overview

November 13, 2022

# Data structures

0.1 Introduction

This is a getting started tutorial for Gammapy.

In this tutorial we will use the Third Fermi-LAT Catalog of High-Energy Sources (3FHL) catalog, corresponding event list and images to learn how to work with some of the central Gammapy data structures.

We will cover the following topics:

- Sky maps
  - We will learn how to handle image based data with gammapy using a Fermi-LAT 3FHL example image. We will work with the following classes:
    - `~gammapy.maps.WcsNDMap`
    - astropy.coordinates.SkyCoord
    - numpy.ndarray
- Event lists
  - We will learn how to handle event lists with Gammapy. Important for this are the following classes:
    - `~gammapy.data.EventList`
    - astropy.table.Table
- Source catalogs
  - We will show how to load source catalogs with Gammapy and explore the data using the following classes:
    - `~gammapy.catalog.SourceCatalog`, specifically `~gammapy.catalog.SourceCatalog3FHL`
    - astropy.table.Table
- Spectral models and flux points
  - We will pick an example source and show how to plot its spectral model and flux points. For this we will use the following classes:
    - `~gammapy.modeling.models.SpectralModel`, specifically the `~gammapy.modeling.models.PowerLaw2SpectralModel`
    - `~gammapy.estimators.FluxPoints`
    - astropy.table.Table

Back to Top

0.2 Setup

Important: to run this tutorial the environment variable `GAMMAPY_DATA` must be defined and point to the directory on your machine where the datasets needed are placed. To check whether your setup is correct you can execute the following cell:

[1]:
```
import os

path = os.path.expandvars("$GAMMAPY_DATA")

if not os.path.exists(path):
    raise Exception("gammapy-data repository not found!")
else:
    print("Great your setup is correct!")
```

```
Great your setup is correct!
```

In case you encounter an error, you can un-comment and execute the following cell to continue. But we recommend to set up your environment correctly as described in getting started after you are done with this notebook.

[2]:
```
# os.environ['GAMMAPY_DATA'] = os.path.join(os.getcwd(), '..')
```

Now we can continue with the usual IPython notebooks and Python imports:

[3]:
```
%matplotlib inline
import matplotlib.pyplot as plt
```

[4]:
```
# defines, converts between, and performs arithmetic with physical quantities␣
↪(meters, seconds, Hz, etc) and logarithmic units (magnitude and decibel)
import astropy.units as u
# representation, manipulation, and transformation between systems of celestial␣
↪coordinates.
from astropy.coordinates import SkyCoord
```

Back to Top

## Sky Maps

The `~gammapy.maps` package contains classes to work with sky images and cubes.

In this section, we will use a simple 2D sky image and will learn how to:

- Read sky images from FITS files
- Smooth images
- Plot images
- Cutout parts from images

[5]:
```
from gammapy.maps import Map

gc_3fhl = Map.read("$GAMMAPY_DATA/fermi-3fhl-gc/fermi-3fhl-gc-counts.fits.gz")
```

The image is a `~gammapy.maps.WcsNDMap` object:

```
[6]: gc_3fhl
```

```
[6]: WcsNDMap

        geom  : WcsGeom
        axes  : ['lon', 'lat']
        shape : (400, 200)
        ndim  : 2
        unit  :
        dtype : >i8
```

The shape of the image is 400 x 200 pixel and it is defined using a cartesian projection in galactic coordinates.

The `geom` attribute is a `~gammapy.maps.WcsGeom` object:

```
[7]: gc_3fhl.geom
```

```
[7]: WcsGeom

        axes       : ['lon', 'lat']
        shape      : (400, 200)
        ndim       : 2
        frame      : galactic
        projection : CAR
        center     : 0.0 deg, 0.0 deg
        width      : 20.0 deg x 10.0 deg
        wcs ref    : 0.0 deg, 0.0 deg
```

Let's take a closer look a the `.data` attribute:

```
[8]: gc_3fhl.data
```

```
[8]: array([[0, 0, 0, …, 0, 0, 0],
           [0, 0, 0, …, 0, 0, 0],
           [0, 0, 0, …, 0, 0, 0],
           …,
           [0, 0, 0, …, 0, 0, 1],
           [0, 0, 0, …, 0, 0, 0],
           [0, 0, 0, …, 0, 0, 1]])
```

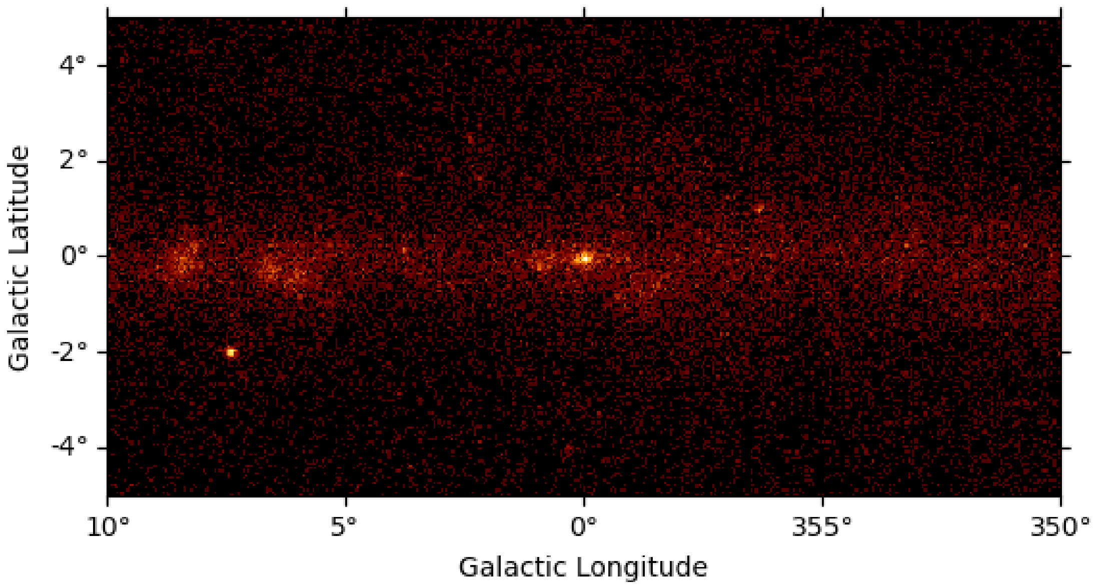
That looks familiar! It just an *ordinary* 2 dimensional numpy array, which means you can apply any known numpy method to it:

```
[9]: print(f"Total number of counts in the image: {gc_3fhl.data.sum():.0f}")
```

```
Total number of counts in the image: 32684
```

To show the image on the screen we can use the `plot` method. It basically calls plt.imshow, passing the `gc_3fhl.data` attribute but in addition handles axis with world coordinates using astropy.visualization.wcsaxes and defines some defaults for nicer plots (e.g. the colormap 'afmhot'):

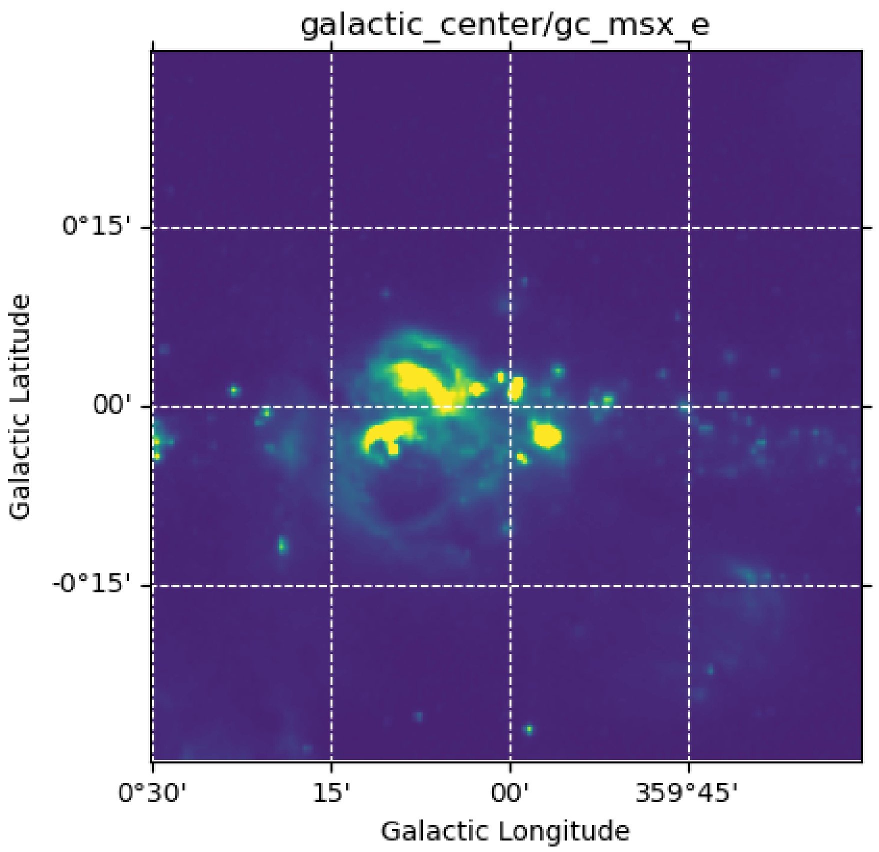
[10]:
```
gc_3fhl.plot(stretch="sqrt");
```



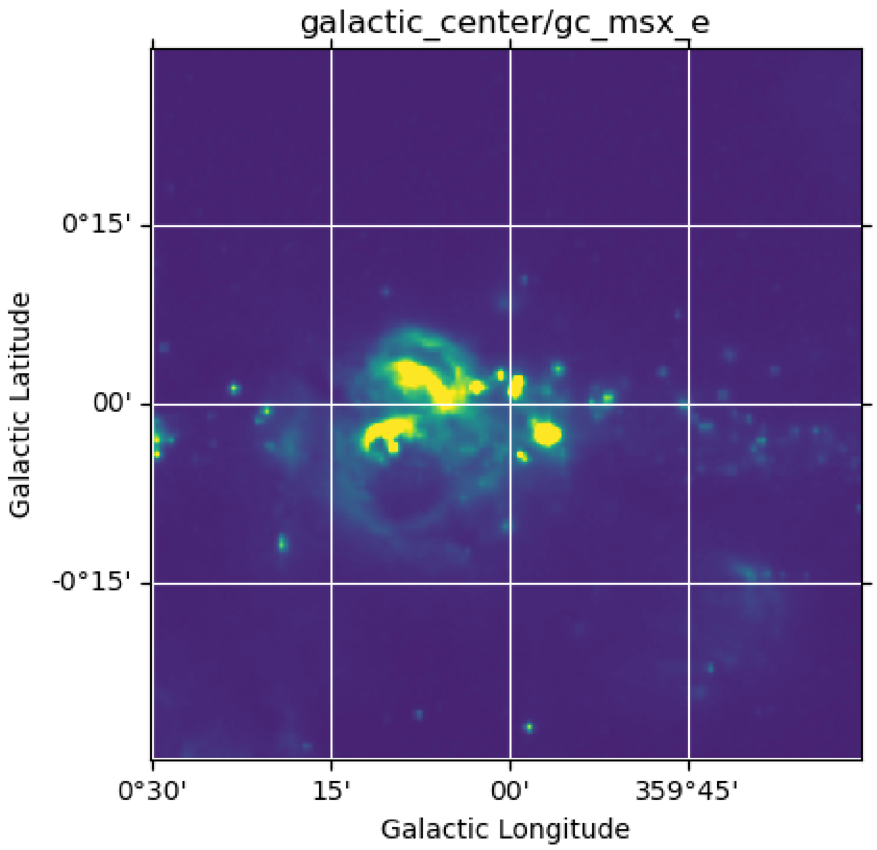
Examples of plotting images with the WCSAxes package:

[11]:
```
from astropy.wcs import WCS
from astropy.io import fits
from astropy.utils.data import get_pkg_data_filename

filename = get_pkg_data_filename('galactic_center/gc_msx_e.fits')

hdu = fits.open(filename)[0]
wcs = WCS(hdu.header)
```

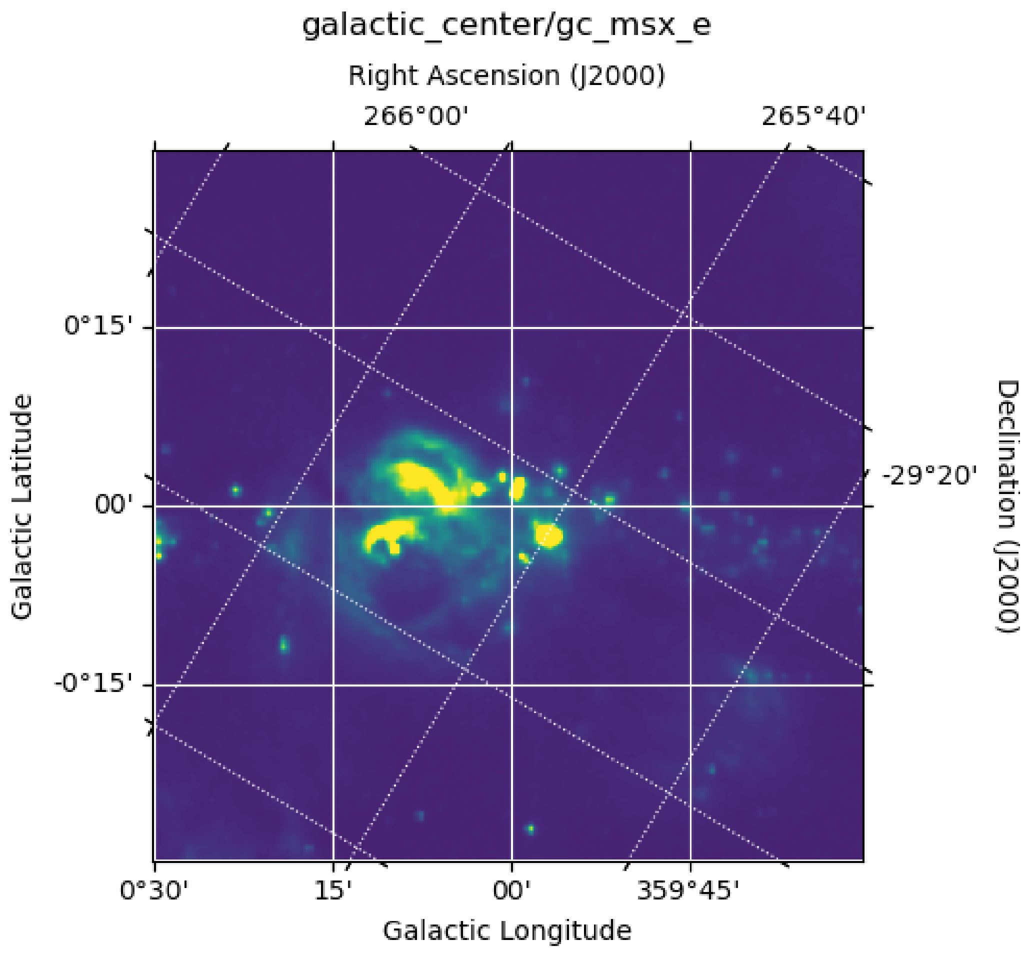
[12]:
```
# A simple example
plt.subplot(projection=wcs)

plt.imshow(hdu.data, vmin=-2.e-5, vmax=2.e-4, origin='lower')

plt.title("galactic_center/gc_msx_e")

plt.grid(color='white', ls='--')

plt.xlabel('Galactic Longitude')
plt.ylabel('Galactic Latitude')
```



[13]:
```
# For example, using the partially object-oriented interface
ax = plt.subplot(projection=wcs)

plt.title("galactic_center/gc_msx_e")

ax.imshow(hdu.data, vmin=-2.e-5, vmax=2.e-4, origin='lower')

ax.grid(color='white', ls='solid')

ax.set_xlabel('Galactic Longitude')
ax.set_ylabel('Galactic Latitude')
```

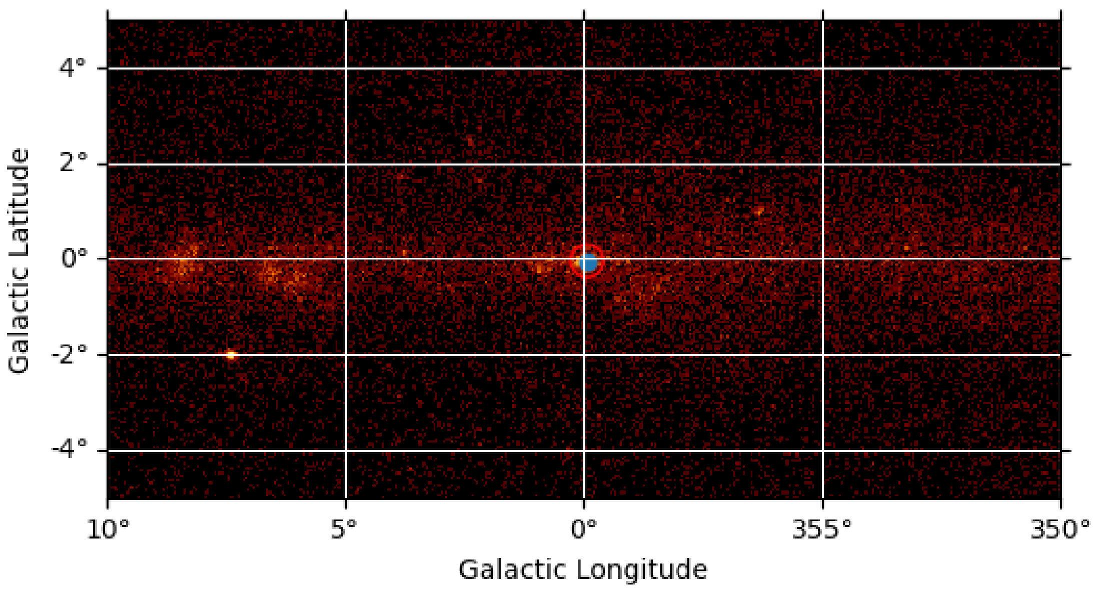


[14]:
```
# For example, using advanced functionalities
ax = plt.subplot(projection=wcs, label='overlays')

plt.title("galactic_center/gc_msx_e", y = 1.14)

ax.imshow(hdu.data, vmin=-2.e-5, vmax=2.e-4, origin='lower')

ax.coords.grid(True, color='white', ls='solid')

ax.coords[0].set_axislabel('Galactic Longitude')
ax.coords[1].set_axislabel('Galactic Latitude')

overlay = ax.get_coords_overlay('fk5')
overlay.grid(color='white', ls='dotted')
overlay[0].set_axislabel('Right Ascension (J2000)')
overlay[1].set_axislabel('Declination (J2000)')
```



```
[15]: ax = gc_3fhl.plot(stretch="sqrt");
      #ax = plt.subplot(projection=wcs)
      #ax.imshow(hdu.data, vmin=-2.e-5, vmax=2.e-4, origin='lower')
      ax.grid(color='white', ls='solid')
      ax.set_xlabel('Galactic Longitude')
      ax.set_ylabel('Galactic Latitude')
      #ax.scatter([0], [1], transform=ax.get_transform('world'))
      ax.scatter(359.94423568, -0.04616002, transform=ax.get_transform('galactic'),␣
       ↪s=150,
                 edgecolor='red', facecolor='none')
      ax.scatter([359.94423568], [-0.04616002], transform=ax.
       ↪get_transform('galactic'))
```

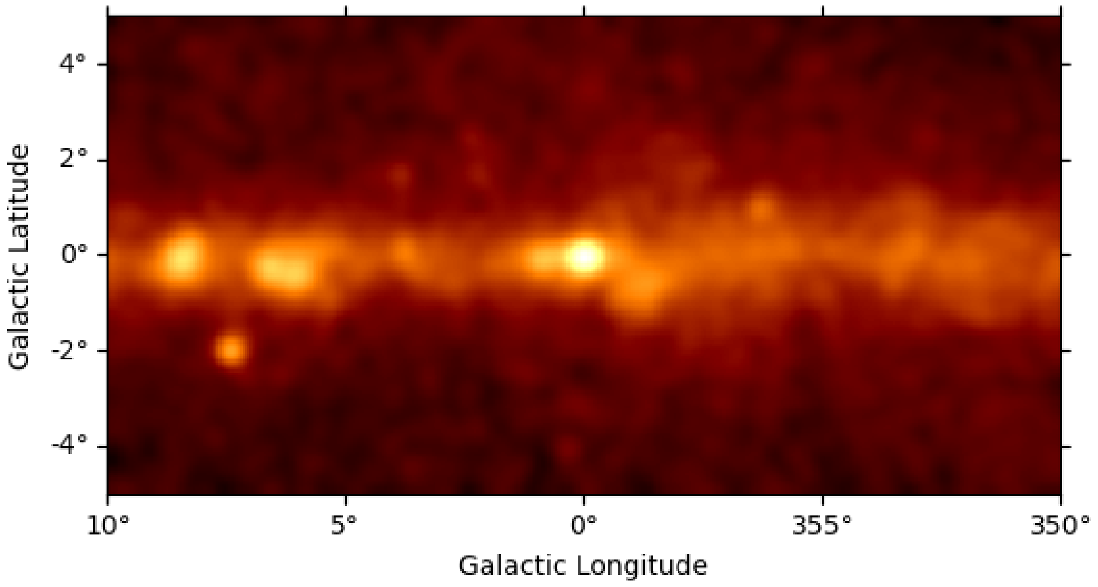
```
[15]: <matplotlib.collections.PathCollection at 0x7f6782b17fa0>
```



To make the structures in the image more visible we will smooth the data using a Gaussian kernel.

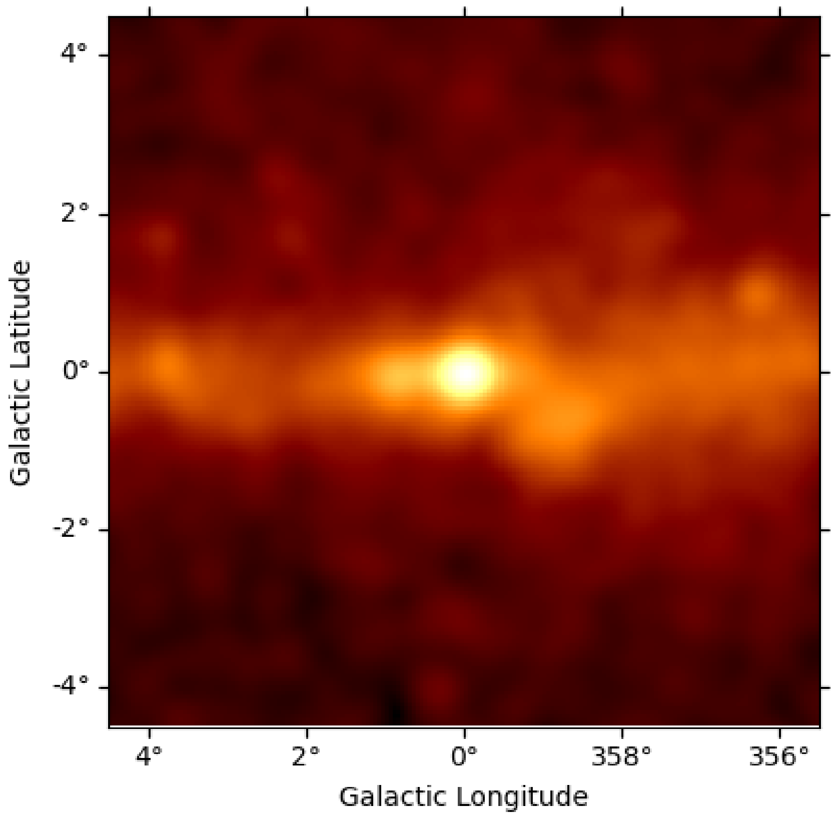
[16]:
```
gc_3fhl_smoothed = gc_3fhl.smooth(kernel="gauss", width=0.2 * u.deg)
```

[17]:
```
gc_3fhl_smoothed.plot(stretch="sqrt");
```



The smoothed plot already looks much nicer, but still the image is rather large. As we are mostly interested in the inner part of the image, we will cut out a quadratic region of the size 9 deg x 9 deg around Vela. Therefore we use ~gammapy.maps.Map.cutout to make a cutout map:

[18]:
```python
# define center and size of the cutout region
center = SkyCoord(0, 0, unit="deg", frame="galactic")
gc_3fhl_cutout = gc_3fhl_smoothed.cutout(center, 9 * u.deg)
gc_3fhl_cutout.plot(stretch="sqrt");
```



For a more detailed introduction to ~gammapy.maps, take a look a the maps.ipynb notebook.

0.2.1 Exercises

- Add a marker and circle at the position of Sag A* (you can find examples in astropy.visualization.wcsaxes).

[ ]:
```

```

Back to Top

## Event lists

Almost any high level gamma-ray data analysis starts with the raw measured counts data, which is stored in event lists. In Gammapy event lists are represented by the ~gammapy.data.EventList class.

In this section we will learn how to:

- Read event lists from FITS files
- Access and work with the `EventList` attributes such as `.table` and `.energy`
- Filter events lists using convenience methods

Let's start with the import from the `~gammapy.data` submodule:

[19]:
```
from gammapy.data import EventList
```

Very similar to the sky map class an event list can be created, by passing a filename to the `~gammapy.data.EventList.read()` method:

[20]:
```
events_3fhl = EventList.read(
    "$GAMMAPY_DATA/fermi-3fhl-gc/fermi-3fhl-gc-events.fits.gz"
)
```

You can find examples in `gammapy.data.EventList`.

[21]:
```
# filename = "$GAMMAPY_DATA/hess-dl3-dr1/data/hess_dl3_dr1_obs_id_023523.fits.
 ↪gz"
# events = EventList.read(filename)

# #Plot the offset^2 distribution wrt. the observation pointing position
# #(this is a commonly used plot to check the background spatial distribution):
# events.plot_offset2_distribution()
# Plot the offset^2 distribution wrt. the Crab pulsar position (this is
# commonly used to check both the gamma-ray signal and the background
# spatial distribution):
```

This time the actual data is stored as an astropy.table.Table object. It can be accessed with `.table` attribute:

[22]:
```
events_3fhl
```

[22]:
```
<gammapy.data.event_list.EventList at 0x7f6782a0c7f0>
```

[23]:
```
print(events_3fhl)
```

```
EventList
---------

  Instrument       : LAT
  Telescope        : GLAST
  Obs. ID          :

  Number of events : 32843
  Event rate       : 0.000 1 / s
```

```
Time start       : 54682.65603222222
Time stop        : 57236.96833546296

Min. energy      : 1.00e+04 MeV
Max. energy      : 1.92e+06 MeV
Median energy    : 1.58e+04 MeV
```

[24]:
```
events_3fhl.table
```

[24]:
```
<Table length=32843>
  ENERGY      RA        DEC      … DIFRSP3 DIFRSP4
   MeV        deg       deg      …
 float32    float32   float32    … float32 float32
--------- --------- ---------- … ------- -------
12186.642 260.45935 -33.553337 …     0.0     0.0
25496.598 261.37506 -34.395004 …     0.0     0.0
      …         …        … …     …       …
18465.783 266.39728 -29.105953 …     0.0     0.0
 14457.25 262.72217 -34.388405 …     0.0     0.0
```

Click here to see the descriptions of the reconstructed parameters.

TIPGet help on the available readers for `Table` using the`help()` method:

```
Table.read.help()  # Get help reading Table and list supported formats
Table.read.help('fits')  # Get detailed help on Table FITS reader
Table.read.list_formats()  # Print list of available formats
```

You can do *len* over event_3fhl.table to find the total number of events.

[25]:
```
len(events_3fhl.table)
```

[25]:
```
32843
```

[26]:
```
events_3fhl.table.info
```

[26]:
```
<Table length=32843>
        name         dtype  shape unit
------------------- ------- ----- ----
             ENERGY float32        MeV
                 RA float32        deg
                DEC float32        deg
                  L float32        deg
                  B float32        deg
              THETA float32        deg
                PHI float32        deg
       ZENITH_ANGLE float32        deg
```

```
EARTH_AZIMUTH_ANGLE float32        deg
               TIME float64          s
           EVENT_ID   int32
             RUN_ID   int32
       RECON_VERSION   int16
       CALIB_VERSION   int16  (3,)
         EVENT_CLASS    bool (32,)
          EVENT_TYPE    bool (32,)
     CONVERSION_TYPE   int16
           LIVETIME float64          s
            DIFRSP0 float32
            DIFRSP1 float32
            DIFRSP2 float32
            DIFRSP3 float32
            DIFRSP4 float32
```

And we can access any other attribute of the `Table` object as well:

[27]:
```
events_3fhl.table.colnames
```

[27]:
```
['ENERGY',
 'RA',
 'DEC',
 'L',
 'B',
 'THETA',
 'PHI',
 'ZENITH_ANGLE',
 'EARTH_AZIMUTH_ANGLE',
 'TIME',
 'EVENT_ID',
 'RUN_ID',
 'RECON_VERSION',
 'CALIB_VERSION',
 'EVENT_CLASS',
 'EVENT_TYPE',
 'CONVERSION_TYPE',
 'LIVETIME',
 'DIFRSP0',
 'DIFRSP1',
 'DIFRSP2',
 'DIFRSP3',
 'DIFRSP4']
```

For convenience we can access the most important event parameters as properties on the `EventList` objects. The attributes will return corresponding Astropy objects to represent the data, such as astropy.units.Quantity, astropy.coordinates.SkyCoord or astropy.time.Time objects:

[28]:
```
events_3fhl.energy.to("GeV")
```

[28]: [12.186643, 25.496599, 15.621499, …, 32.095707, 18.465784, 14.457251] GeV

[29]:
```
events_3fhl.table["ENERGY"]=events_3fhl.energy.to("GeV")
# t[colname][:] = value
events_3fhl.table["ENERGY"]
```

[29]:
```
<Column name='ENERGY' dtype='float32' unit='GeV' length=32843>
12.186643
  25.4966
15.621499
12.816321
18.988388
      …
13.133865
32.095707
18.465784
14.457251
```

[30]:
```
events_3fhl.energy.unit
```

[30]: GeV

[31]:
```
events_3fhl.galactic
# events_3fhl.radec
```

[31]:
```
<SkyCoord (Galactic): (l, b) in deg
    [(353.36228879,  1.75408483), (353.09562941,  0.6522806 ),
     (353.05628243,  2.44528685), …, (359.10295505, -0.1359316 ),
     (359.85157506, -0.08269984), (353.71795506, -0.26883694)]>
```

[32]:
```
events_3fhl.time
```

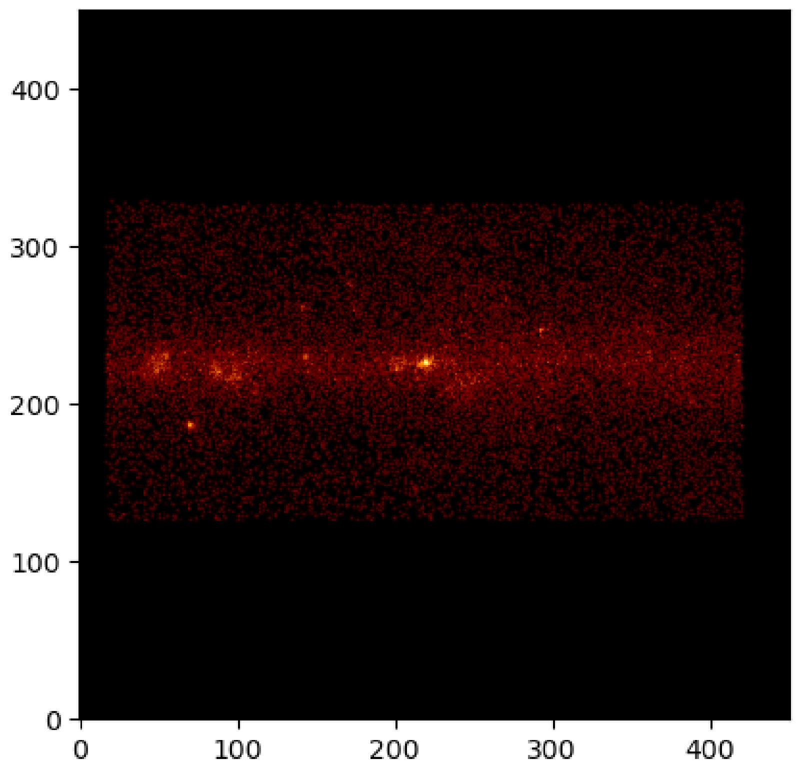
[32]:
```
<Time object: scale='tt' format='mjd' value=[54682.82946153 54682.89243456
54682.89709472 … 57236.75267735
 57233.37455141 57233.44802852]>
```

There is also some convenience to plot the events:

[33]:
```
help(EventList)
```

```
Help on class EventList in module gammapy.data.event_list:

class EventList(builtins.object)
 |  EventList(table)
 |
 |  Event list.
 |
 |  Event list data is stored as ``table`` (`~astropy.table.Table`) data member.
 |
 |  The most important reconstructed event parameters
 |  are available as the following columns:
 |
 |  - ``TIME`` - Mission elapsed time (sec)
 |  - ``RA``, ``DEC`` - ICRS system position (deg)
 |  - ``ENERGY`` - Energy (usually MeV for Fermi and TeV for IACTs)
 |
 |  Note that ``TIME`` is usually sorted, but sometimes it is not.
 |  E.g. when simulating data, or processing it in certain ways.
 |  So generally any analysis code should assume ``TIME`` is not sorted.
 |
 |  Other optional (columns) that are sometimes useful for high level analysis:
 |
 |  - ``GLON``, ``GLAT`` - Galactic coordinates (deg)
 |  - ``DETX``, ``DETY`` - Field of view coordinates (deg)
 |
 |  Note that when reading data for analysis you shouldn't use those
 |  values directly, but access them via properties which create objects
 |  of the appropriate class:
 |
 |  - `time` for ``TIME``
 |  - `radec` for ``RA``, ``DEC``
 |  - `energy` for ``ENERGY``
 |  - `galactic` for ``GLON``, ``GLAT``
 |
 |  Parameters
 |  ----------
 |  table : `~astropy.table.Table`
 |      Event list table
 |
 |  Examples
 |  --------
 |  >>> from gammapy.data import EventList
 |  >>> events = EventList.read("$GAMMAPY_DATA/cta-1dc/data/baseline/gps/gps_bas
eline_110380.fits")
 |  >>> print(events)
 |  EventList
 |  ---------
 |  <BLANKLINE>
 |    Instrument       : None
 |    Telescope        : CTA
 |    Obs. ID          : 110380
 |  <BLANKLINE>
 |    Number of events : 106217
 |    Event rate       : 59.273 1 / s
 |  <BLANKLINE>
 |     Time start       : 59235.5
 |     Time stop        : 59235.52074074074
 |   <BLANKLINE>
 |     Min. energy      : 3.00e-02 TeV
 |     Max. energy      : 1.46e+02 TeV
 |     Median energy    : 1.02e-01 TeV
 |   <BLANKLINE>
 |     Max. offset      : 5.0 deg
 |   <BLANKLINE>
 |
 |   Methods defined here:
 |
 |   __init__(self, table)
 |       Initialize self.  See help(type(self)) for accurate signature.
 |
 |   __str__(self)
 |       Return str(self).
 |
 |   check(self, checks='all')
 |       Run checks.
 |
 |       This is a generator that yields a list of dicts.
 |
 |   map_coord(self, geom)
 |       Event map coordinates for a given geometry.
 |
 |       Parameters
 |       ----------
 |       geom : `~gammapy.maps.Geom`
 |           Geometry
 |
 |       Returns
 |       -------
 |       coord : `~gammapy.maps.MapCoord`
 |           Coordinates
 |
 |   peek(self, allsky=False)
 |       Quick look plots.
 |
 |       Parameters
 |       ----------
 |       allsky : bool
 |           Whether to look at the events allsky
 |
 |   plot_energy(self, ax=None, **kwargs)
 |       Plot counts as a function of energy.
 |
 |       Parameters
 |      ----------
 |      ax : `~matplotlib.axes.Axes` or None
 |          Axes
 |      **kwargs : dict
 |          Keyword arguments passed to `~matplotlib.pyplot.hist`
 |
 |      Returns
 |      -------
 |      ax : `~matplotlib.axes.Axes` or None
 |          Axes
 |
 |  plot_energy_offset(self, ax=None, center=None, **kwargs)
 |      Plot counts histogram with energy and offset axes
 |
 |      Parameters
 |      ----------
 |      ax : `~matplotlib.pyplot.Axis`
 |          Plot axis
 |      center : `~astropy.coordinates.SkyCoord`
 |          Sky coord from which offset is computed
 |      **kwargs : dict
 |          Keyword arguments forwarded to `~matplotlib.pyplot.pcolormesh`
 |
 |      Returns
 |      -------
 |      ax : `~matplotlib.pyplot.Axis`
 |          Plot axis
 |
 |  plot_image(self, ax=None, allsky=False)
 |      Quick look counts map sky plot.
 |
 |      Parameters
 |      ----------
 |      ax : `~matplotlib.pyplot.Axes`
 |          Axes to plot on.
 |      allsky :  bool,
 |          Whether to plot on an all sky geom
 |
 |  plot_offset2_distribution(self, ax=None, center=None, **kwargs)
 |      Plot offset^2 distribution of the events.
 |
 |      The distribution shown in this plot is for this quantity::
 |
 |          offset = center.separation(events.radec).deg
 |          offset2 = offset ** 2
 |
 |      Note that this method is just for a quicklook plot.
 |
 |      If you want to do computations with the offset or offset^2 values, you
can
 |      use the line above. As an example, here's how to compute the 68% event
 |      containment radius using `numpy.percentile`::
 |
 |          import numpy as np
 |          r68 = np.percentile(offset, q=68)
 |
 |      Parameters
 |      ----------
 |      ax : `~matplotlib.axes.Axes` (optional)
 |          Axes
 |      center : `astropy.coordinates.SkyCoord`
 |          Center position for the offset^2 distribution.
 |          Default is the observation pointing position.
 |      **kwargs :
 |          Extra keyword arguments are passed to `~matplotlib.pyplot.hist`.
 |
 |      Returns
 |      -------
 |      ax : `~matplotlib.axes.Axes`
 |          Axes
 |
 |      Examples
 |      --------
 |      Load an example event list:
 |
 |      >>> from gammapy.data import EventList
 |      >>> from astropy import units as u
 |      >>> events = EventList.read('$GAMMAPY_DATA/hess-
dl3-dr1/data/hess_dl3_dr1_obs_id_023523.fits.gz')
 |
 |      >>> #Plot the offset^2 distribution wrt. the observation pointing
position
 |      >>> #(this is a commonly used plot to check the background spatial
distribution):
 |      >>> events.plot_offset2_distribution() # doctest: +SKIP
 |      Plot the offset^2 distribution wrt. the Crab pulsar position
 |      (this is commonly used to check both the gamma-ray signal and the
background spatial distribution):
 |
 |      >>> import numpy as np
 |      >>> from astropy.coordinates import SkyCoord
 |      >>> center = SkyCoord(83.63307, 22.01449, unit='deg')
 |      >>> bins = np.linspace(start=0, stop=0.3 ** 2, num=30) * u.deg ** 2
 |      >>> events.plot_offset2_distribution(center=center, bins=bins) #
doctest: +SKIP
 |
 |      Note how we passed the ``bins`` option of `matplotlib.pyplot.hist` to
control the histogram binning,
 |      in this case 30 bins ranging from 0 to (0.3 deg)^2.
 |
 |  plot_time(self, ax=None, **kwargs)
 |      Plots an event rate time curve.
 |
 |      Parameters
 |      ----------
 |      ax : `~matplotlib.axes.Axes` or None
 |          Axes
 |      **kwargs : dict
 |          Keyword arguments passed to `~matplotlib.pyplot.errorbar`
 |
 |      Returns
 |      -------
 |      ax : `~matplotlib.axes.Axes`
 |          Axes
 |
 |  select_energy(self, energy_range)
 |      Select events in energy band.
 |
 |      Parameters
 |      ----------
 |      energy_range : `~astropy.units.Quantity`
 |          Energy range ``[energy_min, energy_max)``
 |
 |      Returns
 |      -------
 |      event_list : `EventList`
 |          Copy of event list with selection applied.
 |
 |      Examples
 |      --------
 |      >>> from astropy import units as u
 |      >>> from gammapy.data import EventList
 |      >>> event_list =
EventList.read('$GAMMAPY_DATA/fermi_3fhl/fermi_3fhl_events_selected.fits.gz')
 |      >>> energy_range =[1, 20] * u.TeV
 |      >>> event_list = event_list.select_energy(energy_range=energy_range)
 |
 |  select_mask(self, mask)
 |      Select events inside a mask (`EventList`).
 |
 |      Parameters
 |      ----------
 |      mask : `~gammapy.maps.Map`
 |          Mask
 |
 |      Returns
 |      -------
 |      event_list : `EventList`
 |          Copy of event list with selection applied.
 |
 |      Examples
 |      --------
 |      >>> from gammapy.data import EventList
 |      >>> from gammapy.maps import WcsGeom, Map
 |      >>> geom = WcsGeom.create(skydir=(0,0), width=(4, 4), frame="galactic")
 |      >>> mask = geom.region_mask("galactic;circle(0, 0, 0.5)")
 |      >>> events = EventList.read("$GAMMAPY_DATA/cta-1dc/data/baseline/gps/gps
_baseline_110380.fits")
 |      >>> masked_event = events.select_mask(mask)
 |      >>> len(masked_event.table)
 |      5594
 |
 |  select_offset(self, offset_band)
 |      Select events in offset band.
 |
 |      Parameters
 |      ----------
 |      offset_band : `~astropy.coordinates.Angle`
 |          offset band ``[offset_min, offset_max)``
 |
 |      Returns
 |      -------
 |      event_list : `EventList`
 |          Copy of event list with selection applied.
 |
 |      Examples
 |      --------
 |      >>> from gammapy.data import EventList
 |      >>> import astropy.units as u
 |      >>> events = EventList.read("$GAMMAPY_DATA/cta-1dc/data/baseline/gps/gps
_baseline_110380.fits")
 |      >>> selected_events = events.select_offset([0.3, 0.9]*u.deg)
 |      >>> len(selected_events.table)
 |      12688
 |
 |  select_parameter(self, parameter, band)
 |      Select events with respect to a specified parameter.
 |
 |      Parameters
 |      ----------
 |      parameter : str
 |          Parameter used for the selection. Must be present in `self.table`.
 |      band : tuple or `astropy.units.Quantity`
 |          Min and max value for the parameter to be selected (min <= parameter
< max).
 |          If parameter is not dimensionless you have to provide a Quantity.
 |
 |      Returns
 |      -------
 |      event_list : `EventList`
 |          Copy of event list with selection applied.
 |
 |      Examples
 |      --------
 |      >>> from astropy import units as u
 |      >>> from gammapy.data import EventList
 |      >>> event_list =
EventList.read('$GAMMAPY_DATA/fermi_3fhl/fermi_3fhl_events_selected.fits.gz')
 |      >>> zd = (0, 30) * u.deg
 |      >>> event_list = event_list.select_parameter(parameter='ZENITH_ANGLE',
band=zd)
 |      >>> print(len(event_list.table))
 |      123944
 |
 |  select_rad_max(self, rad_max, position=None)
 |      Select energy dependent offset
 |
 |      Parameters
 |      ----------
 |      rad_max : `~gamapy.irf.RadMax2D`
 |          Rad max definition
 |      position : `~astropy.coordinates.SkyCoord`
 |          Center position. By default the pointing position is used.
 |
 |      Returns
 |      -------
 |      event_list : `EventList`
 |          Copy of event list with selection applied.
 |
 |  select_region(self, regions, wcs=None)
 |      Select events in given region.
 |
 |      Parameters
 |      ----------
 |      regions : str, `~regions.Region` or list of `~regions.Region`
 |          Region or list of regions (pixel or sky regions accepted).
 |          A region can be defined as a string ind DS9 format as well.
 |          See http://ds9.si.edu/doc/ref/region.html for details.
 |      wcs : `~astropy.wcs.WCS`
 |          World coordinate system transformation
 |
 |      Returns
 |      -------
 |      event_list : `EventList`
 |          Copy of event list with selection applied.
 |
 |  select_row_subset(self, row_specifier)
 |      Select table row subset.
 |
 |      Parameters
 |      ----------
 |      row_specifier : slice, int, or array of ints
 |          Specification for rows to select,
 |          passed on to ``self.table[row_specifier]``.
 |
 |      Returns
 |      -------
 |      event_list : `EventList`
 |          New event list with table row subset selected
 |
 |      Examples
 |      --------
 |
 |      >>> from gammapy.data import EventList
 |      >>> import numpy as np
 |      >>> events = EventList.read("$GAMMAPY_DATA/cta-1dc/data/baseline/gps/gps
_baseline_110380.fits")
 |      >>> #Use a boolean mask as ``row_specifier``:
 |      >>> mask = events.table['MC_ID'] == 1
 |      >>> events2 = events.select_row_subset(mask)
 |      >>> print(len(events2.table))
 |      97978
 |      >>> #Use row index array as ``row_specifier``:
 |      >>> idx = np.where(events.table['MC_ID'] == 1)[0]
 |      >>> events2 = events.select_row_subset(idx)
 |      >>> print(len(events2.table))
 |      97978
 |
 |  select_time(self, time_interval)
 |      Select events in time interval.
 |
 |      Parameters
 |      ----------
 |      time_interval : `astropy.time.Time`
 |          Start time (inclusive) and stop time (exclusive) for the selection.
 |
 |      Returns
 |      -------
 |      events : `EventList`
 |          Copy of event list with selection applied.
 |
 |  stack(self, other)
 |      Stack with another EventList in place.
 |
 |      Calls `~astropy.table.vstack`.
 |
 |      Parameters
 |      ----------
 |      other : `~gammapy.data.EventList`
 |          Event list to stack to self
 |
 |  to_table_hdu(self, format='gadf')
 |      Convert event list to a `~astropy.io.fits.BinTableHDU`
 |
 |      Parameters
 |      ----------
 |      format: str
 |          Output format, currently only "gadf" is supported
 |
 |      Returns
 |      -------
 |      hdu: `astropy.io.fits.BinTableHDU`
 |          EventList converted to FITS representation
 |
 |  write(self, filename, gti=None, overwrite=False, format='gadf')
 |      Write the event list to a FITS file.
 |
 |      If a GTI object is provided, it is saved into
 |      a second extension in the file.
 |
 |      Parameters
 |      ----------
 |      filename : `pathlib.Path`, str
 |          Filename
 |      gti : `~gammapy.data.GTI`
 |          Good Time Intervals object to save to the same file.
 |          Default is None.
 |      overwrite : bool
 |          Overwrite existing file?
 |      format : str, optional
 |          FITS format convention.  By default files will be written
 |          to the gamma-astro-data-formats (GADF) format.
 |
 |  ----------------------------------------------------------------------
 |  Class methods defined here:
 |
 |  from_stack(event_lists, **kwargs) from builtins.type
 |      Stack (concatenate) list of event lists.
 |
 |      Calls `~astropy.table.vstack`.
 |
 |      Parameters
 |      ----------
 |      event_lists : list
 |          list of `~gammapy.data.EventList` to stack
 |
 |  read(filename, **kwargs) from builtins.type
 |      Read from FITS file.
 |
 |      Format specification: :ref:`gadf:iact-events`
 |
 |      Parameters
 |      ----------
 |      filename : `pathlib.Path`, str
 |          Filename
 |
 |  ----------------------------------------------------------------------
 |  Readonly properties defined here:
 |
 |  altaz
 |      ALT / AZ position computed from RA / DEC
(`~astropy.coordinates.SkyCoord`).
 |
 |  altaz_frame
 |      ALT / AZ frame (`~astropy.coordinates.AltAz`).
 |
 |  altaz_from_table
 |      ALT / AZ position from table (`~astropy.coordinates.SkyCoord`).
 |
 |  energy
 |      Event energies (`~astropy.units.Quantity`).
 |
 |  galactic
 |      Event Galactic sky coordinates (`~astropy.coordinates.SkyCoord`).
 |
 |      Always computed from RA / DEC using Astropy.
 |
 |  galactic_median
 |      Median position in radec
 |
 |  is_pointed_observation
 |      Whether observation is pointed
 |
 |  observation_dead_time_fraction
 |      Dead-time fraction (float).
 |
 |      This is a keyword related to IACTs
 |      Defined as dead-time over observation time.
 |
 |      Dead-time is defined as the time during the observation
 |      where the detector didn't record events:
 |      http://en.wikipedia.org/wiki/Dead_time
 |      https://ui.adsabs.harvard.edu/abs/2004APh…22..285F
 |
 |      The dead-time fraction is used in the live-time computation,
 |      which in turn is used in the exposure and flux computation.
 |
 |  observation_live_time_duration
 |      Live-time duration in seconds (`~astropy.units.Quantity`).
 |
 |      The dead-time-corrected observation time.
 |
 |      - In Fermi-LAT it is automatically provided in the header of the event
list.
 |      - In IACTs is computed as ``t_live = t_observation * (1 - f_dead)``
 |
 |      where ``f_dead`` is the dead-time fraction.
 |
 |  observation_time_duration
 |      Observation time duration in seconds (`~astropy.units.Quantity`).
 |
 |      This is a keyword related to IACTs
 |      The wall time, including dead-time.
 |
 |  observation_time_start
 |      Observation start time (`~astropy.time.Time`).
 |
 |  observation_time_stop
 |      Observation stop time (`~astropy.time.Time`).
 |
 |  observatory_earth_location
 |      Observatory location (`~astropy.coordinates.EarthLocation`).
 |
 |  offset
 |      Event offset from the array pointing position
(`~astropy.coordinates.Angle`).
 |
 |  offset_from_median
 |      Event offset from the median position (`~astropy.coordinates.Angle`).
 |
 |  pointing_radec
 |      Pointing RA / DEC sky coordinates (`~astropy.coordinates.SkyCoord`).
 |
 |  radec
 |      Event RA / DEC sky coordinates (`~astropy.coordinates.SkyCoord`).
 |
 |  time
 |      Event times (`~astropy.time.Time`).
 |
 |      Notes
 |      -----
 |      Times are automatically converted to 64-bit floats.
 |      With 32-bit floats times will be incorrect by a few seconds
 |      when e.g. adding them to the reference time.
 |
 |  time_ref
 |      Time reference (`~astropy.time.Time`).
 |
 |  ----------------------------------------------------------------------
 |  Data descriptors defined here:
 |
 |  __dict__
 |      dictionary for instance variables (if defined)
 |
 |  __weakref__
 |      list of weak references to the object (if defined)
```

[34]:
```
events_3fhl.plot_image()
```



In addition `EventList` provides convenience methods to filter the event lists. One possible use case is to find the highest energy event within a radius of 0.5 deg around the vela position:

[35]:
```python
# select all events within a radius of 0.5 deg around center
from gammapy.utils.regions import SphericalCircleSkyRegion

region = SphericalCircleSkyRegion(center, radius=0.5 * u.deg)
events_gc_3fhl = events_3fhl.select_region(region)

# sort events by energy
events_gc_3fhl.table.sort("ENERGY")

# and show highest energy photon
events_gc_3fhl.energy[-1].to("GeV")
```

[35]:
```
1917.8594 GeV
```

0.2.2 Exercises

- Make a counts energy spectrum for the galactic center region, within a radius of 10 deg.

[ ]:
```

```

Back to Top

## Source catalogs

Gammapy provides a convenient interface to access and work with catalog based data.

In this section we will learn how to:

- Load builtins catalogs from `~gammapy.catalog`
- Sort and index the underlying Astropy tables
- Access data from individual sources

Let's start with importing the 3FHL catalog object from the `~gammapy.catalog` submodule:

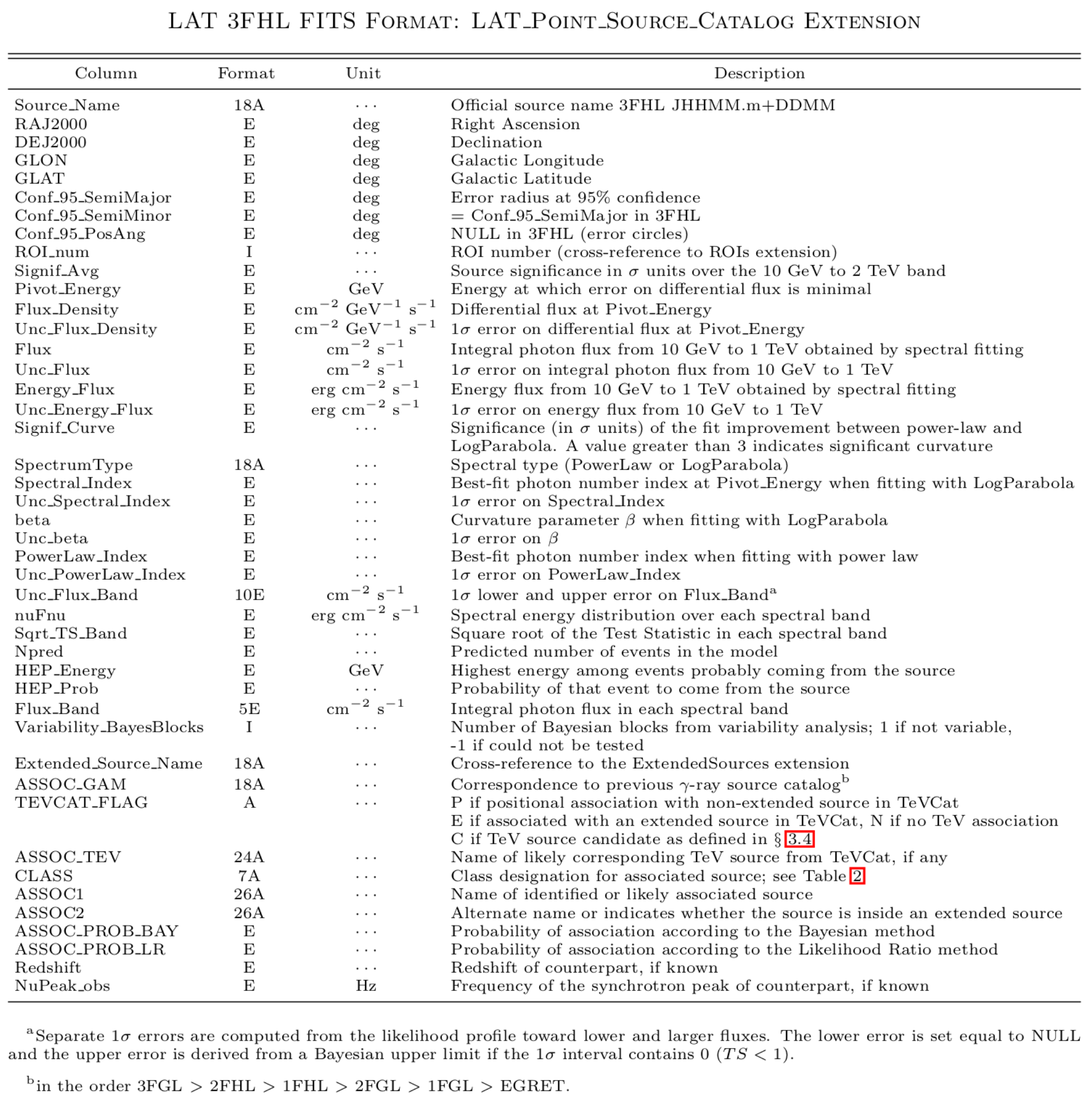
[36]:
```
from gammapy.catalog import SourceCatalog3FHL
```

First we initialize the Fermi-LAT 3FHL catalog and directly take a look at the `.table` attribute:

[37]:
```
fermi_3fhl = SourceCatalog3FHL()
fermi_3fhl.table
```

[37]:
```
<Table length=1556>
   Source_Name      RAJ2000  …     NuPeak_obs
                      deg    …         Hz
     bytes18        float32  …      float32
------------------ -------- … ------------------
3FHL J0001.2-0748    0.3107 …  306196370000000.0
3FHL J0001.9-4155    0.4849 … 6309576500000000.0
               …        … …                  …
3FHL J2359.1-3038  359.7760 …       2.818388e+17
3FHL J2359.3-2049  359.8293 … 4073799600000000.0
```

The following table contains a description of the reconstructed parameters [2]:



LAT 3FHL FITS Format: LAT_Point_Source_Catalog Extension

Column	Format	Unit	Description
Source_Name	18A	⋯	Official source name 3FHL JHHMM.m+DDMM
RAJ2000	E	deg	Right Ascension
DEJ2000	E	deg	Declination
GLON	E	deg	Galactic Longitude
GLAT	E	deg	Galactic Latitude
Conf_95_SemiMajor	E	deg	Error radius at 95% confidence
Conf_95_SemiMinor	E	deg	= Conf_95_SemiMajor in 3FHL
Conf_95_PosAng	E	deg	NULL in 3FHL (error circles)
ROI_num	I	⋯	ROI number (cross-reference to ROIs extension)
Signif_Avg	E	⋯	Source significance in σ units over the 10 GeV to 2 TeV band
Pivot_Energy	E	GeV	Energy at which error on differential flux is minimal
Flux_Density	E	cm^{-2} GeV^{-1} s^{-1}	Differential flux at Pivot_Energy
Unc_Flux_Density	E	cm^{-2} GeV^{-1} s^{-1}	1σ error on differential flux at Pivot_Energy
Flux	E	cm^{-2} s^{-1}	Integral photon flux from 10 GeV to 1 TeV obtained by spectral fitting
Unc_Flux	E	cm^{-2} s^{-1}	1σ error on integral photon flux from 10 GeV to 1 TeV
Energy_Flux	E	erg cm^{-2} s^{-1}	Energy flux from 10 GeV to 1 TeV obtained by spectral fitting
Unc_Energy_Flux	E	erg cm^{-2} s^{-1}	1σ error on energy flux from 10 GeV to 1 TeV
Signif_Curve	E	⋯	Significance (in σ units) of the fit improvement between power-law and LogParabola. A value greater than 3 indicates significant curvature
SpectrumType	18A	⋯	Spectral type (PowerLaw or LogParabola)
Spectral_Index	E	⋯	Best-fit photon number index at Pivot_Energy when fitting with LogParabola
Unc_Spectral_Index	E	⋯	1σ error on Spectral_Index
beta	E	⋯	Curvature parameter β when fitting with LogParabola
Unc_beta	E	⋯	1σ error on β
PowerLaw_Index	E	⋯	Best-fit photon number index when fitting with power law
Unc_PowerLaw_Index	E	⋯	1σ error on PowerLaw_Index
Unc_Flux_Band	10E	cm^{-2} s^{-1}	1σ lower and upper error on Flux_Band[a]
nuFnu	E	erg cm^{-2} s^{-1}	Spectral energy distribution over each spectral band
Sqrt_TS_Band	E	⋯	Square root of the Test Statistic in each spectral band
Npred	E	⋯	Predicted number of events in the model
HEP_Energy	E	GeV	Highest energy among events probably coming from the source
HEP_Prob	E	⋯	Probability of that event to come from the source
Flux_Band	5E	cm^{-2} s^{-1}	Integral photon flux in each spectral band
Variability_BayesBlocks	I	⋯	Number of Bayesian blocks from variability analysis; 1 if not variable, -1 if could not be tested
Extended_Source_Name	18A	⋯	Cross-reference to the ExtendedSources extension
ASSOC_GAM	18A	⋯	Correspondence to previous γ-ray source catalog[b]
TEVCAT_FLAG	A	⋯	P if positional association with non-extended source in TeVCat E if associated with an extended source in TeVCat, N if no TeV association C if TeV source candidate as defined in § 3.4
ASSOC_TEV	24A	⋯	Name of likely corresponding TeV source from TeVCat, if any
CLASS	7A	⋯	Class designation for associated source; see Table 2
ASSOC1	26A	⋯	Name of identified or likely associated source
ASSOC2	26A	⋯	Alternate name or indicates whether the source is inside an extended source
ASSOC_PROB_BAY	E	⋯	Probability of association according to the Bayesian method
ASSOC_PROB_LR	E	⋯	Probability of association according to the Likelihood Ratio method
Redshift	E	⋯	Redshift of counterpart, if known
NuPeak_obs	E	Hz	Frequency of the synchrotron peak of counterpart, if known

[a]Separate 1σ errors are computed from the likelihood profile toward lower and larger fluxes. The lower error is set equal to NULL and the upper error is derived from a Bayesian upper limit if the 1σ interval contains 0 ($TS < 1$).

[b]in the order 3FGL > 2FHL > 1FHL > 2FGL > 1FGL > EGRET.

This looks very familiar again. The data is just stored as an astropy.table.Table object. We have all the methods and attributes of the `Table` object available. E.g. we can sort the underlying table by `Signif_Avg` to find the top 5 most significant sources:

```
[38]: type(fermi_3fhl.table["ASSOC1"])
```

```
[38]: astropy.table.column.Column
```

```
[39]: # sort table by significance
      fermi_3fhl.table.sort("Signif_Avg")

      # invert the order to find the highest values and take the top 5
      top_five_TS_3fhl = fermi_3fhl.table[::-1][:5]

      # print the top five significant sources with association and source class
      top_five_TS_3fhl[["Source_Name", "ASSOC1", "ASSOC2", "CLASS", "Signif_Avg"]]
```

[39]:
```
<Table length=5>
   Source_Name      … Signif_Avg
     bytes18        …  float32
------------------ … ----------
3FHL J0534.5+2201  …    168.641
3FHL J1104.4+3812  …    144.406
3FHL J0835.3-4510  …    138.801
3FHL J0633.9+1746  …     99.734
3FHL J1555.7+1111  …     94.411
```

If you are interested in the data of an individual source you can access the information from catalog using the name of the source or any alias source name that is defined in the catalog:

[40]:
```
print(fermi_3fhl["3FHL J0534.5+2201"])
```

```
*** Basic info ***

Catalog row index (zero-based) : 1555
Source name          : 3FHL J0534.5+2201
Extended name        :
Associations     : Crab Nebula, Crab Nebula, 3FGL J0534.5+2201i
ASSOC_PROB_BAY   : 1.000
ASSOC_PROB_LR    : 1.000
Class            : PWN
TeVCat flag      : P

*** Other info ***

Significance (10 GeV - 2 TeV)    : 168.641
Npred                            : 2602.9

HEP Energy       : 1463.300 GeV
HEP Probability  : 1.000
Bayesian Blocks  : 1
Redshift         : --
NuPeak_obs       : 0.0 Hz

*** Position info ***

RA                   : 83.635 deg
DEC                  : 22.019 deg
GLON                 : 184.554 deg
GLAT                 : -5.780 deg

Semimajor (95%)      : 0.0080 deg
Semiminor (95%)      : 0.0080 deg
Position angle (95%) : 0.00 deg
```

```
ROI number           : 430

*** Spectral fit info ***

Spectrum type                    : PowerLaw
Significance curvature           : 1.4
Power-law spectral index         : 2.220 +- 0.025
Pivot energy                     : 22.7 GeV
Flux Density at pivot energy     : 1.71e-10 +- 3.39e-12 cm-2 GeV-1 s-1
Integral flux (10 GeV - 1 TeV)   : 8.66e-09 +- 1.71e-10 cm-2 s-1
Energy flux (10 GeV - TeV)       : 4.91e-10 +- 1.66e-11 erg cm-2 s-1

*** Spectral points ***

 e_min   e_max       flux     flux_errn   flux_errp      e2dnde     e2dnde_errn
e2dnde_errp  is_ul   flux_ul     e2dnde_ul    sqrt_ts
  GeV     GeV    1 / (cm2 s) 1 / (cm2 s) 1 / (cm2 s) erg / (cm2 s) erg / (cm2 s)
erg / (cm2 s)       1 / (cm2 s) erg / (cm2 s)
-------- --------- ----------- ----------- ----------- ------------- -------------
------------- ----- ----------- ------------- -------
 10.000  20.000  5.170e-09   1.334e-10   1.334e-10     1.642e-10     4.237e-12
4.237e-12 False         nan           nan 125.157
 20.000  50.000  2.245e-09   8.672e-11   8.672e-11     1.181e-10     4.561e-12
4.561e-12 False         nan           nan  88.715
 50.000  150.000  9.243e-10   5.497e-11   5.497e-11     1.087e-10     6.464e-12
6.464e-12 False         nan           nan  59.087
150.000  500.000  2.759e-10   2.916e-11   3.136e-11     9.230e-11     9.757e-12
1.049e-11 False         nan           nan  33.076
500.000 2000.000  6.684e-11   1.463e-11   1.692e-11     6.901e-11     1.510e-11
1.747e-11 False         nan           nan  15.573

/home/bornagain/miniconda3/envs/gammapy-0.20.1/lib/python3.8/site-
packages/gammapy/catalog/fermi.py:1092: FutureWarning: Format strings passed to
MaskedConstant are ignored, but in future may error or produce different
behavior
  ss += "{:<16s} : {:.3f}\n".format("Redshift", d["Redshift"])
```

[41]:
```python
mkn_421_3fhl = fermi_3fhl["3FHL J1104.4+3812"]

# or use any alias source name that is defined in the catalog
mkn_421_3fhl = fermi_3fhl["Mkn 421"]
print(mkn_421_3fhl.data["Signif_Avg"])
```

```
144.40611
```

0.2.3 Exercises

- Try to load the Fermi-LAT 2FHL catalog and check the total number of sources it contains.

- Select all the sources from the 2FHL catalog which are contained in the Galactic Center region. The methods `~gammapy.maps.WcsGeom.contains()` and `~gammapy.catalog.SourceCatalog.positions` might be helpful for this. Add markers for all these sources and try to add labels with the source names.
- Try to find the source class of the object at position ra=68.6803, dec=9.3331

Back to Top

## Spectral models and flux points

In the previous section we learned how access basic data from individual sources in the catalog. Now we will go one step further and explore the full spectral information of sources. We will learn how to:

- Plot spectral models
- Compute integral and energy fluxes
- Read and plot flux points

As a first example we will start with the Crab Nebula:

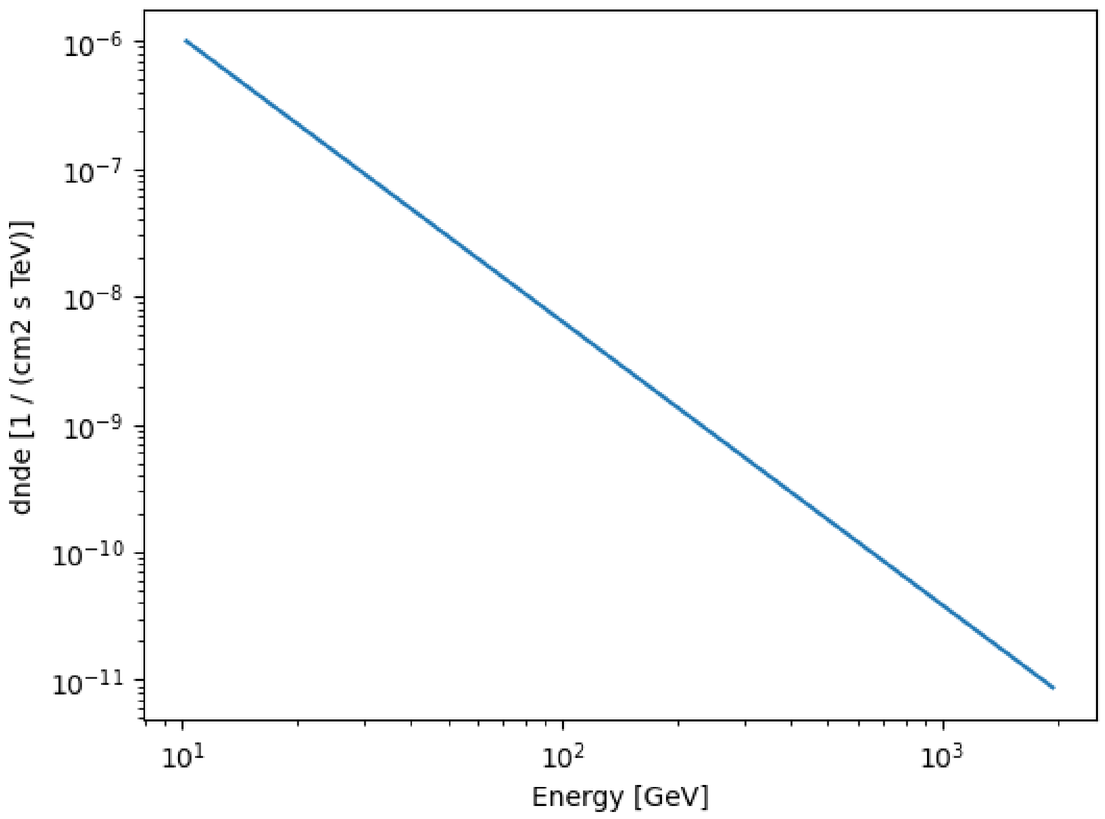
[42]:
```
crab_3fhl = fermi_3fhl["Crab Nebula"]
crab_3fhl_spec = crab_3fhl.spectral_model()
print(crab_3fhl_spec)
```

```
PowerLawSpectralModel

  type      name     value    … is_norm link
-------- --------- ---------- … ------- ----
spectral     index 2.2202e+00 …   False
spectral amplitude 1.7132e-10 …    True
spectral reference 2.2726e+01 …   False
```

The `crab_3fhl_spec` is an instance of the `~gammapy.modeling.models.PowerLaw2SpectralModel` model, with the parameter values and errors taken from the 3FHL catalog.

Let's plot the spectral model in the energy range between 10 GeV and 2000 GeV:

[43]:
```
ax_crab_3fhl = crab_3fhl_spec.plot(
    energy_bounds=[10, 2000] * u.GeV, energy_power=0
)
```



We assign the return axes object to variable called `ax_crab_3fhl`, because we will re-use it later to plot the flux points on top.

To compute the differential flux at 100 GeV we can simply call the model like normal Python function and convert to the desired units:

[44]:
```
crab_3fhl_spec(100 * u.GeV).to("cm-2 s-1 GeV-1")
```

[44]: $6.3848913 \times 10^{-12}\ \frac{1}{\mathrm{GeV\,s\,cm^{2}}}$

Next we can compute the integral flux of the Crab between 10 GeV and 2000 GeV:

[45]:
```
crab_3fhl_spec.integral(energy_min=10 * u.GeV, energy_max=2000 * u.GeV).to(
    "cm-2 s-1"
)
```

[45]: $8.6745734 \times 10^{-9}\ \frac{1}{\mathrm{s\,cm^{2}}}$

We can easily convince ourself, that it corresponds to the value given in the Fermi-LAT 3FHL catalog:

[46]:
```
crab_3fhl.data["Flux"]
```

[46]: $8.6589091 \times 10^{-9}\ \frac{1}{\mathrm{s\,cm^{2}}}$

In addition we can compute the energy flux between 10 GeV and 2000 GeV:

```
[47]: crab_3fhl_spec.energy_flux(energy_min=10 * u.GeV, energy_max=2000 * u.GeV).to(
          "erg cm-2 s-1"
      )
```

[47]: $5.3114892 \times 10^{-10}\ \frac{\mathrm{erg}}{\mathrm{s\,cm^{2}}}$

Next we will access the flux points data of the Crab:

```
[48]: print(crab_3fhl.flux_points)
```

```
FluxPoints
----------

  geom                       : RegionGeom
  axes                       : ['lon', 'lat', 'energy']
  shape                      : (1, 1, 5)
  quantities                 : ['norm', 'norm_errp', 'norm_errn', 'norm_ul',
'sqrt_ts', 'is_ul']
  ref. model                 : pl
  n_sigma                    : 1
  n_sigma_ul                 : 2
  sqrt_ts_threshold_ul       : 1
  sed type init              : flux
```

If you want to learn more about the different flux point formats you can read the specification here.

No we can check again the underlying astropy data structure by accessing the `.table` attribute:

```
[49]: crab_3fhl.flux_points.to_table(sed_type="dnde", formatted=True)
```

```
[49]: <Table length=5>
       e_ref    e_min   e_max   … sqrt_ts is_ul
        GeV      GeV     GeV    …
      float64  float64 float64  … float32  bool
      -------- ------- -------- … ------- -----
        14.142  10.000   20.000 … 125.157 False
        31.623  20.000   50.000 …  88.715 False
        86.603  50.000  150.000 …  59.087 False
       273.861 150.000  500.000 …  33.076 False
      1000.000 500.000 2000.000 …  15.573 False
```

Finally let's combine spectral model and flux points in a single plot and scale with `energy_power=2` to obtain the spectral energy distribution:

```
[ ]: ax = crab_3fhl_spec.plot(energy_bounds=[10, 2000] * u.GeV, energy_power=2)
     ax = crab_3fhl_spec.plot_error(
         energy_bounds=[10, 2000] * u.GeV,
         energy_power=2,
         facecolor="tab:blue"
     )
     crab_3fhl.flux_points.plot(ax=ax, sed_type="dnde", energy_power=2);
```

0.2.4 Exercises

- Plot the spectral model and flux points for PKS 2155-304 for the 3FGL and 2FHL catalogs. Try to plot the error of the model (aka "Butterfly") as well. Note this requires the uncertainties package to be installed on your machine.

[ ]:
```

```

0.3 What next?

This was a quick introduction to some of the high level classes in Astropy and Gammapy.

- To learn more about those classes, go to the API docs (links are in the introduction at the top).
- To learn more about other parts of Gammapy (e.g. Fermi-LAT and TeV data analysis), check out the other tutorial notebooks.
- To see what's available in Gammapy, browse the Gammapy docs or use the full-text search.
- If you have any questions, ask on the mailing list.

Back to Top

0.4 References

1. Cicerone: Data — LAT Data Files - Column Descriptions. Retrieved [November 5, 2022] from https://fermi.gsfc.nasa.gov/ssc/data/analysis/documentation/Cicerone/Cicerone_Data/LAT_Data_Columns.ht

2. M. Ajello et al. [Fermi-LAT Collaboration], TFHL: The third catalog of hard Fermi-LAT sources, Astrophys. J. Suppl. 232, no. 2, 18 (2017) doi:10.3847/1538-4365/aa8221 [arXiv:1702.00664v3 [astro-ph.HE]].

[ ]:
```

```

Back to Top

The description of the reconstructed parameters in the event list[1]:

0.5 Table 1

Index	Event Parameter (units)	Description
0	ENERGY (MeV)	Reconstructed energy of the event
1	RA (degrees)	Reconstructed direction of the event in Right Ascension
2	DEC (degrees)	Reconstructed direction of the event in Declination

Index	Event Parameter (units)	Description
3	L (degrees)	Reconstructed direction of the event in Galactic Longitude
4	B (degrees)	Reconstructed direction of the event in Galactic Latitude
5	THETA (degrees)	Reconstructed angle of incidence of the event with respect to the LAT boresight (+Z axis of the spacecraft - the line normal to the top surface of the LAT)
6	PHI (degrees)	Reconstructed angle of incidence of the event with respect to the +X axis (the line normal to the sun-facing side of the spacecraft)
7	ZENITH_ANGLE (degrees)	Angle between the reconstructed event direction and the zenith line (originates at the center of the Earth and passes through the center of mass of the spacecraft)
8	EARTH_AZIMUTH_ANGLE (degrees)	Angle of the reconstructed event direction with respect to North (line from spacecraft origin to north celestial pole) as projected onto a plane normal to the zenith. The angle is measured in degrees east of north, such that 90 degrees indicates that the event originated from the west
9	TIME (seconds)	Mission elapsed time when the event was detected (MET is the total number of seconds since 00:00:00 on January 1, 2001 UTC)
10	EVENT_ID	Sequence number for the event in the LAT data acquisition period
11	RUN_ID	Unique identifier for each LAT data acquisition period
12	RECON_VERSION	Version of event reconstruction software in use at the time the event was detected

Index	Event Parameter (units)	Description
13	CALIB_VERSION (3-element array)	Version of the calibration tables for the ACD, CAL, and TKR (in that order) in use at the time the event was detected. (This column is currently unused)
14	EVENT_CLASS	A bitfield indicating which event class selections a given event has passed. In Pass 8 the internal FITS format of this column has been changed from a 32-bit integer (TFORMn=J) to a 32-bit bit column (TFORMn=32X) and supports bitwise selections with the fselect FTOOL. Pass 8 populates a much larger number of bit values than Pass 7. Bits for the recommended event classes are bit 4 (P8R2_TRANSIENT020), bit 7 (P8R2_SOURCE), and bit 10 (P8R2_ULTRACLEANVETO)
15	EVENT_TYPE	A bitfield indicating which event type selections a given event has passed. This column is a 32-bit bit column (TFORMn=32X) and supports bitwise selections with fselect.
16	CONVERSION_TYPE	Indicates whether the event induced pair production in the front (thin) layers or the back (thick) layers of the tracker (front=0, back=1)

Index	Event Parameter (units)	Description
17	LIVETIME (seconds)	A short-term measure of accumulated livetime of the LAT. This value can have gaps and it resets every few seconds. For large time intervals, the LIVETIME documented in the spacecraft file is correct. However, for short time intervals, this LIVETIME value can be compared between two events to gauge the fraction of dead time
18	DIFRSP0	Diffuse response for an additional component (currently unused)
19	DIFRSP1	Diffuse response for an additional component (currently unused)
20	DIFRSP2	Diffuse response for an additional component (currently unused)
21	DIFRSP3	Diffuse response for an additional component (currently unused)
22	DIFRSP4	Diffuse response for an additional component (currently unused)

```
[ ]:
```