin containing the columns listed below.

The best-fit model amplitudes, errors, and upper limits are all normalized to a reference spectral model. The `ref` columns define the amplitude of the reference model in different units. The reference model amplitudes are arbitrary and could for instance be set to the best-fit amplitude in each energy bin. `norm` columns contain the best-fit value, its errors, and upper limit in units of the reference model amplitude. Unit conversion of the `norm` columns can be performed by doing a row-wise multiplication with the respective `ref` column.

Sample Files

See the `likelihood` files in *Sample Files*.

Header Keywords

- **`SED_TYPE`**
  - SED type string. Should be set to `likelihood`.
- **`UL_CONF`, optional**
  - Confidence level of the upper limit (range: 0 to 1) of the value in the `norm_ul` column.

Columns

The columns listed here are a subset of the columns defined in the *SED* format. See *Columns* for the full column specifications.

Required Columns

- `e_min` – ndim: 1, Dimension: nebins

- `e_max` – ndim: 1, Dimension: nebins
- `e_ref` – ndim: 1, Dimension: nebins
- `ref_dnde` – ndim: 1, Dimension: nebins
- `ref_eflux` – ndim: 1, Dimension: nebins
- `ref_flux` – ndim: 1, Dimension: nebins
- `ref_npred` – ndim: 1, Dimension: nebins
- `norm` – ndim: 1, Dimension: nebins
- `norm_err` – ndim: 1, Dimension: nebins
- `norm_scan` – ndim: 2, Dimension: nebins x nnorms
- `ts` – ndim: 1, Dimension: nebins
- `loglike` – ndim: 1, Dimension: nebins
- `dloglike_scan` – ndim: 2, Dimension: nebins x nnorms

Optional Columns

- `ref_dnde_e_min` – ndim: 1, Dimension: nebins
- `ref_dnde_e_max` – ndim: 1, Dimension: nebins
- `norm_errp` – ndim: 1, Dimension: nebins
- `norm_errn` – ndim: 1, Dimension: nebins
- `norm_ul` – ndim: 1, Dimension: nebins

Likelihood SED Cube

The Likelihood SED Cube is format for storing a sequence of Likelihood SEDs in a single table. The format defines a file with two BINTABLE HDUs: *SCANDATA* and *EBOUNDS*. SCANDATA has one row per Likelihood SED while EBOUNDS has one row per energy bin. Table rows in SCANDATA can be mapped to a list of sources, spatial pixels, or observations epochs. Because the row mapping is not defined by the format itself additional columns can be added to SCANDATA that defined the mapping of each row. Examples would be columns for source name designation, pixel coordinate, or observation epoch.

In the following we use *nrows* to designate table rows, *nebins* to designate the number of energy bins and *nnorms* to designate the number of points in the normalization scan. As for the Likelihood SED format, columns that contain `norm` are expressed in units of the reference model amplitude. These can be multiplied by `ref_eflux`, `ref_flux`, `ref_dnde`, or `ref_npred` columns in the EBOUNDS HDU to get the normalization in the respective units.

Sample FITS files:

- Low Significance Source: `tscube_lowts.fits`
- High Significance Source: `tscube_hights.fits`

SCANDATA Table

The SCANDATA HDU is a BINTABLE with the following columns. The columns listed here are a subset of the columns in the *SED* format. Relative to the 1D SED formats the dimensionality of all columns is increased by one with the first dimension (rows) mapping to spatial pixels. See *Columns* for the full column specifications.

Header Keywords

- **`UL_CONF`, optional**
  - Confidence level of the upper limit given in the `norm_ul` column.

Required Columns

- `dloglike_scan` – ndim: 3, Dimension: nrows x nebins x nnorms
- `norm_scan` – ndim: 3, Dimension: nrows x nebins x nnorms
- `norm` – ndim: 2, Dimension: nrows x nebins
- `norm_err` – ndim: 2, Dimension: nrows x nebins
- `ts` – ndim: 2, Dimension: nrows x nebins
- `loglike` – ndim: 2, Dimension: nrows x nebins

Optional Columns

- `ref_npred` – ndim: 2, Dimension: nrows x nebins
- `norm_errp` – ndim: 2, Dimension: nrows x nebins
- `norm_errn` – ndim: 2, Dimension: nrows x nebins
- `norm_ul` – ndim: 2, Dimension: nrows x nebins
- **`bin_status` – ndim: 2, unit: None**
  - Dimension: nrows x nebins
  - Fit status code. 0 = OK, >0 = Not OK

EBOUNDS Table

The EBOUNDS HDU is a BINTABLE with 1 row per energy bin and the following columns. The columns listed here are a subset of the columns in the *SED* format. See *Columns* for the full column specifications. Note that for backwards compatibility with existing EBOUNDS table convention (e.g. as used for WCS counts cubes) columns names are upper case.

Required Columns

- `E_MIN`, unit: keV, Dimension: nebins
- `E_REF`, unit: keV, Dimension: nebins
- `E_MAX`, unit: keV, Dimension: nebins
- `REF_DNDE` – ndim: 1, Dimension: nebins
- `REF_EFLUX` – ndim: 1, Dimension: nebins
- `REF_FLUX` – ndim: 1, Dimension: nebins

Optional Columns

- `REF_DNDE_E_MIN` – ndim: 1, Dimension: nebins
- `REF_DNDE_E_MAX` – ndim: 1, Dimension: nebins
- `REF_NPRED` – ndim: 1, Dimension: nebins

TSCube Output Format

Recent releases of the Fermi ScienceTools provide a *gttscube* application that fits a test source on a grid of spatial positions within the ROI. At each test source position this tool calculates the following information:

- TS and best-fit amplitude of the test source.
- A likelihood SED.

The output of the tool is a FITS file containing a Likelihood SED Cube with *nrows* in which each table row maps to a pixel in the grid scan. The PRIMARY HDU contains the same output as *gttsmap* – a 2-dimensional FITS IMAGE with the test source TS evaluated at each position. The primary fit results are contained in the following BINTABLE HDUs:

- A *SCANDATA Table* containing the likelihood SEDs for each spatial pixel.
- A *FITDATA Table* containing fit results for the reference model at each spatial pixel over the full energy range.
- A *EBOUNDS Table* containing the bin definitions and the amplitude of the reference model.

The mapping of rows to pixels is defined by the WCS header keywords in the SCANDATA HDU. Following the usual FITS convention both tables use columnwise ordering for mapping rows to pixel indices.

Here is the list of HDUs:

Table 1.1: TS Cube HDUs

HDU	HDU Type	HDU Name	Description
0	IMAGE	PRIMARY	TS map of the region using the test source
1	BINTABLE	SCANDATA	Table with the data from the likelihood v. normalization scans. Follows format specification given in *SCANDATA Table*.
2	BINTABLE	FITDATA	Table with the data from the reference model fits.
3	BINTABLE	BASELINE	Parameters and Covariences of Baseline fit.
4	BINTABLE	EBOUNDS	Energy bin edges, fluxes and NPREDs for test source in each energy bin. Follows format specification given in *EBOUNDS Table*.

FITDATA Table

The FITDATA HDU is a BINTABLE with 1 row per spatial pixel (*nrows*) and the following columns:

- **`fit_norm` – ndim: 1, unit: None**
  - Dimension: nrows
  - Best-fit normalization for the global model in units of the reference model amplitude.
- **`fit_norm_err` – ndim: 1, unit: None**

  - Dimension: nrows
  - Symmetric error on the global model normalization in units of the reference model amplitude.
- **`fit_norm_errp` – ndim: 1, unit: None**
  - Dimension: nrows
  - Positive error on the global model normalization in units of the reference model amplitude.
- **`fit_norm_errn` – ndim: 1, unit: None**
  - Dimension: nrows
  - Negative error on the global model normalization in units of the reference model amplitude.
- **`fit_norm_ul` – ndim: 1, unit: None**
  - Dimension: nrows
  - Upper limit on the global model normalization in units of the reference model amplitude.
- **`fit_ts` – ndim: 1, unit: None**
  - Dimension: nrows
  - Test statistic of the best-fit global model.
- **`fit_status` – ndim: 1, unit: None**
  - Dimension: nrows
  - Status code for the fit. 0 = OK, >0 = Not OK