Gamma Astro Data Specs Documentation

Release 0.1

Gamma-ray astronomy community

Jun 20, 2018

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: http://gamma-astro-data-formats.readthedocs.org/
- Mailing list: https://lists.nasa.gov/mailman/listinfo/open-gamma-ray-astro

CHAPTER 1

Table of contents

1.1 Background information

This section contains background 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 In some cases the formats described here will likely be adopted or superceded by more official formats in the coming years.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 here, 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:

- *IACT event lists*:
  - 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` – float, MJD, days
  - Column `TSTART_STR`, `TSTOP_STR`, `TMID_STR` – UTC string
  - No time-related header keywords.

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 Sky 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 here 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 here).
- 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

Field of view coordinates for a given observation have the pointing position at `(0, 0)`.

At the moment they are only used for background modeling, where off runs are stacked in the field of view coordinate system. We are also discussing if we should use them for IRFs like effective area, where for large field of views a gradient due to varying zenith angle can be present and we'd like to store this dependency / use it in exposure computations.

In detail the definition of field of view coordinates is tricky and still under discussion.

The main questions are:

- How exactly are the field of view coordinates defined?
- Is a projection (e.g. the FITS TAN aka gnomonic projection) involved or are they spherical coordinates? I.e. are they angles or lengths?
- Are the field of view coordinate axes aligned with RA / DEC or ALT / AZ? (we probably need or at least want both for different applications / investigations, i.e. there are two field of view coordinates.)
- How should this be stored in FITS (e.g. axis info or even WCS object in background cube models)

Here's some useful links about the TAN projection:

- https://en.wikipedia.org/wiki/Gnomonic_projection
- http://mathworld.wolfram.com/GnomonicProjection.html
- http://bl.ocks.org/mbostock/3795048

- http://adsabs.harvard.edu/abs/2002A%26A…395.1077C

TODO: document what exactly is filled / assumed at the moment in the HESS exporters, Gammalib and Gammapy.

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 /
```

(continues on next page)

(continued from previous page)

```
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
        ----               ----       ----------
```

(continues on next page)

(continued from previous page)

```
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)'
```

(continues on next page)

(continued from previous page)

```
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 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 IACT DL3 event lists.

Event lists are stored in FITS files with two required and one optional extensions (HDUs).

- Suggested filename: `events_OBS_ID.fits.gz`
- Suggested HDU name events: `EVENTS`
- Suggested HDU name good time intervals: `GTI`
- Suggested HDU name telescope array: `TELARRAY`

1.2.1 EVENTS extension

The first extension contains characteristic information about each event. Suggested extension name `EVENTS`. These information are stored in a FITS binary table. The columns are listed below. In addition, a list of header keywords to be contained in each FITS event list is also documented. Many of the keywords are not necessarily required for an analysis. The information is, however, included as meta data in the event lists to enable instrument-dependent studies and selections of particular observations.

1.2.2 Required columns

- **`EVENT_ID` type: int**
  - Event identification number at the DL3 level (lower data levels could be different, see note below).
  - Required: The pair (`OBS_ID`, `` `EVENT_ID ``) must be globally unique for all events from a given instrument. (to be discussed … it's not clear if CTA will have "runs" `OBS_ID`)

    - Required: `EVENT_ID` should increase monotonically with `TIME`. (to be discussed if this should be changed to a recommendation only)
    - Required: event lists should be sorted by `EVENT_ID` and `TIME`. (to be discussed if this should be changed to a recommendation only)

- **`TIME` type: double, unit: s**
    - Time stamp of event in MET
- **`RA` type: float, unit: deg**
    - Event right ascension (see *RA / DEC*)
- **`DEC` type: float, unit: deg**
    - Event declination (see *RA / DEC*)
- **`ENERGY` type: float, unit: TeV**
    - Reconstructed event energy

1.2.3 Notes on EVENT_ID

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 achive 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).

1.2.4 Optional Column Names

- **`MULTIP` type: int**
    - Telescope multiplicity. Number of telescopes that have seen the event
- **`OBS_ID` type: int**
    - Unique observation identifier (Run number)
- **`DIR_ERR` type: float, unit: deg**
    - Direction error of reconstruction
- **`ENERGY_ERR` type: float, unit: TeV**
    - Error on reconstructed event energy

- **`ALT` type: float, unit: deg**
  - Altitude coordinate of event (horizon system, see *Alt / Az*)
- **`AZ` type: float, unit: deg**
  - Azmuth coordinate of event (horizon system, see *Alt / Az*)
- **`DETX` type: float, unit: deg**
  - X-coordinate in detector system (nominal system, see *Field of view*)
- **`DETY` type: float, unit: deg**
  - Y-coordinate in detector system (nominal system, see *Field of view*)
- **`THETA` type: float, unit: deg**
  - Offset from the observation pointing position
- **`COREX` type: float, unit: m**
  - Core position X of shower
- **`COREY` type: float, unit: m**
  - Core position Y of shower
- **`CORE_ERR` type: float, unit: m**
  - Error on core position of shower
- **`XMAX` type: float, unit: radiation lengths**
  - First interaction depth
- **`XMAX_ERR` type: float, unit: radiation lengths**
  - Error on first interaction depth
- **`HIL_MSW` type: float**
  - Hillas mean scaled width
- **`HIL_MSW_ERR` type: float**
  - Hillas mean scaled width error
- **`HIL_MSL` type: float**
  - Hillas mean scaled length
- **`HIL_MSL_ERR` type: float**
  - Hillas mean scaled length error

1.2.5 Required Header keywords:

- **`OBS_ID` type: int**
  - Unique observation identifier (Run number)
- **`TELESCOP` type: int**
  - Telescope (e.g. ‘HESS’)
- **`TSTART` type: float, unit: s**
  - Start time of observation [MET]

- **`TSTOP` type: float, unit: s**
  - End time of observation [MET]
- **`TSTART_STR` type: string**
  - Start of observation in UTC string format: "YYYY-MM-DD HH:MM:SS"
- **`TSTOP_STR` type: string**
  - End of observation in UTC string format: "YYYY-MM-DD HH:MM:SS"
- **`MJDREFI` type: int, unit: days**
  - Integer part of MJD time reference
- **`MJDREFF` type: float, unit: days**
  - Float part of MJD time reference
- **`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.
- **`OBJECT` type: string**
  - Observed object
- **`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*)
- **`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*)
- **`RA_OBJ` type: float, unit: deg**
  - Right ascension of `OBJECT`
- **`DEC_OBJ` type: float, unit: deg**
  - Declination of `OBJECT`
- **`TELLIST` type: string**
  - Telescope IDs in observation (e.g. '1,2,3,4')
- **`N_TELS` type: int**

- Number of observing telescopes

- **`EUNIT` type: string**
  - Unit of energies in event list (e.g. 'TeV')
- **`GEOLON` type: float, unit: deg**
  - Geographic longitude of array centre (e.g. -23.27 for HESS)
- **`GEOLAT` type: float, unit: deg**
  - Geographic latitude of array centre (e.g. -16.5 for HESS)
- **`ALTITUDE` type: float, unit: km**
  - Altitude of array center above sea level (1.835 for HESS)

1.2.6 Optional header keywords

- **`OBSERVER` type: string**
  - Name of observer (e.g. 'HESS'). This could be the PI of a proposal later on.
- **`CREATOR` type: string**
  - Software that created the file
- **`CREATED` type: string**
  - Time when file was created (UTC): "YYYY-MM-DD HH:MM:SS"
- **`RADECSYS` type: string**
  - Equatorial system type (e.g. 'FK5')
- **`EQUINOX` type: float**
  - Base equinox (e.g. 2000.)
- **`TIMESYS` type: string**
  - Time system (currently 'TT')
- **`TIMEREF` type: string**
  - Time reference ('LOCAL')
- **`TASSIGN` type: string**
  - Place of time reference ('Namibia')
- **`OBS_MODE` type: string**
  - Observation mode (e.g. wobble, survey, or any mode that is supported by `TELESCOP`)
- **`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?

1.2.7 GTI extension

Each event list file contains an extension to specify the good time intervals ('GTIs'). A general description of GTIs can be found in the OGIP GTI standard. This HDU contains two 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 associated 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). Eventually, this extension could disappear from the required extensions. High-level Science tools could add the GTIs to the files according to user parameter. See e.g. gtmktime for an application example from the Fermi Science Tools. The column names and FITS header keywords are documented in the following, respectively.

1.2.8 GTI Column Names

- **`START` type: double, unit: s**
  - Start time of good time interval (observation) [MET]
- **`STOP` type: double, unit: s**
  - End time of good time interval (observation) [MET]

1.2.9 GTI Header Keywords

- **`MJDREFI` type: int, unit: days**
  - Integer part of MJD time reference
- **`MJDREFF` type: float, unit: days**
  - Float part of MJD time reference

1.2.10 TELARRAY extension

To be defined

1.3 IACT IRFs

The instrument response function (IRF) formats currently in use for imaging atmospheric Cherenkov telescopes (IACTs) are stored in FITS binary tables using the "multidimentional 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.

At the moment (November 2015), this format is used by H.E.S.S. and VERITAS and supported by Gammapy and Gammalib and is being proposed for DL3 IRFs (i.e. the format distributed to end users and used by the science tools for CTA). Note that a different format has been proposed for CTA to serialise multidimensional arrays and axis information: http://adsabs.harvard.edu/abs/2015arXiv150807437W As far as we know, this format is currently not supported by any analysis package.

Here we specify the IRFs in use for IACT data.

1.3.1 IRF axes

Most IRFs are dependent on parameters, and the 1-dim. 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).

The IRF format specs 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 specifiec by the `CUNIT` header keywords (see *BINTABLE HDU*)

1.3.2 Specific IRFs

Effective Area

The proposed effective area format follows mostly the OGIP effective area format document.

For the moment, the format for the effective area works to satisfactory detail. 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.

`aeff_2d` format

The effective area information is saved as a *BINTABLE HDU* with required columns listed below.

Columns:

- **`THETA_LO`, `THETA_HI` – ndim: 1, unit: deg**
  - Field of view offset axis
- **`ENERG_LO`, `ENERG_HI` – ndim: 1, unit: TeV**
  - Energy axis
- **`EFFAREA` – ndim: 2, unit: none**
  - Effective area value as a function of true energy
- **`EFFAREA_RECO` – ndim: 2, unit: deg**
  - Effective area value as a function of reco 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. Another optional header keyword contains the theta squared cut that was applied in the case of a effective area generation for point-like sources.

- **`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
- **`RAD_MAX` type: float, unit: deg**

  - On region radius for point-like observations

An example file is provided `here`.

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**

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

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 $\Delta E_{\rm true}$ within a certain reconstructed energy bin *J* of width $\Delta E_{\rm reco}$. In order to do so, the following integration has to be performed (for a fixed offset).

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

where

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

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

`edisp_2d` format

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

Columns:

- **`MATRIX` type: float, dimensions: 3**
  - Matrix holding the probability for a given energy migration at a certain true energy and offset.
- **`ENERG_LO`, `ENERG_HI` – ndim: 1, unit: TeV**
  - Energy axis
- **`THETA_LO`, `THETA_HI` – ndim: 1, unit: deg**
  - Field of view offset axis
- **`MIGRA_LO`, `MIGRA_HI` – ndim: 1, unit: dimensionless**
  - Energy migration axis (defined above)
- **`MATRIX` – ndim: 3, unit: dimensionless**
  - Energy dispersion $dP/d\mu$, see formula above.

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

Header keywords: none

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.

PSF formats

`psf_table` format

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 and the units are different.

Columns:

- **`RAD_LO`, `RAD_HI` – ndim: 1, unit: deg**

  - Offset angle from source position
- **THETA_LO, THETA_HI – ndim: 1, unit: deg**
  - Field of view offset axis
- **ENERG_LO, ENERG_HI – ndim: 1, unit: TeV**
  - Energy axis
- **RPSF – ndim: 3, unit: deg^-2**
  - Point spread function value $dP/d\Omega$, see *Probability distributions*.

Recommended axis order: RAD, THETA, ENERGY.

Header keywords: none

psf_3gauss format

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:

- **THETA_LO, THETA_HI – ndim: 1, unit: deg**
  - Field of view offset axis
- **ENERG_LO, ENERG_HI – ndim: 1, unit: TeV**
  - Energy 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: none

Example data file: TODO: add HESS HAP example file as soon as available.

`psf_king` format

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:

- **`THETA_LO`, `THETA_HI` – ndim: 1, unit: deg**
  - Field of view offset axis
- **`ENERG_LO`, `ENERG_HI` – ndim: 1, unit: TeV**
  - Energy 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: none

Example data file: TODO: add HESS HAP example file as soon as available.

`gtpsf` format

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: none

Example data file: `psf-fermi.fits`

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.

Here we specify two formats for such background template models:

- `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`.

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.

`bkg_2d` format

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

Header keywords:

- `HDU_CLASS = bkg_2d`
- `HDU_DOC = TODO`

Columns:

- **`THETA_LO`, `THETA_HI` – ndim: 1, unit: deg**
  - Field of view offset axis
  - Binning is often chosen with a square root scale, so that each `THETA` bin has equal solid angle, which means bins at the center of the field of view have smaller width `THETA_HI - THETA_LO`.
- **`ENERG_LO`, `ENERG_HI` – ndim: 1, unit: TeV**
  - Energy axis
- **`BKG` ndim: 2, unit: s^-1 MeV^-1 sr^-1**
  - Absolute post-select background rate (expected background per time, energy and solid angle).
  - Note that this is not a "flux" or "surface brightness". This 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.

Recommended axis order: `ENERGY`, `THETA`

Example data file: TODO

`bkg_3d` format

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

TODO: maybe we should we use TeV as units, since we use this for IACTs and also store energy in TeV?

Columns:

- **`ENERG_LO`, `ENERG_HI` – ndim: 1, unit: TeV**
  - 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).
  - Note that this is not a "flux" or "surface brightness". This 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.

Recommended axis order for `BKG`: `ENERGY`, `DETX`, `DETY`

Header keywords:

- `HDU_CLASS = bkg_3d`
- `HDU_DOC = TODO`

Example data file: TODO

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: days**
  - Start of observation in MJD
- **`TSTART_STR` type: string**
  - Start of observation in UTC string format: “YYYY-MM-DD HH:MM:SS”
- **`TSTOP` type: float, unit: days**
  - End time of observation in MJD
- **`TSTOP_STR` type: string**
  - End of observation in UTC string format: “YYYY-MM-DD 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:**

    * 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"
- **`RA_OBJ` type: float, unit: deg**
  - Right ascension of `OBJECT`
- **`DEC_OBJ` type: float, unit: deg**
  - Declination of `OBJECT`
- **`TMID` type: float, unit: days**
  - Mid time of observation in MJD (= `TSTART` + 0.5 * `ONTIME`)
- **`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

Notes

- This table doesn't require header keywords. We recommend FITS is used, but it can be stored e.g. in CSV as well.
- 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.
- Times are given as a UTC string or MJD float. This is preferred over the use of mission elapsed time (MET), because MET requires a reference timepoint stored in header keywords `MJDREFI` and `MJDREFF`, and we felt that having a simpler table format here that doesn't require a header would be nice.
- 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: *IACT event lists*
- `gti` - see format spec: TODO
- `aeff_2d` - see format spec: *aeff_2d format*
- `edisp_2d` - see format spec: *edisp_2d format*
- `psf_table` - see format spec: *psf_table format*
- `psf_3gauss` - see format spec: *psf_3gauss format*
- `psf_king` - see format spec: *psf_king format*
- `psf_gtpsf` – see format spec: *gtpsf format*
- `bkg_2d` - see format spec: *bkg_2d format*
- `bkg_3d` - see format spec: *bkg_3d format*

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.

Future ideas

- Not required columns are for future usage when downloading and syncing files with a server.
- We keep in mind to incoorporate "CHUNK_ID" column to support splitting of runs into chunks.

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 *IACT event lists* and 2D *IACT IRFs* 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 Source Models

What UCDs could spectral models use?

The file `` ` ``spectral_models.yaml defines a minimum specification for the names of the models, the name of the parameters and the minimum properties these parameters should have.

1.6.1 Spectral models

TODO: describe

```
NewUCDs:
    - model
    - model.param
    - model.spectral

SpectralModel:
    name:
        type: str
        description: Name
        ucd:
    norm:
        type: float
        description: Global normalization
        ucd: phot.flux.density

PowerLaw:
    type: SpectralModel
    reference:
        type: float
        description: Value at which normalization is defined.
        ucd: [em.wl, em.energ, em.freq]
    index:
        type: float
        description: Spectral index
        ucd: spect.index

BrokenPowerLaw:
    type: SpectralModel
    index1:
        type: float
        description: Spectral index below the break.
        ucd: spect.index
    index2:
        type: float
        description: Spectral index above the break.
        ucd: spect.index
    break:
        type: float
```

(continues on next page)

(continued from previous page)

```
        description: Value at which spectral index changes.
        ucd: [em.wl, em.energ, em.freq]

LogParabola:
    type: SpectralModel
    reference:
        type: float
        description: Value at which normalization is defined.
        ucd: [em.wl, em.energ, em.freq]
    alpha:
        type: float
        description: Spectral index at break value.
        ucd: spect.index
    beta:
        type: float
        description: Rate of spectral index chane.
        ucd: spect.index
    break:
        type: float
        description: Value at which alpha is defined.
        ucd: [em.wl, em.energ, em.freq]

ExponentialCutoffPowerLaw:
    type: SpectralModel
    reference:
        type: float
        description: Value at which normalization is defined.
        ucd: [em.wl, em.energ, em.freq]
    index:
        type: float
        description: Spectral index
        ucd: spect.index
    cutoff:
        type: float
        description: Value of the exponential cutoff.
        ucd: [em.wl, em.energ, em.freq]
    beta:
        type: float
        description: Hardness of the exponential cutoff.
        ucd:

ExponentialCutoffBrokenPowerLaw:
    type: SpectralModel
    reference:
        type: float
        description: Value at which normalization is defined.
        ucd: [em.wl, em.energ, em.freq]
    index1:
        type: float
        description: Spectral index below the break.
        ucd: spect.index
    index2:
        type: float
        description: Spectral index above the break.
        ucd: spect.index
    break:
        type: float
```

(continues on next page)

(continued from previous page)

```
        description: Value at which spectral index changes.
        ucd: [em.wl, em.energ, em.freq]
    cutoff:
        type: float
        description: Value of the exponential cutoff.
        ucd: [em.wl, em.energ, em.freq]
    beta:
        type: float
        description: Hardness of the exponential cutoff.
        ucd:
```

1.6.2 Spatial models

TODO: describe

```
NewUCDs:
    - model
    - model.param
    - model.spectral

SpectralModel:
    name:
        type: str
        description: Name
        ucd:
    norm:
        type: float
        description: Global normalization
        ucd: phot.flux.density

PowerLaw:
    type: SpectralModel
    reference:
        type: float
        description: Value at which normalization is defined.
        ucd: [em.wl, em.energ, em.freq]
    index:
        type: float
        description: Spectral index
        ucd: spect.index

BrokenPowerLaw:
    type: SpectralModel
    index1:
        type: float
        description: Spectral index below the break.
        ucd: spect.index
    index2:
        type: float
        description: Spectral index above the break.
        ucd: spect.index
    break:
        type: float
        description: Value at which spectral index changes.
        ucd: [em.wl, em.energ, em.freq]
```

(continues on next page)

(continued from previous page)

```

LogParabola:
    type: SpectralModel
    reference:
        type: float
        description: Value at which normalization is defined.
        ucd: [em.wl, em.energ, em.freq]
    alpha:
        type: float
        description: Spectral index at break value.
        ucd: spect.index
    beta:
        type: float
        description: Rate of spectral index chane.
        ucd: spect.index
    break:
        type: float
        description: Value at which alpha is defined.
        ucd: [em.wl, em.energ, em.freq]

ExponentialCutoffPowerLaw:
    type: SpectralModel
    reference:
        type: float
        description: Value at which normalization is defined.
        ucd: [em.wl, em.energ, em.freq]
    index:
        type: float
        description: Spectral index
        ucd: spect.index
    cutoff:
        type: float
        description: Value of the exponential cutoff.
        ucd: [em.wl, em.energ, em.freq]
    beta:
        type: float
        description: Hardness of the exponential cutoff.
        ucd:

ExponentialCutoffBrokenPowerLaw:
    type: SpectralModel
    reference:
        type: float
        description: Value at which normalization is defined.
        ucd: [em.wl, em.energ, em.freq]
    index1:
        type: float
        description: Spectral index below the break.
        ucd: spect.index
    index2:
        type: float
        description: Spectral index above the break.
        ucd: spect.index
    break:
        type: float
        description: Value at which spectral index changes.
        ucd: [em.wl, em.energ, em.freq]
```

(continues on next page)

(continued from previous page)

```
    cutoff:
        type: float
        description: Value of the exponential cutoff.
        ucd: [em.wl, em.energ, em.freq]
    beta:
        type: float
        description: Hardness of the exponential cutoff.
        ucd:
```

Other references:

- http://docs.astropy.org/en/stable/modeling/
- http://fermi.gsfc.nasa.gov/ssc/data/analysis/scitools/source_models.html

1.7 High-level results

Here we describe a few data format specs for high-level 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 Flux points

TODO. See https://github.com/open-gamma-ray-astro/gamma-astro-data-formats/issues/6

1.7.2 Light curve

TODO: write a spec. how to store light curves in a table.

- Are there any standard formats for light curves? (e.g. a VO spec?)
- See monthly light-curves in the Fermi catalogs
- http://fermi.gsfc.nasa.gov/ssc/data/access/lat/4yr_catalog/ap_lcs.php
- See https://github.com/gammapy/gammapy/issues/281