Democratising Archival X-ray Astronomy (DAXA)

David J Turner, Jessica E Pilling, Agrim Gupta

Apr 26, 2024

CONTENTS:

CHAPTER

ONE

INTRODUCTION TO DAXA

DAXA is a Python module designed to make the acquisition and processing of archives of X-ray astronomy data as painless as possible. It provides a consistent interface to the downloading and cleaning processes of each telescope, allowing the user to easily create multi-mission X-ray archives, allowing for the community to make better use of archival X-ray data. This process can be as simple or as in-depth as the user requires; if the default settings are used then data can be acquired and processed into an archive in only a few lines of code.

As the missions (i.e. telescopes) that should be included in the archive are defined, the user can filter the desired observations based on a unique identifier (i.e. observation ID), on whether observations are near to a coordinate (or set of coordinates), and the time frame in which the observations were taken. As such it is possible to very quickly identify what archival data might be available for a set of objects you wish to study. It is also possible to place no filters on the desired observations, and as such process every observation available for a set of missions.

1.1 Which missions are supported?

DAXA is still in a relatively early stage of development, and as such the support for local re-processing is limited; however, support for the acquisition and use of pre-processed data is implemented for a wide selection of telescopes:

- XMM-Newton Pointed
- eROSITA Commissioning
- eROSITA All-Sky Survey DR1 (German Half)
- [Under Development - data acquisition implemented] NuSTAR Pointed
- [Under Development - data acquisition implemented] Chandra
- [Under Development - RASS/pointed data acquisition implemented] ROSAT
- [Under Development - XRT/BAT/UVOT data acquisition implemented] Swift
- [Under Development - data acquisition implemented] Suzaku
- [Under Development - data acquisition implemented] ASCA
- [Under Development - data acquisition implemented] INTEGRAL

If you wish to help with implementation of Chandra, NuSTAR, or some other mission, please get in contact!

1.2 Analysing the processed archives

Once an archive of cleaned X-ray data has been created, it can be analysed in all the standard ways, however you may also wish to consider [X-ray: Generate and Analyse (XGA)](https://github.com/DavidT3/XGA), a companion module to DAXA.

XGA is also completely open source, and is a generalised tool for the analysis of X-ray emission from astrophysical sources. The software operates on a 'source based' paradigm, where the user declares sources or samples of objects which are analogous to astrophysical sources in the sky, with XGA determining which data (if any) are relevant to a particular source, and providing a powerful (but easy to use) interface for the generation and analysis of data products. The module is fully documented, with tutorials and API documentation available.

CHAPTER

TWO

INSTALLING AND CONFIGURING DAXA

This is a slightly more complex installation than many Python modules, but shouldn't be too difficult. If you're having issues feel free to contact me.

2.1 The Module

DAXA is not yet available on PyPi or Conda, though it will be in the future

I strongly recommend that you make use of Python virtual environments, or (even better) Conda/Mamba virtual environments when installing DAXA.

You can fetch the current working version from the git repository, and install it (this method has replaced 'python setup.py install'):

```
git clone https://github.com/DavidT3/DAXA
cd DAXA
python -m pip install .
```

Alternatively you could use the 'editable' option (this has replaced running setup.py and passing 'develop') so that any changes you pull from the remote repository are reflected without having to reinstall DAXA.

```
git clone https://github.com/DavidT3/DAXA
cd DAXA
python -m pip install --editable .
```

Once installed, you can import DAXA in the usual way (the command will be lowercase, 'import daxa'). If you have stayed in the DAXA directory cloned from GitHub and opened Python there then it is possible to 'import DAXA', but that will behave very strangely as it hasn't actually imported the module, but the directory.

2.2 Required Dependencies for Processing Data

- **XMM**
  - Science Analysis System (SAS) - v14 or above
  - HEASoft (lcurve is required for XMM processing) - tested on v6.29 and v6.31
- **eROSITA**
  - eROSITA Science Analysis Software System (eSASS)
  - HEASoft - tested on v6.29 and v6.31

All required Python modules can be found in requirements.txt, and should be added to your system during the installation of DAXA.

Excellent installation guides for SAS and HEASoft exist.

A docker container for DAXA (and the related module XGA) is in development, where all dependencies will already be installed.

2.3 Configuring DAXA

The first run of DAXA will create a configuration file, which by default will be ~/.config/daxa/daxa.cfg. This does not **need** to be configured by the user, but you can use it to set the default directory for saving DAXA outputs, as well as an override of the number of cores that DAXA is allowed to use.

CHAPTER

THREE

TUTORIALS

3.1 Identifying and acquiring relevant X-ray observations

This tutorial will explain the basic concepts and components of the X-ray astronomy Python module 'Democratising Archival X-ray Astronomy' (DAXA). We will particularly focus on the various 'mission' classes (implemented for each of the X-ray telescopes that DAXA supports), and the functionality that allows for large numbers of observations to be selected and downloaded.

DAXA mission classes allow the user to interact with and search various X-ray telescope archives, all through an identical interface, within a single module, and through Python commands. It should be simple for anyone with a passing familiarity with Python to identify and acquire X-ray data relevant to their research.

3.1.1 Import Statements

[1]:
```
from daxa.mission import MISS_INDEX, XMMPointed, Chandra, eRASS1DE, eROSITACalPV,␣
↪NuSTARPointed, Swift, \
    ROSATAllSky, ROSATPointed, ASCA, INTEGRALPointed, Suzaku

from datetime import date
import numpy as np
from astropy.units import Quantity
```

3.1.2 What missions are available?

We have implemented support for access and searching the archives of many X-ray telescopes; we would also be willing to provide an interface to the data archives of other X-ray telescopes, if feasible - please feel free to reach out using the support page if you think there is one we should add.

Some telescopes (such as the ROSATPointed/ROSATAllSky and eROSITACalPV/eRASS1DE classes) are not uniquely represented by a single DAXA mission - this is generally the case when the telescope in question has been used in distinctly different ways (e.g. ROSAT had a survey phase and a pointed phase), such that the data for one mode may not be relevant to all applications.

[2]:
```
MISS_INDEX
```

[2]:
```
{'xmm_pointed': daxa.mission.xmm.XMMPointed,
 'nustar_pointed': daxa.mission.nustar.NuSTARPointed,
 'erosita_calpv': daxa.mission.erosita.eROSITACalPV,
 'erosita_all_sky_de_dr1': daxa.mission.erosita.eRASS1DE,
```

(continues on next page)

(continued from previous page)

```
 'chandra': daxa.mission.chandra.Chandra,
 'rosat_all_sky': daxa.mission.rosat.ROSATAllSky,
 'rosat_pointed': daxa.mission.rosat.ROSATPointed,
 'swift': daxa.mission.swift.Swift,
 'suzaku': daxa.mission.suzaku.Suzaku,
 'asca': daxa.mission.asca.ASCA,
 'integral_pointed': daxa.mission.integral.INTEGRALPointed}
```

XMM-Pointed

This class is for acquiring XMM-Newton data, particularly that taken when the telescope is pointing at specific targets; i.e. it cannot be used to filter and download data taken when the telescope is slewing from one target to the next. XMM is still in use, so the number of observations that are available are constantly increasing - this also means that some data are still in a proprietary period, and DAXA will not be able to access them. **Note that only raw, unprocessed, data can currently be downloaded for XMM using DAXA.**

The three EPIC instruments are supported (PN, MOS1, and MOS2), as well as the two grating spectrometers (RGS1 and RGS2). We also support the acquisition of XMM optical monitor (OM) data, but support for processing it is more limited.

Values that can be passed to the `insts` argument of XMMPointed on declaration (either as single strings or as part of a list):

- PN
- MOS1
- MOS2
- RGS1
- RGS2
- OM

[3]:
```
xm = XMMPointed()
```

```
/Users/dt237/code/DAXA/daxa/mission/xmm.py:83: UserWarning: 140 of the 17701␣
↪observations located for this mission have been removed due to NaN RA or Dec values
  self._fetch_obs_info()
```

Chandra

The class for acquiring Chandra data, specifically taken when pointing at a target - Chandra does not have easily accessible 'slew' data, and this class does not consider it at all.

Data from all instruments can be downloaded by DAXA - though unlike XMM there are not data being taken simultaneously, so each ObsID is generally only associated with one instrument. Note though that the gratings (HETG and LETG) are used *with* another instrument, so cannot be passed by themselves.

Note that the standard format of Chandra data can be downloaded using DAXA, allowing for the use of the standard Chandra re-processing scripts. This standard download includes pre-made images.

Values that can be passed to the `insts` argument of Chandra (either as single string or as part of a list):

- ACIS-I

- ACIS-S
- HRC-I
- HRC-S
- LETG [an instrument that this grating was used with must be specified]
- HETG [an instrument that this grating was used with must be specified]

[4]:
```
ch = Chandra()
```

eROSITA All-Sky DR1 (German Half)

The class for acquiring data from the first data release by the German part of the eROSITA consortium - this covers half the sky to 1/8th the planned final survey depth of eROSITA. All of this data is taken in survey mode, where the telescope is constantly slewing. **Note that the data downloads can include pre-generated images and exposure maps.**

The eROSITA telescope is made up of 7 telescope modules that observe simultaneously, and which can be individually selected on declaration with the `insts` argument - **it is not recommended to use DAXA to limit the telescope modules being considered, as we crudely modify the event lists to remove events from non-selected telescope modules**.

Values that can be passed to the `insts` argument of eRASS1DE (either as a single string or as part of a list):

- TM1
- TM2
- TM3
- TM4
- TM5
- TM6
- TM7

[5]:
```
ea = eRASS1DE()
```

eROSITA Calibration and Performance Verification

This class provides DAXA access to the data from the calibration and performance verification stages of the eROSITA mission, including all pointed observations and survey regions (such as the eROSITA Final Equatorial-Depth Survey; eFEDS). **Note that only event list data can be downloaded for this class.**

The eROSITA telescope is made up of 7 telescope modules that observe simultaneously, and which can be individually selected on declaration with the `insts` argument - **it is not recommended to use DAXA to limit the telescope modules being considered, as we crudely modify the event lists to remove events from non-selected telescope modules**.

Values that can be passed to the `insts` argument of eROSITACalPV (either as a single string or as part of a list):

- TM1
- TM2
- TM3
- TM4
- TM5

- TM6
- TM7

[6]:
```
ecpv = eROSITACalPV()
```

NuSTAR-Pointed

This class is for acquiring NuSTAR data, particularly that taken when the telescope is pointing at specific targets; i.e. it cannot be used to filter and download data taken when the telescope is slewing from one target to the next. NuSTAR is still in use, so the number of observations that are available are constantly increasing - this also means that some data are still in a proprietary period, and DAXA will not be able to access them. **Note that data downloads can optionally include pre-generated images**

NuSTAR has two instruments that observe simultaneously, and are essentially identical, Focal Plane Module A & B.

Values that can be passed to the `insts` argument of NuSTARPointed on declaration (either as single strings or as part of a list):

- FPMA
- FPMB

[7]:
```
nu = NuSTARPointed()
```

Swift

This class is for acquiring Swift data. Swift is still in use, so the number of observations that are available are constantly increasing. Also, due to the Swift's primary mission of rapid transient follow-up, and how observations are split up, the table of available observations is unusually large (only INTEGRAL is comparable) - **as such this class may take several minutes to declare, depending on your internet connection**.

Swift has three instruments that generally observe simultaneously; the X-ray Telescope (XRT), the Burst Alert Telescope (BAT) which is has poor spatial resolution but has a large field of view and is sensitive to very high energy photons, and the Ultraviolet and Optical Telescope (UVOT). **Note that data downloads can optionally include pre-generated images, but not for BAT observations.**

Values that can be passed to the `insts` argument of Swift on declaration (either as single strings or as part of a list); the XRT and BAT instruments are selected by default:

- XRT
- BAT
- UVOT

[8]:
```
sw = Swift()
```

```
/Users/dt237/code/DAXA/daxa/mission/swift.py:122: UserWarning: 599 of the 354020␣
↪observations located for Swift have been removed due to all instrument exposures being␣
↪zero.
  self._fetch_obs_info()
/Users/dt237/code/DAXA/daxa/mission/swift.py:122: UserWarning: 17 of the 354020␣
↪observations located for Swift have been removed due to all chosen instrument (XRT,␣
↪BAT) exposures being zero.
  self._fetch_obs_info()
```

ROSAT All-Sky Survey

This class provides access to data taken by ROSAT during its all-sky survey. Though ROSAT had multiple instruments, this was all taken with the position-sensitive proportional counters (PSPC) - specifically with the 'PSPC-C' instrument (ROSAT had two PSPC instruments). The initial all-sky survey was abandoned after an accidental pass over the sun destroyed PSPC-C, but follow-up observations with PSPC-B were taken towards the end of ROSAT's lifetime to complete the survey (**these are not included in this class**).

Note: Data acquired through this class will include just event lists by default, but can also include pre-generated images and exposure maps.

[9]:
```
ra = ROSATAllSky()
```

ROSAT-Pointed

This class provides access to data taken by ROSAT during the pointed phase of its lifetime (including follow-up observations used to complete the all-sky survey). ROSAT instruments could not observe simultaneously, so each separate observation uses a single instrument. **Note: Data acquired through this class will include just event lists by default, but can also include pre-generated images (PSPC & HRI) and exposure maps (just PSPC).**

Values that can be passed to the `insts` argument of ROSATPointed on declaration (either as single strings or as part of a list):

- PSPCB
- PSPCC
- HRI

[10]:
```
rp = ROSATPointed()
```

Suzaku

This class provides access to data taken by the Suzaku X-ray telescope during pointed observations (data taken while slewing are not included in the public archive). **Note: Data acquired through this class will include just event lists by default, but can also include pre-generated images.**

We provide access to XIS data, but not XRS (as the cooling system was damaged soon after launch) or HXD (as it was not an imaging instrument).

Values that can be passed to the `insts` argument of Suzaku on declaration (either as single strings or as part of a list):

- XIS0
- XIS1
- XIS2
- XIS3

[11]:
```
su = Suzaku()
```

```
/Users/dt237/code/DAXA/daxa/mission/suzaku.py:109: UserWarning: 14 of the 3055␣
↪observations located for Suzaku have been removed due to all instrument exposures␣
↪being zero.
  self._fetch_obs_info()
```

ASCA

This class provides access to data taken by the ASCA X-ray telescope during pointed observations (we cannot find anywhere to access the data taken whilst slewing). **Note: Data acquired through this class will include just event lists by default, but can also include pre-generated images, spectra, and lightcurves.**

Values that can be passed to the `insts` argument of ASCA on declaration (either as single strings or as part of a list):

- SIS0
- SIS1
- GIS2
- GIS3

[12]:
```
asca = ASCA()
```

```
/Users/dt237/code/DAXA/daxa/mission/asca.py:125: UserWarning: 5 of the 3079 observations␣
↪located for ASCA have been removed due to all instrument exposures being zero.
  self._fetch_obs_info()
```

INTEGRAL-Pointed

This class is for acquiring INTEGRAL data. INTEGRAL is still in use, so the number of observations that are available are still increasing (*though operations will see around the end of 2024*). The table of available observations is unusually large (only Swift is comparable) - **as such this class may take several minutes to declare, depending on your internet connection**.

INTEGRAl has a selection of instruments that cover different parts of the X-ray and Gamma-ray energy range - most of them are based on the 'coded mask' technology, and so have very limited spatial resolution. **Note that only raw data/calibration files can be downloaded through DAXA, there are no pre-processed images available**.

Values that can be passed to the `insts` argument of INTEGRALPointed on declaration (either as single strings or as part of a list):

- JEMX1
- JEMX2
- ISGRI
- PICsIT
- SPI

[13]:
```
inte = INTEGRALPointed()
```

```
/Users/dt237/code/DAXA/daxa/mission/integral.py:110: UserWarning: 241 of the 205930␣
↪observations located for INTEGRAL have been removed due to all instrument exposures␣
↪being zero.
  self._fetch_obs_info()
/Users/dt237/code/DAXA/daxa/mission/integral.py:110: UserWarning: 7403 of the 205930␣
↪observations located for INTEGRAL have been removed due to all chosen instrument␣
↪(JEMX1, JEMX2, ISGRI) exposures being zero.
  self._fetch_obs_info()
```

3.1.3 Mission properties

Here we run through the basic properties that each of the DAXA mission classes share. We also show examples, particularly in cases where differences between telescopes result in us assigning different values for particular properties.

Name

The name assigned to each mission class, so that they can be differentiated both by the user and by DAXA functions. Each mission class has two names, the 'internal DAXA name' (used by DAXA to identify missions) and the 'pretty name', which is typically in a more aethsetically pleasing format.

For instance, we show the 'name' and 'pretty name' of the XMMPointed and eROSITA All-Sky Survey 1 classes:

[14]:
```
print(xm.name)
print(xm.pretty_name, '\n')

print(ea.name)
print(ea.pretty_name)
```

```
xmm_pointed
XMM-Newton Pointed

erosita_all_sky_de_dr1
eRASS DE:1
```

All Instruments & Chosen Instruments

Most telescopes have multiple instruments, though not all are necessarily selected by default. This can be for a number of reasons, but is generally because they either aren't suited to archival/serindipitious science (which is the primary reason this module exists) or because they aren't X-ray telescopes (like the optical monitors on XMM and Swift).

The instruments whose data is to be acquired are generally specified when the mission class is declared (using the `insts` argument), but can also be set through the `chosen_instruments` property.

Every available instrument for a mission is stored in the `all_mission_instruments` property:

[15]:
```
print(rp.all_mission_instruments)
print(ch.all_mission_instruments)
```

```
['PSPCB', 'PSPCC', 'HRI']
['ACIS-I', 'ACIS-S', 'HRC-I', 'HRC-S', 'HETG', 'LETG']
```

The selected instruments (normally specified on declaration) are stored in the `chosen_instruments` property, which can also be set:

[16]:
```
print(rp.chosen_instruments)
print(ch.chosen_instruments)
ch.chosen_instruments = ['ACIS-I', 'ACIS-S']
print(ch.chosen_instruments)
```

```
['PSPCB', 'PSPCC', 'HRI']
['ACIS-I', 'ACIS-S', 'HRC-I', 'HRC-S']
['ACIS-I', 'ACIS-S']
```

ObsID Regular Expression

Each of the mission's observations are uniquely identified by an 'ObsID', and each telescope/mission has a different format of ObsID (generally just made up of numeric characters) - there are points where the mission class may have to check the format of a supplied ObsID, and it does that by comparing to the ObsID regular expression:

[17]:
```
print(xm.id_regex)
print(rp.id_regex)
```

```
^[0-9]{10}$
^(RH|rh|RP|rp|RF|rf|WH|wh|WP|wp|WF|wf)\d{6}([A-Z]\d{2}|)$
```

Field of View

The field of view (FoV) values attached to DAXA mission classes represent the half-width (or radius) of the region of sky that a telescope/instrument observes. Given that each telescope instrument tends to have a unique geometry, this is a simplification, but it is beneficial to have a single number that represents how much of the sky an instrument can see.

In the simplest cases, the FoV property is just a single quantity, meaning that there is either only one instrument, or that every instrument has the same field of view. In other cases there may be multiple instruments associated with a mission, in which case they will all have their own entry in a FoV dictionary.

Finally, some telescopes (such as Chandra) have instruments which have irregular geometries (the ACIS-S and HRC-S chips), or that are frequently used in different observational modes that enable and disable different parts of the sensor. As such the actual coverage of the FoV can vary dramatically, in cases like these we will have gone with the half-width of the longest possible side of the field of view.

[18]:
```
print(xm.pretty_name, '-', xm.fov, '\n')
print(rp.pretty_name, '-', rp.fov, '\n')
print(sw.pretty_name, '-', sw.fov, '\n')
print(ch.pretty_name, '-', ch.fov)
```

```
XMM-Newton Pointed - 15.0 arcmin

ROSAT Pointed - {'PSPCB': <Quantity 60. arcmin>, 'PSPCC': <Quantity 60. arcmin>, 'HRI':
↪<Quantity 19. arcmin>}

Swift - {'XRT': <Quantity 11.8 arcmin>, 'BAT': <Quantity 50. arcmin>, 'UVOT': <Quantity␣
↪8.5 arcmin>}

Chandra - {'ACIS-I': <Quantity 27.8 arcmin>, 'ACIS-S': <Quantity 27.8 arcmin>, 'HRC-I':
↪<Quantity 49.5 arcmin>, 'HRC-S': <Quantity 49.5 arcmin>}
```

```
/var/folders/td/gw9qkx6d3szb1nkt_cfvcbzm000vzl/T/ipykernel_23287/579883568.py:4:␣
↪UserWarning: Chandra FoV are difficult to define, as they can be strongly dependant on␣
↪observation mode; as such take these as very approximate.
  print(ch.pretty_name, '-', ch.fov)
```

Coordinate Frame

This property contains the coordinate frame for the central positions of observations taken by the mission - this is largely irrelevant to the user, and will be used in cases where the mission class needs to compare an input coordinate to an observation coordinate.

Also, practically speaking the difference between the ICRS and FK5 frames (most commonly used) is neglible compared to the typical spatial uncertainty involved in X-ray data:

[19]:
```
print(ch.coord_frame)
print(asca.coord_frame)
```

```
<class 'astropy.coordinates.builtin_frames.icrs.ICRS'>
<class 'astropy.coordinates.builtin_frames.fk5.FK5'>
```

Pre-processed Energy Bands

This property will contain the upper and lower energy bounds available for pre-processed data products for a particular mission (if their online dataset supplies energy-bound data products) - these energy bounds are provided on an instrument level (as some missions provide different energy-bound products for different instruments). The left hand column indicates the lower energy bound, and the right hand column the upper energy bound.

An energy bound being present here does not guarantee that all products supplied by the mission online dataset are available in that bound - e.g. some missions provide bound images and a single, general, exposure map.

[20]:
```
rp.preprocessed_energy_bands
```

[20]:
```
{'PSPCB': <Quantity [[0.07, 2.4 ],
            [0.07, 0.4 ],
            [0.4 , 2.4 ]] keV>,
 'PSPCC': <Quantity [[0.07, 2.4 ],
            [0.07, 0.4 ],
            [0.4 , 2.4 ]] keV>,
 'HRI': <Quantity [[0.07, 2.4 ]] keV>}
```

If the mission in question cannot provide pre-processed data products that are energy bound, then an error will be raised:

[21]:
```
inte.preprocessed_energy_bands
```

```
---------------------------------------------------------------------------
PreProcessedNotSupportedError             Traceback (most recent call last)
Cell In [21], line 1
----> 1 inte.preprocessed_energy_bands

File ~/code/DAXA/daxa/mission/base.py:730, in BaseMission.preprocessed_energy_bands(self)
    727 # If this attribute is not set then we're going to assume that the archive doesn
↪'t provide any products
    728 #  which are energy bound
    729 if self._template_en_trans is None:
--> 730     raise PreProcessedNotSupportedError("This mission's archive does not supply␣
↪pre-processed products within "
    731                                         "specific energy bands.")
    733 # The attribute is organized as a nested dictionary - with two possible␣
↪configurations, one with instrument
```

(continues on next page)

(continued from previous page)

```
    734 #  names as top level keys, then lower level keys being lower energy bounds, and␣
↪the
    735 #  lowest level keys being upper energy bounds - the other configuration is the␣
↪same, but doesn't have top
    736 #  level instrument keys (these then apply to all instruments of a mission).
    737 if isinstance(list(self._template_en_trans.keys())[0], Quantity):

PreProcessedNotSupportedError: This mission's archive does not supply pre-processed␣
↪products within specific energy bands.
```

Observation Information

One of the most important properties of a DAXA mission class - this returns a dataframe of all the observations that are associated with a mission. This can include observations that are not yet publicly available (i.e. they are still in a proprietary period), but will never include observations that are planned but haven't been taken yet.

In most cases this data is dynamically updated when a mission is declared (i.e. the table is pulled down from a mission server) - this is not the case for eROSITACalPV and eRASS1DE. **Please also note that the Swift and INTEGRAL-Pointed missions have very large ``all_obs_info`` tables due to the way their data/observations are organised.**

Some information will be constant across telescopes, and some will be mission specific. We present truncated versions of `all_obs_info` for every DAXA mission as of the current date:

[22]:
```
date.today().strftime('%d-%B-%Y')
```

[22]:
```
'24-April-2024'
```

[23]:
```
xm.all_obs_info
```

[23]:

	ra	dec	ObsID	start	science_usable	duration	proprietary_end_date	revolution	proprietary_usable
0	64.925415	55.999440	0000110101	2001-08-19 07:05:23	True	0 days 09:08:33	2002-09-29	310	True
1	263.674950	-32.581670	0001730201	2001-03-09 12:44:21	True	0 days 04:44:43	2002-05-25	229	True
2	263.674950	-32.581670	0001730301	2001-03-09 17:30:16	True	0 days 02:36:02	2002-05-25	229	True
3	263.674950	-32.581670	0001730401	2001-03-09 09:41:25	True	0 days 03:00:59	2002-05-25	229	True
4	99.349995	6.135278	0001730501	2002-09-17 18:35:28	True	0 days 06:05:39	2004-12-31	508	True
...	...	...	...	...	...	...	...	...	...
17556	137.211167	55.379083	0924140101	2024-04-11 19:58:54	True	0 days 04:41:40	NaT	4458	False
17557	116.748000	-45.490722	0922040801	2024-04-14 06:14:32	True	0 days 06:43:20	NaT	4459	False
17558	179.719375	58.583583	0924141701	2024-04-14 02:14:33	True	0 days 02:13:20	NaT	4459	False
17559	153.948542	54.520361	0924141801	2024-04-13 19:51:03	True	0 days 05:48:20	NaT	4459	False
17560	287.956500	4.982722	0934990101	2024-04-14 14:56:52	True				

(continues on next page)

(continued from previous page)

```
17560 0 days 18:35:00                 NaT        4459              False

                      end
0     2001-08-19 16:13:56
1     2001-03-09 17:29:04
2     2001-03-09 20:06:18
3     2001-03-09 12:42:24
4     2002-09-18 00:41:07
...                   ...
17556 2024-04-12 00:40:34
17557 2024-04-14 12:57:52
17558 2024-04-14 04:27:53
17559 2024-04-14 01:39:23
17560 2024-04-15 09:31:52

[17561 rows x 10 columns]
```

[24]:
```
ch.all_obs_info
```

[24]:
```
              ra       dec  ObsID  science_usable  proprietary_usable  \
0      274.43140 -33.01883   6616            True                True
1       83.63292  22.01447   7587            True                True
2      202.50000  47.20000  13814            True                True
3      266.41667 -29.00781  13842            True                True
4      316.72458  38.74942  13651            True                True
...          ...       ...    ...             ...                 ...
23136  332.17000  45.74231  24644            True                True
23137   84.91458 -69.74361   1203            True                True
23138  332.17010  45.74230   1336            True                True
23139  350.85750  58.81483   1409            True                True
23140  332.17000  45.74231  24645            True                True

                          start                       end          duration  \
0     2006-02-24 04:33:41.000003 2007-09-21 00:26:11.000003 573 days 19:52:30
1     2007-02-03 09:58:57.000000 2008-07-21 21:53:57.000000 534 days 11:55:00
2     2012-09-20 07:21:41.999999 2012-09-22 12:47:41.999999   2 days 05:26:00
3     2012-07-21 11:52:41.000002 2012-07-23 17:08:41.000002   2 days 05:16:00
4     2012-02-13 20:18:26.999997 2012-02-16 00:51:26.999997   2 days 04:33:00
...                          ...                        ...              ...
23136 2020-09-29 01:23:47.000003 2020-09-29 01:23:47.000003   0 days 00:00:00
23137 1999-08-31 03:28:53.000003 1999-08-31 03:28:53.000003   0 days 00:00:00
23138 1999-10-03 21:48:53.000004 1999-10-03 21:48:53.000004   0 days 00:00:00
23139 1999-10-23 18:29:33.999999 1999-10-23 18:29:33.999999   0 days 00:00:00
23140 2020-10-25 12:05:32.000001 2020-10-25 12:05:32.000001   0 days 00:00:00

      proprietary_end_date target_category instrument grating data_mode
0               2007-02-28              GS     ACIS-S    HETG  CC_000A8
1               2008-02-06             SNR     ACIS-S    HETG  TE_0077C
2               2013-10-11             NGS     ACIS-S    NONE  TE_00958
3               2012-07-25             NGS     ACIS-S    HETG  TE_008D0
4               2013-02-21            MISC      HRC-S    LETG   DEFAULT
...                    ...             ...        ...     ...       ...
```

(continues on next page)

(continued from previous page)

```
23136           2020-09-30             CAL      HRC-I    NONE   DEFAULT
23137           2003-06-16              GS      HRC-I    NONE
23138           1999-12-14             CAL      HRC-I    NONE
23139           1999-12-09             SNR      HRC-I    NONE
23140           2020-10-27             CAL      HRC-S    NONE   SCENTER

[23141 rows x 13 columns]
```

[25]:
```
ea.all_obs_info
```

[25]:
```
              ra        dec   ObsID  science_usable               start  \
0     116.703297  42.008289  117048            True 2020-04-15 10:13:39
1     120.659341  42.008289  121048            True 2020-04-17 10:14:32
2     124.615385  42.008289  125048            True 2020-04-19 14:14:56
3     128.571429  42.008289  129048            True 2020-04-21 18:15:19
4     132.527473  42.008289  133048            True 2020-04-23 22:15:47
...          ...        ...     ...             ...                 ...
2442  220.000000 -87.045181  220177            True 2020-03-19 20:42:36
2443  260.000000 -87.045181  260177            True                 NaT
2444  300.000000 -87.045181  300177            True 2020-03-28 16:42:43
2445  340.000000 -87.045181  340177            True 2020-03-31 20:42:37
2446    0.000000 -90.000000  001180            True 2020-03-26 00:45:06

                     end         duration      ra_min      ra_max  dec_min  \
0    2020-04-19 02:13:48  3 days 16:00:09  114.725275  118.681319     40.5
1    2020-04-21 10:13:37  3 days 23:59:05  118.681319  122.637363     40.5
2    2020-04-23 14:14:20  3 days 23:59:24  122.637363  126.593407     40.5
3    2020-04-25 18:14:56  3 days 23:59:37  126.593407  130.549451     40.5
4    2020-04-28 02:15:10  4 days 03:59:23  130.549451  134.505495     40.5
...                  ...              ...         ...         ...      ...
2442 2020-04-03 04:43:25 14 days 08:00:49  200.000000  240.000000    -88.5
2443 2020-04-05 12:43:45              NaT  240.000000  280.000000    -88.5
2444 2020-04-09 04:42:26 11 days 11:59:43  280.000000  320.000000    -88.5
2445 2020-04-12 00:44:26 11 days 04:01:49  320.000000  360.000000    -88.5
2446 2020-04-06 12:45:53 11 days 12:00:47    0.000000  360.000000    -90.0

      dec_max                                          neigh_obs
0        43.5  114045,118045,122045,113048,121048,112051,1160...
1        43.5  118045,122045,117048,125048,116051,119051,1230...
2        43.5  122045,126045,121048,129048,123051,127051,0000...
3        43.5  126045,130045,125048,133048,127051,131051,0000...
4        43.5  130045,134045,129048,136048,131051,135051,0000...
...       ...                                              ...
2442    -85.5  001180,191174,214174,236174,259174,180177,2601...
2443    -85.5  001180,236174,259174,281174,304174,220177,3001...
2444    -85.5  001180,259174,281174,304174,326174,260177,3401...
2445    -85.5  001180,304174,326174,349174,011174,300177,0201...
2446    -88.5  020177,060177,100177,140177,180177,220177,2601...

[2447 rows x 12 columns]
```

[26]:
```
ecpv.all_obs_info
```

[26]:
```
             ra        dec   ObsID  science_usable               start  \
0    129.550000   1.500000  300007            True 2019-11-03 02:42:50
1    133.860000   1.500000  300008            True 2019-11-04 03:49:16
2    138.140000   1.500000  300009            True 2019-11-05 05:29:18
3    142.450000   1.500000  300010            True 2019-11-06 07:24:46
4    130.331300 -78.963400  300004            True 2019-11-16 23:14:40
..          ...        ...     ...             ...                 ...
165  284.146250 -37.909167  700008            True 2019-10-24 11:11:19
166  281.540771  79.873726  900060            True 2019-09-24 15:27:06
167  281.500275  79.885376  900068            True 2019-09-28 15:49:51
168  281.489410  79.888214  900069            True 2019-09-29 15:23:24
169  281.478271  79.891029  900070            True 2019-09-30 15:23:24

                    end  duration           Field_Name            Field_Type  \
0   2019-11-04 03:36:37   89627.0                EFEDS                SURVEY
1   2019-11-05 05:16:39   91643.0                EFEDS                SURVEY
2   2019-11-06 06:40:06   90648.0                EFEDS                SURVEY
3   2019-11-07 08:20:08   89722.0                EFEDS                SURVEY
4   2019-11-18 18:17:12  154952.0              ETA_CHA                SURVEY
..                  ...       ...                  ...                   ...
165 2019-10-25 08:55:52   78273.0  1RXS_J185635_375433  EXTRAGALACTIC_FIELDS
166 2019-09-24 21:30:32   21806.0               3C390_3  EXTRAGALACTIC_FIELDS
167 2019-09-28 21:30:32   20441.0               3C390_3  EXTRAGALACTIC_FIELDS
168 2019-09-29 21:30:37   22033.0               3C390_3  EXTRAGALACTIC_FIELDS
169 2019-09-30 21:30:32   22028.0               3C390_3  EXTRAGALACTIC_FIELDS

                   active_insts
0    TM1,TM2,TM3,TM4,TM5,TM6,TM7
1    TM1,TM2,TM3,TM4,TM5,TM6,TM7
2    TM1,TM2,TM3,TM4,TM5,TM6,TM7
3    TM1,TM2,TM3,TM4,TM5,TM6,TM7
4    TM1,TM2,TM3,TM4,TM5,TM6,TM7
..                           ...
165  TM1,TM2,TM3,TM4,TM5,TM6,TM7
166                          TM6
167                          TM6
168                          TM6
169                  TM5,TM6,TM7

[170 rows x 10 columns]
```

[27]:
```
nu.all_obs_info
```

[27]:
```
            ra      dec        ObsID  science_usable  proprietary_usable  \
0      83.8281 -69.2465  40001014013            True                True
1      83.7759 -69.2677  40001014016            True                True
2      83.8965 -69.2477  40001014023            True                True
3     340.6530  29.6985  60401031004            True                True
4     344.4048 -36.9765  60901012002            True               False
...        ...      ...          ...             ...                 ...
5477    0.0000   0.0000  20627025001            True                True
5478    0.0000   0.0000  20624008001            True                True
5479    0.0000   0.0000  20624001002            True                True
```

(continues on next page)

(continued from previous page)

```
5480    0.0000   0.0000  20601030002            True                True
5481    0.0000   0.0000  80802021002            True                True

                          start                         end  \
0    2013-06-29 01:16:07.183998 2013-07-05 08:51:07.184004
1    2014-04-22 21:06:07.183999 2014-04-29 09:31:07.183998
2    2014-08-01 23:46:07.183998 2014-08-08 02:41:07.184000
3    2018-11-28 22:21:09.184000 2018-12-08 17:16:09.183997
4    2023-11-02 06:06:09.184003 2023-11-10 11:41:09.184004
...                         ...                         ...
5477 2022-06-04 00:08:52.183997 2022-06-04 00:26:09.184004
5478 2022-06-04 00:26:09.184004 2022-06-04 02:01:09.184002
5479 2022-06-04 02:01:09.184002 2022-06-04 03:06:09.184003
5480 2022-06-04 03:06:09.184003 2022-06-04 03:26:09.184004
5481 2022-06-16 21:56:09.184001 2022-06-16 22:16:09.184002

                  duration proprietary_end_date target_category  \
0    6 days 07:35:00.000006  2015-09-17 00:00:00             SNR
1    6 days 12:24:59.999999  2015-09-17 00:00:00             SNR
2    6 days 02:55:00.000002  2015-09-17 00:00:00             SNR
3    9 days 18:54:59.999997  2019-12-25 00:00:00             AGN
4    8 days 05:35:00.000001  2024-05-20 00:00:00             AGN
...                     ...                  ...             ...
5477 0 days 00:17:17.000007  2022-06-13 00:00:00             SOL
5478 0 days 01:34:59.999998  2022-06-13 00:00:00             SOL
5479 0 days 01:05:00.000001  2022-06-13 00:00:00             SOL
5480 0 days 00:20:00.000001  2022-06-13 00:00:00             SOL
5481 0 days 00:20:00.000001  2023-01-11 00:00:00             TOO

          exposure_a      exposure_b        ontime_a        ontime_b  \
0    5 days 11:20:42 5 days 11:09:58 5 days 20:32:48 5 days 20:33:46
1    4 days 23:56:47 4 days 23:35:17 5 days 08:14:46 5 days 08:14:09
2    4 days 22:46:56 4 days 22:35:32 5 days 07:01:07 5 days 07:02:55
3    4 days 17:35:58 4 days 17:01:43 5 days 02:48:25 5 days 02:49:43
4    4 days 11:03:16 4 days 10:06:55 4 days 19:02:15 4 days 19:02:36
...              ...             ...             ...             ...
5477 0 days 00:00:00 0 days 00:00:00 0 days 00:00:00 0 days 00:00:00
5478 0 days 00:00:00 0 days 00:00:00 0 days 00:00:00 0 days 00:00:00
5479 0 days 00:00:00 0 days 00:00:00 0 days 00:00:00 0 days 00:00:00
5480 0 days 00:00:00 0 days 00:00:00 0 days 00:00:00 0 days 00:00:00
5481 0 days 00:00:00 0 days 00:00:00 0 days 00:00:00 0 days 00:00:00

      nupsdout  issue_flag
0         2119           1
1            0           1
2            0           0
3            0           0
4            0           0
...        ...         ...
5477         0           1
5478         0           1
5479         0           1
```

(continues on next page)

(continued from previous page)

```
5480         0           0
5481         0           0

[5482 rows x 16 columns]
```

[28]:
```
sw.all_obs_info
```

[28]:
```
               ra       dec        ObsID  science_usable  \
0       328.70739  16.89364  00173780007            True
1        39.95079 -25.21897  00131560002            True
2       228.46115  30.87659  00164268014            True
3       350.64958   5.74990  00148833003            True
4        75.27823  45.28513  00321174011            True
...           ...       ...          ...             ...
353399  225.01926 -35.01827  00067092005            True
353400   77.63318 -14.96722  00074164018            True
353401    7.30920 -73.71531  00048706113            True
353402   51.88545  26.38554  03111918013            True
353403   20.08971  -5.01057  00076384008            True

                          start                       end  \
0      2005-12-25 00:42:01.999999 2005-12-28 00:00:23.000000
1      2005-06-03 23:59:00.999998 2005-06-06 22:47:30.000002
2      2005-12-03 01:00:00.999996 2005-12-06 00:07:43.000000
3      2005-08-06 00:26:11.999996 2005-08-09 00:50:56.000003
4      2008-08-31 02:25:00.999998 2008-09-03 00:34:06.999998
...                          ...                       ...
353399 2005-01-23 13:07:01.000004 2005-01-23 23:15:55.999996
353400 2011-05-03 12:25:59.000002 2011-05-03 17:31:53.999999
353401 2019-03-04 23:56:36.000001 2019-03-05 00:51:04.999997
353402 2022-12-25 10:22:35.999999 2022-12-25 10:32:25.999999
353403 2023-05-24 11:05:38.000003 2023-05-24 12:42:01.999999

                     duration target_category           xrt_exposure  \
0      2 days 23:18:21.000001            MISC 1 days 02:44:09.014000
1      2 days 22:48:29.000004            MISC 0 days 22:36:12.946000
2      2 days 23:07:42.000004            MISC 0 days 19:38:55.384000
3      3 days 00:24:44.000007            MISC 0 days 19:04:07.125000
4            2 days 22:09:06             MISC 0 days 18:49:28.970000
...                       ...             ...                    ...
353399 0 days 10:08:54.999992            MISC        0 days 00:00:00
353400 0 days 05:05:54.999997            MISC        0 days 00:00:00
353401 0 days 00:54:28.999996            MISC        0 days 00:00:00
353402        0 days 00:09:50            MISC        0 days 00:00:00
353403 0 days 01:36:23.999996            MISC        0 days 00:00:00

                bat_exposure          uvot_exposure
0            1 days 00:04:10 1 days 02:40:01.030000
1            0 days 22:52:03 0 days 17:46:44.297000
2            0 days 19:59:15        0 days 00:00:00
3            0 days 19:23:36 0 days 18:21:43.023000
4      0 days 21:08:01.650000 0 days 18:24:15.039000
```

(continues on next page)

(continued from previous page)

```
...                       ...                       ...
353399        0 days 02:57:51        0 days 00:00:00
353400        0 days 00:05:31        0 days 00:00:00
353401        0 days 00:01:01 0 days 00:00:53.499000
353402        0 days 00:00:02        0 days 00:00:00
353403        0 days 00:07:14        0 days 00:00:00

[353404 rows x 11 columns]
```

[29]:
```
ra.all_obs_info
```

[29]:
```
             ra     dec        ObsID  science_usable      start        end  \
0     263.57088  67.500  RS930521N00            True 1990-07-11 1991-08-13
1     276.42533  67.500  RS930522N00            True 1990-07-11 1991-08-13
2      96.42533 -67.500  RS932908N00            True 1990-07-11 1991-08-13
3      83.57088 -67.500  RS932907N00            True 1990-07-11 1991-08-13
4     267.27100  61.875  RS930625N00            True 1990-07-11 1991-08-13
...         ...     ...          ...             ...        ...        ...
1373  289.08783 -61.875  RS932827N00            True 1990-09-10 1990-10-08
1374  350.17942 -33.750  RS932354N00            True 1990-11-09 1990-12-02
1375  278.17942 -61.875  RS932826N00            True 1990-09-04 1990-09-30
1376  273.75000 -45.000  RS932537N00            True 1990-09-07 1990-09-23
1377  280.46275 -50.625  RS932634N00            True 1990-09-13 1990-10-01

            duration target_category
0    0 days 11:19:06             ASK
1    0 days 11:18:11             ASK
2    0 days 05:47:48             ASK
3    0 days 05:47:39             ASK
4    0 days 03:44:35             ASK
...              ...             ...
1373 0 days 00:02:44             ASK
1374 0 days 00:02:42             ASK
1375 0 days 00:02:42             ASK
1376 0 days 00:02:42             ASK
1377 0 days 00:02:34             ASK

[1378 rows x 8 columns]
```

[30]:
```
rp.all_obs_info
```

[30]:
```
             ra      dec        ObsID  science_usable  \
0      163.1800  57.4800  RH701867A01            True
1      203.6500  37.9100  RH900717N00            True
2      163.1800  57.4800  RH701867A04            True
3      350.8700  58.8100  RH500444N00            True
4      163.1800  57.4800  RH701867A02            True
...         ...      ...          ...             ...
11426   84.2900 -80.4700  RP999998A02           False
11427   93.1800 -81.8300  RH150094N00           False
11428  218.1540 -44.2031  RH001034N00           False
11429  258.1383 -23.3850  RH800067M01           False
```

(continues on next page)

(continued from previous page)

```
11430  205.4270 -62.3400  RH001027N00           False

                          start                         end        duration  \
0     1995-04-15 23:24:16.000001 1995-05-11 14:24:47.000001 2 days 13:29:32
1     1997-06-04 16:13:00.999998 1997-07-13 22:26:45.000004 2 days 07:58:33
2     1997-04-15 21:51:16.999998 1997-04-28 16:37:45.999998 2 days 06:33:41
3     1995-12-23 22:18:36.999999 1996-02-01 10:17:53.999998 2 days 02:07:27
4     1996-05-01 02:09:33.000002 1996-05-29 15:31:24.999997 2 days 01:05:24
...                          ...                         ...            ...
11426 1991-10-20 04:07:51.999998 1991-11-01 22:43:43.000003 0 days 00:00:00
11427 1990-07-29 22:13:42.999997 1990-07-29 22:56:58.999998 0 days 00:00:00
11428 1997-08-28 00:29:42.000000 1997-08-28 01:09:58.999997 0 days 00:00:00
11429 1991-03-19 20:33:48.999997 1991-03-19 20:35:59.999997 0 days 00:00:00
11430 1997-08-25 04:54:58.000003 1997-08-25 05:18:28.000000 0 days 00:00:00

      instrument with_filter target_category         target_name  proc_rev  \
0            HRI           N              AGN        LOCKMAN HOLE         2
1            HRI           N             MISC         DEEP SURVEY         2
2            HRI           N              AGN        LOCKMAN HOLE         2
3            HRI           N              SNR               CAS A         2
4            HRI           N              AGN        LOCKMAN HOLE         2
...          ...         ...              ...                 ...       ...
11426      PSPCB           N              EGE          Idle Point         2
11427        HRI           N              STR  Calibration Source         2
11428        HRI           N             MISC                              2
11429        HRI           N              GCL   OPHIUCHUS CLUSTER         2
11430        HRI           N             MISC                              2

       fits_type
0      RFITS V4.
1        RDF 4_0
2      RFITS V3.
3        RDF 3_4
4      RFITS V4.
...          ...
11426    RDF 3_4
11427  RFITS V4.
11428    RDF 4_2
11429  RFITS V3.
11430    RDF 4_2

[11431 rows x 13 columns]
```

[31]:
```
su.all_obs_info
```

[31]:
```
            ra      dec      ObsID  science_usable                      start  \
0      91.1523 -86.6779  701018010            True 2006-04-13 16:24:07.999998
1      48.9864 -85.5003  404019010            True 2009-07-16 14:26:31.000001
2     239.2836 -79.2302  703059010            True 2008-10-13 15:35:23.999997
3     265.9438 -76.3446  705013010            True 2010-04-14 00:16:05.000002
4      74.6207 -75.2810  404036010            True 2009-06-14 01:51:18.000003
...        ...      ...        ...             ...                         ...
```

(continues on next page)

(continued from previous page)

```
3036  247.6537  82.9256  706004010           True 2011-04-16 00:38:47.000003
3037  116.7261  85.6829  804031010           True 2010-02-08 09:53:43.999999
3038  112.8158  85.6965  804030010           True 2010-02-08 18:18:12.000001
3039  112.8340  85.6974  804030020           True 2010-02-10 04:18:27.999997
3040  264.6080  87.3047  705012010           True 2010-04-26 23:41:56.000000

                          end               duration target_category  \
0    2006-04-14 01:52:19.000001 0 days 09:28:11.000003             EGS
1    2009-07-18 03:30:15.999998 1 days 13:03:44.999997              GS
2    2008-10-15 21:37:21.000000 2 days 06:01:57.000003             EGS
3    2010-04-14 16:45:11.000002        0 days 16:29:06             EGS
4    2009-06-16 13:27:19.000002 2 days 11:36:00.999999              GS
...                         ...                   ...             ...
3036 2011-04-16 10:23:15.000000 0 days 09:44:27.999997             EGS
3037 2010-02-08 18:15:19.000002 0 days 08:21:35.000003             GCL
3038 2010-02-09 12:00:25.000004 0 days 17:42:13.000003             GCL
3039 2010-02-11 06:00:24.000002 1 days 01:41:56.000005             GCL
3040 2010-04-27 10:16:15.000001 0 days 10:34:19.000001             EGS

                  xis0_expo  xis0_num_modes              xis1_expo  \
0    0 days 05:29:41.700000               2 0 days 05:29:41.700000
1    0 days 17:31:24.300000               2 0 days 17:31:24.300000
2           0 days 22:49:57               2 0 days 22:49:43.500000
3    0 days 11:46:23.600000               2 0 days 11:46:18.300000
4    1 days 05:42:37.800000               2 1 days 05:42:37.800000
...                     ...             ...                    ...
3036 0 days 05:13:55.700000               3        0 days 05:13:42
3037        0 days 05:19:56               1        0 days 05:19:56
3038 0 days 10:17:39.100000               2 0 days 10:17:39.100000
3039 0 days 14:36:01.100000               2 0 days 14:36:01.100000
3040 0 days 05:37:59.600000               2 0 days 05:37:59.600000

      xis1_num_modes              xis2_expo  xis2_num_modes  \
0                  2 0 days 05:29:25.700000               2
1                  2        0 days 00:00:00               0
2                  2        0 days 00:00:00               0
3                  2        0 days 00:00:00               0
4                  2        0 days 00:00:00               0
...              ...                    ...             ...
3036               2        0 days 00:00:00               0
3037               1        0 days 00:00:00               0
3038               2        0 days 00:00:00               0
3039               2        0 days 00:00:00               0
3040               2        0 days 00:00:00               0

                  xis3_expo  xis3_num_modes
0    0 days 05:29:33.700000               2
1    0 days 17:31:24.300000               2
2    0 days 22:49:35.500000               2
3           0 days 11:46:29               2
4    1 days 05:42:29.800000               2
...                     ...             ...
```

(continues on next page)

(continued from previous page)

```
3036 0 days 05:13:55.600000               2
3037        0 days 05:19:56               1
3038 0 days 10:17:39.100000               2
3039 0 days 14:36:01.100000               2
3040 0 days 05:37:59.600000               2

[3041 rows x 16 columns]
```

[32]:
```
asca.all_obs_info
```

[32]:
```
            ra      dec     ObsID  science_usable                   start  \
0     288.0116   5.0542  48002000            True 2000-03-23 05:52:13.440
1     253.5540  39.8158  78002000            True 2000-03-01 12:08:12.480
2      14.3115 -22.4431  77036000            True 1999-12-03 11:50:38.400
3     203.8856 -34.3378  77003000            True 1999-07-19 00:40:19.200
4     266.4864 -28.9429  48004000            True 2000-03-11 00:08:29.760
...        ...      ...       ...             ...                       ...
3069   87.7568 -32.2321  77010010            True 1999-10-11 13:47:08.160
3070  150.6550 -58.6295  15000090            True 1993-08-05 18:31:58.080
3071  335.1555 -24.7349  66013000            True 1998-11-16 16:24:48.960
3072  333.8738 -17.8223  77002000            True 1999-11-17 04:45:41.760
3073  244.9340 -15.8493  40022010            True 1993-08-16 08:40:42.240

                        end                duration target_category  \
0    2000-04-04 00:00:25.920 11 days 18:08:12.480000             XRB
1    2000-03-11 00:00:51.840  9 days 11:52:39.360000             AGN
2    1999-12-15 19:00:28.800 12 days 07:09:50.400000             AGN
3    1999-07-29 15:40:27.840 10 days 15:00:08.640000             AGN
4    2000-03-22 00:00:43.200 10 days 23:52:13.440000             XRB
...                      ...                    ...             ...
3069 1999-10-11 14:19:32.160         0 days 00:32:24             AGN
3070 1993-08-05 19:12:43.200  0 days 00:40:45.120000              GS
3071 1998-11-18 17:34:39.360  2 days 01:09:50.400000             NGS
3072 1999-11-19 01:35:54.240  1 days 20:50:12.480000             AGN
3073 1993-08-16 10:20:21.120  0 days 01:39:38.880000             XRB

        sis_exposure    gis_exposure
0    5 days 07:25:04 5 days 14:47:28
1    4 days 00:45:04 5 days 04:32:16
2    4 days 08:14:08 5 days 00:42:40
3    4 days 02:03:12 4 days 16:33:20
4    3 days 22:37:04 4 days 07:30:24
...              ...             ...
3069 0 days 00:06:40 0 days 00:06:40
3070 0 days 00:06:56 0 days 00:05:52
3071 0 days 14:38:24 0 days 00:00:00
3072 0 days 16:30:24 0 days 00:00:00
3073 0 days 00:53:04 0 days 00:00:00

[3074 rows x 10 columns]
```

[33]:
```
inte.all_obs_info
```

[33]:

```
               ra       dec         ObsID  science_usable  proprietary_usable  \
0       294.97232 -89.65222  218500510010            True                True
1       117.47166 -89.57144  218600510010            True                True
2       166.19875 -89.54264  229200810010            True                True
3       130.24533 -89.51414  209300110010            True                True
4       125.61337 -89.47669  209300370010            True                True
...           ...       ...           ...             ...                 ...
198281  341.90903  88.90408  260500570010            True                True
198282  153.50116  88.91008  260500660010            True                True
198283  213.64879  89.06850  260900540010            True                True
198284  268.11816  89.32161  260700500010            True                True
198285    6.62067  89.56795  261300370010            True                True

                          start                       end  \
0      2020-01-22 20:40:39.751343 2020-01-22 22:17:46.754676
1      2020-01-25 11:40:02.881811 2020-01-25 12:38:50.883829
2      2020-11-02 08:42:24.928042 2020-11-02 09:41:14.930022
3      2019-05-21 07:51:13.039554 2019-05-21 08:24:33.041123
4      2019-05-21 22:51:03.082145 2019-05-21 23:24:24.083715
...                           ...                       ...
198281 2023-02-12 08:47:47.388930 2023-02-12 09:17:47.390084
198282 2023-02-12 13:36:15.400319 2023-02-12 14:06:15.401552
198283 2023-02-22 18:05:50.978884 2023-02-22 19:29:20.982045
198284 2023-02-17 10:30:57.695328 2023-02-17 11:00:58.696476
198285 2023-03-05 00:34:23.640085 2023-03-05 01:04:23.641170

                    duration target_category  jemx1_exposure  jemx2_exposure  \
0      0 days 01:37:07.003333            MISC 0 days 01:36:12 0 days 01:36:12
1      0 days 00:58:48.002018            MISC 0 days 00:57:57 0 days 00:57:57
2      0 days 00:58:50.001980            MISC 0 days 00:58:00 0 days 00:58:00
3      0 days 00:33:20.001569            MISC 0 days 00:32:25 0 days 00:32:25
4      0 days 00:33:21.001570            MISC 0 days 00:32:30 0 days 00:32:30
...                       ...             ...             ...             ...
198281 0 days 00:30:00.001154            MISC 0 days 00:29:11 0 days 00:29:11
198282 0 days 00:30:00.001233            MISC 0 days 00:28:40 0 days 00:28:40
198283 0 days 01:23:30.003161            MISC 0 days 01:21:28 0 days 01:21:00
198284 0 days 00:30:01.001148            MISC 0 days 00:29:10 0 days 00:29:10
198285 0 days 00:30:00.001085            MISC 0 days 00:29:04 0 days 00:29:04

        isgri_exposure picsit_exposure    spi_exposure scw_ver
0      0 days 01:36:10 0 days 00:28:29 0 days 01:36:12     001
1      0 days 00:57:55 0 days 00:57:27 0 days 00:57:57     001
2      0 days 00:57:44 0 days 00:57:29 0 days 00:58:00     001
3      0 days 00:32:23 0 days 00:32:04 0 days 00:32:25     001
4      0 days 00:32:28 0 days 00:32:02 0 days 00:32:30     001
...                ...             ...             ...     ...
198281 0 days 00:29:10 0 days 00:28:44 0 days 00:29:11     001
198282 0 days 00:29:09 0 days 00:00:00 0 days 00:29:10     001
198283 0 days 01:21:09 0 days 00:28:33 0 days 01:20:57     001
198284 0 days 00:29:10 0 days 00:28:47 0 days 00:29:12     001
198285 0 days 00:29:48 0 days 00:29:19 0 days 00:29:52     001

[198286 rows x 15 columns]
```

Filter Array

This is unlikely to ever be accessed directly by the user, but is what defines the observations that a mission currently deems to be accepted/selected. It is a boolean numpy array with a length equal to the number of observations in the all_obs_info dataframe, a `True` value means the observation is accepted and a `False` value means it is excluded; all observations start off as accepted.

Various filtering methods can be used to adjust the filter array and set the observations which are to be downloaded/included in a DAXA archive, depending on your particular sample and science case.

It is also possible to manually set this filter array, as is demonstrated below:

[34]:
```
# The filter array defaults to all True, so all observations are accepted
xm.filter_array
```

[34]:
```
array([ True,  True,  True, ...,  True,  True,  True])
```

[35]:
```
# Demonstrating manually setting a filter array - it must be boolean and be the same␣
↪length as the
#  'all_obs_info' table, otherwise it will not be accepted
demo_filt_arr = np.full(len(xm.all_obs_info), True)
demo_filt_arr[0] = False
xm.filter_array = demo_filt_arr
xm.filter_array
```

[35]:
```
array([False,  True,  True, ...,  True,  True,  True])
```

3.1.4 Selecting relevant observations

Few users will wish to download, process, and maintain **complete** observation archives, preferring to just locate data that may be relevant to the sources that they are studying. This can be achieved with the use of several filtering methods which are built into all DAXA missions.

Here we introduce the different filtering methods that are currently implemented for DAXA missions, but we do not provide detailed demonstrations of their use; that is left to specific case studies designed to show scientists with different needs how DAXA can be used in ways that are most relevant to them.

Filtering on ObsID

The most basic filtering method available can be used when you already know which observation(s) you are interested in - if you have the ObsID(s) you can just pass them to the `filter_on_obs_ids` method, and select only that data:

[36]:
```
help(xm.filter_on_obs_ids)
```

```
Help on method filter_on_obs_ids in module daxa.mission.base:

filter_on_obs_ids(allowed_obs_ids: Union[str, List[str]]) method of daxa.mission.xmm.
↪XMMPointed instance
    This filtering method will select only observations with IDs specified by the␣
↪allowed_obs_ids argument.

    Please be aware that filtering methods are cumulative, so running another method␣
↪will not remove the
    filtering that has already been applied, you can use the reset_filter method for␣
↪that.
```

(continues on next page)

(continued from previous page)

```
    :param str/List[str] allowed_obs_ids: The ObsID (or list of ObsIDs) that you wish to␣
↪be let
        through the filter.
```

Filtering on position

Arguably the most useful type of filtering supported by DAXA missions, the `filter_on_positions` method allows us to search for observations that are relevant to specific positions on the sky (generally these will represent particular objects). A search can be performed either for a single position, or for a whole sample.

The `search_distance` argument controls how close the central coordinate of an observation must be to a search position for that observation to be accepted. The default search positions are defined by the field-of-view of the telescope (if a mission has multiple instruments with different field-of-views then each instrument will be searched with the correct field-of-view). The user may also specify their own search distance value (or values), and it is also possible to specify a different search distance for every position.

The `return_pos_obs_info` argument controls whether a dataframe is returned that links passed positions to particular observations that are relevant to them - this dataframe would not include positions that are determined to have no relevant observations.

[37]:
```
help(xm.filter_on_positions)
```

```
Help on method filter_on_positions in module daxa.mission.base:

filter_on_positions(positions: Union[list, numpy.ndarray, astropy.coordinates.sky_
↪coordinate.SkyCoord], search_distance: Union[astropy.units.quantity.Quantity, float,␣
↪int, list, numpy.ndarray, dict] = None, return_pos_obs_info: bool = False) ->␣
↪Optional[pandas.core.frame.DataFrame] method of daxa.mission.xmm.XMMPointed instance
    This method allows you to filter the observations available for a mission based on a␣
↪set of coordinates for
    which you wish to locate observations. The method searches for observations by the␣
↪current mission that have
    central coordinates within the distance set by the search_distance argument.

    Please be aware that filtering methods are cumulative, so running another method␣
↪will not remove the
    filtering that has already been applied, you can use the reset_filter method for␣
↪that.

    :param list/np.ndarray/SkyCoord positions: The positions for which you wish to␣
↪search for observations. They
        can be passed either as a list or nested list (i.e. [r, d] OR [[r1, d1], [r2,␣
↪d2]]), a numpy array, or
        an already defined SkyCoord. If a list or array is passed then the coordinates␣
↪are assumed to be in
        degrees, and the default mission frame will be used.
    :param Quantity/float/int/list/np.ndarray/dict search_distance: The distance within␣
↪which to search for
        observations by this mission. Distance may be specified either as an Astropy␣
↪Quantity that can be
```

(continues on next page)

(continued from previous page)

```
        converted to degrees (a float/integer will be assumed to be in units of degrees),
↪ as a dictionary of
        quantities/floats/ints where the keys are names of different instruments␣
↪(possibly with different field
        of views), or as a non-scalar Quantity, list, or numpy array with one entry per␣
↪set of coordinates (for
        when you wish to use different search distances for each object). The default is␣
↪None, in which case a
        value of 1.2 times the approximate field of view defined for each instrument␣
↪will be used; where different
        instruments have different FoVs, observation searches will be undertaken on an␣
↪instrument-by-instrument
        basis using the different field of views.
    :param bool return_pos_obs_info: Allows this method to return information (in the␣
↪form of a Pandas dataframe)
        which identifies the positions which have been associated with observations, and␣
↪the observations they have
        been associated with. Default is False.
    :return: If return_pos_obs_info is True, then a dataframe containing information on␣
↪which ObsIDs are relevant
        to which positions will be returned. If return_pos_obs_info is False, then None␣
↪will be returned.
    :rtype: Union[None,pd.DataFrame]
```

Filtering on name

If you are interested in data relevant to named objects, you can pass the name(s) to the `filter_on_name` method. It will use a lookup service (specifically Sesame), to locate coordinates for the object(s), and then pass that to the `filter_on_positions` method.

Bear in mind that you are reliant on the lookup service having accurate central coordinates for the named object, so it could be worth checking with your own coordinates and using `filter_on_positions` directly if you can't find any observations!

[38]:
```
help(xm.filter_on_name)
```

```
Help on method filter_on_name in module daxa.mission.base:

filter_on_name(object_name: Union[str, List[str]], search_distance: Union[astropy.units.
↪quantity.Quantity, float, int, list, numpy.ndarray, dict] = None, parse_name: bool =␣
↪False) method of daxa.mission.xmm.XMMPointed instance
    This method wraps the 'filter_on_positions' method, and allows you to filter the␣
↪mission's observations so
    that it contains data on a single (or a list of) specific objects. The names are␣
↪passed by the user, and
    then parsed into coordinates using the Sesame resolver. Those coordinates and the␣
↪search distance are
    then used to find observations that might be relevant.

    :param str/List[str] object_name: The name(s) of objects you would like to search␣
↪for.
```

(continues on next page)

(continued from previous page)

```
    :param Quantity/float/int/list/np.ndarray/dict search_distance: The distance within␣
↪which to search for
        observations by this mission. Distance may be specified either as an Astropy␣
↪Quantity that can be
        converted to degrees (a float/integer will be assumed to be in units of degrees),
↪ as a dictionary of
        quantities/floats/ints where the keys are names of different instruments␣
↪(possibly with different field
        of views), or as a non-scalar Quantity, list, or numpy array with one entry per␣
↪set of coordinates (for
        when you wish to use different search distances for each object). The default is␣
↪None, in which case a
        value of 1.2 times the approximate field of view defined for each instrument␣
↪will be used; where different
        instruments have different FoVs, observation searches will be undertaken on an␣
↪instrument-by-instrument
        basis using the different field of views.
    :param bool parse_name: Whether to attempt extracting the coordinates from the name␣
↪by parsing with a regex.
        For objects catalog names that have J-coordinates embedded in their names, e.g.,
        'CRTS SSS100805 J194428-420209', this may be much faster than a Sesame query for␣
↪the same object name.
```

Filtering on target type

It is possible to filter observations based on the type of object that was the original target of the observation. **Warning: this should be used with significant caution, as our object taxonomy may not be granular enough to represent all different types of astronomical objects, and conversions between the different target types used by different missions and our target types is imperfect!**

Here we display the DAXA source type taxonomy, the short form codes on the left are what should be passed to the `filter_on_target_type` method - the user may pass either a single target type, or a list of them:

[39]:
```
xm.show_allowed_target_types()
```

```
| Target Type   | Description                                 |
| AGN           | Active Galaxies and Quasars                 |
| BLZ           | Blazars                                     |
| CV            | Cataclysmic Variables                       |
| CAL           | Calibration Observation (possibly of objects) |
| EGS           | Extragalactic Surveys                       |
| GCL           | Galaxy Clusters                             |
| GS            | Galactic Survey                             |
```

(continues on next page)

(continued from previous page)

ASK	All Sky Survey
MAG	Magnetars and Rotation-Powered Pulsars
NGS	Normal and Starburst Galaxies
NS	Neutron stars and Black Holes
STR	Non-degenerate and White Dwarf Stars
OAGN	Obscured Active Galaxies and Quasars
SNE	Non-ToO Supernovae
SNR	Supernova Remnants and Galactic diffuse
SOL	Solar System Observations
ULX	Ultra-luminous X-ray Sources
XRB	X-ray Binaries
TOO	Targets of Opportunity
EGE	Extended galactic or extragalactic
MISC	Catch-all for other sources

[40]:
```
help(xm.filter_on_target_type)
```

```
Help on method filter_on_target_type in module daxa.mission.base:

filter_on_target_type(target_type: Union[str, List[str]]) method of daxa.mission.xmm.
↪XMMPointed instance
    This method allows the filtering of observations based on what type of object their␣
↪target source was. It
    is only supported for missions that have that data available, and will raise an␣
↪exception for those
    missions that don't support this filtering.

    WARNING: You should not trust these target types without question, they are the␣
↪result of crude mappings, and
    some may be incorrect. They also don't take into account sources that might␣
↪serendipitously appear in
    a particular observation.

    :param str/List[str] target_type: The types of target source you would like to find␣
↪observations of. For
        allowed types, please use the 'show_allowed_target_types' method. Can either be␣
↪a single type, or
```

(continues on next page)

(continued from previous page)

```
        a list of types.
```

Filtering on time

Observations can be filtered on **when** they were taken, with the user specifying a time frame (defined by a start and end date-time) from which they wish to select observations. By default any observation that coincides with that time window (either starting in it, ending in it, or starting and ending outside but being taken during it) will be selected - it is also possible to require that an observation must have taken place entirely within the time window.

Warning: Observations of survey missions like eRASS1DE and ROSATAllSky may repeatedly visit a particular location, and all that data may be incorporated as one observation, meaning that there may not constant coverage in such an observation

[41]:
```
help(xm.filter_on_time)
```

```
Help on method filter_on_time in module daxa.mission.base:

filter_on_time(start_datetime: datetime.datetime, end_datetime: datetime.datetime, over_
↪run: bool = True) method of daxa.mission.xmm.XMMPointed instance
    This method allows you to filter observations for this mission based on when they␣
↪were taken. A start
    and end time are passed by the user, and observations that fall within that window␣
↪are allowed through
    the filter. The exact behaviour of this filtering method is controlled by the over_
↪run argument, if set
    to True then observations with a start or end within the search window will be␣
↪selected, but if False
    then only observations with a start AND end within the window are selected.

    Please be aware that filtering methods are cumulative, so running another method␣
↪will not remove the
    filtering that has already been applied, you can use the reset_filter method for␣
↪that.

    :param datetime start_datetime: The beginning of the time window in which to search␣
↪for observations.
    :param datetime end_datetime: The end of the time window in which to search for␣
↪observations.
    :param bool over_run: This controls whether selected observations have to be␣
↪entirely within the passed
        time window or whether either a start or end time can be within the search␣
↪window. If set
        to True then observations with a start or end within the search window will be␣
↪selected, but if False
        then only observations with a start AND end within the window are selected.␣
↪Default is True.
```

Filtering on time & position

A method of filtering that combines positional and temporal filtering, so that observations of a particular position, within a particular time window, can be located. The user **does not** have to filter for one position-time combination at a time - they may pass a set of positions, with a corresponding set of time windows, and find all the observations that fulfill those requirements.

The `filter_on_positions_at_time` method supports the same arguments passed to `filter_on_positions` and `filter_on_time`, which set search distances from the passed coordinate (`search_distance`), whether a dataframe linking particular input coordinates to particular selected observations (`return_obs_info`), and whether only observations that start and end within the specified time period should be considered (`over_run`).

[42]:
```
help(xm.filter_on_positions_at_time)
```

```
Help on method filter_on_positions_at_time in module daxa.mission.base:

filter_on_positions_at_time(positions: Union[list, numpy.ndarray, astropy.coordinates.
↪sky_coordinate.SkyCoord], start_datetimes: Union[numpy.ndarray, datetime.datetime],␣
↪end_datetimes: Union[numpy.ndarray, datetime.datetime], search_distance: Union[astropy.
↪units.quantity.Quantity, float, int, list, numpy.ndarray, dict] = None, return_obs_
↪info: bool = False, over_run: bool = True) method of daxa.mission.xmm.XMMPointed␣
↪instance
    This method allows you to filter the observations available for a mission based on a␣
↪set of coordinates for
    which you wish to locate observations that were taken within a certain time frame.␣
↪The method spatially
    searches for observations that have central coordinates within the distance set by␣
↪the search_distance
    argument, and temporally by start and end times passed by the user; and observations␣
↪that fall within that
    window are allowed through the filter.

    The exact behaviour of the temporal filtering method is controlled by the over_run␣
↪argument, if set
    to True then observations with a start or end within the search window will be␣
↪selected, but if False
    then only observations with a start AND end within the window are selected.

    Please be aware that filtering methods are cumulative, so running another method␣
↪will not remove the
    filtering that has already been applied, you can use the reset_filter method for␣
↪that.

    :param list/np.ndarray/SkyCoord positions: The positions for which you wish to␣
↪search for observations. They
        can be passed either as a list or nested list (i.e. [r, d] OR [[r1, d1], [r2,␣
↪d2]]), a numpy array, or
        an already defined SkyCoord. If a list or array is passed then the coordinates␣
↪are assumed to be in
        degrees, and the default mission frame will be used.
    :param np.array(datetime)/datetime start_datetimes: The beginnings of time windows␣
↪in which to search for
        observations. There should be one entry per position passed.
    :param np.array(datetime)/datetime end_datetimes: The endings of time windows in␣
↪which to search for
```

(continues on next page)

(continued from previous page)

```
        observations. There should be one entry per position passed.
    :param Quantity/float/int/list/np.ndarray/dict search_distance: The distance within␣
↪which to search for
        observations by this mission. Distance may be specified either as an Astropy␣
↪Quantity that can be
        converted to degrees (a float/integer will be assumed to be in units of degrees),
↪ as a dictionary of
        quantities/floats/ints where the keys are names of different instruments␣
↪(possibly with different field
        of views), or as a non-scalar Quantity, list, or numpy array with one entry per␣
↪set of coordinates (for
        when you wish to use different search distances for each object). The default is␣
↪None, in which case a
        value of 1.2 times the approximate field of view defined for each instrument␣
↪will be used; where different
        instruments have different FoVs, observation searches will be undertaken on an␣
↪instrument-by-instrument
        basis using the different field of views.
    :param bool return_obs_info: Allows this method to return information (in the form␣
↪of a Pandas dataframe)
        which identifies the positions which have been associated with observations, in␣
↪the specified time
        frame, and the observations they have been associated with. Default is False.
    :param bool over_run: This controls whether selected observations have to be␣
↪entirely within the passed
        time window or whether either a start or end time can be within the search␣
↪window. If set
        to True then observations with a start or end within the search window will be␣
↪selected, but if False
        then only observations with a start AND end within the window are selected.␣
↪Default is True.
```

3.1.5 Downloading data

Once the user has decided upon a set of observations, for a particular mission, that are relevant to their research - the next step is often to download them. This section specifies what can be downloaded, and how to download it.

What can be downloaded?

The exact data that can be downloaded depends on what a particular mission has made available on their online archive - Event lists are available for all missions bar INTEGRAL, and most missions support the acquisition of pre-generated images, but not all (INTEGRAL and eROSITA CalPV) for instance).

We make a distinction between 'raw' and 'pre-processed' data, which is necessarily fuzzy due to the disparate natures of the various data archives we have to deal with. Broadly speaking '**raw**' data means either absolutely raw files that need to be processed into initial event lists (the case with XMMPointed) or just event lists (uncleaned and pre-cleaned); '**pre-processed**' data includes pre-processed data products such as images, exposure maps, and background maps.

This means that the user can either set up an archive using the pre-processed data, or re-process data using DAXA interfaces to the various telescope backend software packages.

Pre-processed products are downloaded by default (in addition to the raw data), but when setting up an Archive you can choose between using pre-processed data or re-processing the raw data; if you do not wish to acquire pre-processed data products, you can pass `download_products=False` to the download method (see below).

Event lists are considered raw data (i.e. not pre-processed) for most missions, and will always be downloaded

XMM-Newton

Currently we only support the acquisition of raw data.

Chandra

The following data products can be downloaded:

- Full FoV image (if an Archive is constructed including pre-processed Chandra data, this is what is included).
- High-resolution central image.

eROSITA All-Sky DR1 (German Half)

The following data products can be downloaded:

- Images
- Exposure maps
- Background maps

eROSITA Calibration and Performance Verification

No pre-processed data products are available, only event lists.

NuSTAR-Pointed

The following data products can be downloaded:

- Images

Swift

The following data products can be downloaded (*no products are available for BAT, only raw data*):

- Images (XRT and UVOT)
- Exposure maps (XRT and UVOT)

ROSAT All-Sky Survey

The following data products can be downloaded:

- Images
- Exposure maps

ROSAT-Pointed

The following data products can be downloaded:

- Images
- Exposure maps (only PSPC, no HRI)

Suzaku

The following data products can be downloaded:

- Images

ASCA

The following data products can be downloaded:

- Images (SIS0 + SIS1, and GIS2 + GIS3)
- Exposure maps
- Lightcurves (not included in Archive constructed from pre-processed data)
- Spectra (not included in Archive constructed from pre-processed data)

INTEGRAL-Pointed

No pre-processed data (nor even event lists) are available to download for INTEGRAL, only raw data and calibration files.

How can it be downloaded?

The selected data can be downloaded using the `download` method that is built into each mission class.

If multiple observations have been selected, then downloads can be multi-threaded; this can sometimes offer a speed benefit, though only if the archive and user internet connections are not saturated by the download. The number of cores used can be set via the `num_cores` argument; by defualt this is set to the NUM_CORES DAXA constant, which can either be set by the user in the DAXA configuration file/by setting the value of daxa.NUM_CORES, or will be 90% of the cores of the system.

Note that it is not necessary to manually activate the download method if you will be creating a DAXA Archive from the filtered mission objects, as that will be done on archive initialisation -see the Archive tutorial for more information.

Here we demonstrate a simple XMM download, as well as a Chandra download that includes pre-generated images (standard Chandra reprocessing scripts can be used on this downloaded data):

[43]:
```
xm.filter_on_obs_ids('0201903501')
xm.download(num_cores=1)

ch.filter_on_obs_ids('3205')
ch.download(num_cores=1, download_products=True)
```

```
Downloading XMM-Newton Pointed data: 100%|| 1/1 [00:32<00:00, 32.13s/it]
Downloading Chandra data: 100%|| 1/1 [00:31<00:00, 31.23s/it]
```

3.1.6 Getting paths to downloaded mission data products

If pre-processed data have been downloaded, it is possible to use methods built into the mission class to retrieve the paths to various data products. **If you intend on using an XGA archive, using these methods should not be necessary** as the data will be moved to the Archive processed data storage structure, but it may still be useful if you just wish to use DAXA to download data.

There are four different get methods:

- `get_evt_list_path` - to retrieve event lists.
- `get_image_path` - to retrieve images.
- `get_expmap_path` - to retrieve exposure maps.
- `get_background_path` - to retrieve background maps.

They are very easy to use - only the ObsID, instrument (in most cases, not necessary if the mission has only one instrument per ObsID), and lower/upper energy bound (for images, exposure maps, and background maps - though if the products only have one energy band it will be completed automatically) need to be provided. They also provide helpful, detailed, error messages if what is requested isn't possible.

Here we retrive the path to the event list (note that we do not need to pass the instrument name, as Chandra has only one per ObsID):

[44]:
```
ch.get_evt_list_path('3205')
```

[44]:
```
'/Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_output/chandra_raw/3205/
↪primary/acisf03205N006_evt2.fits'
```

We can also retrieve the path to the full-FoV image (again we don't need to pass instrument in this specific instance, nor do we need to pass energy bounds because Chandra only supplies one):

[45]:
```
ch.get_image_path('3205')
```

[45]:
```
'/Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_output/chandra_raw/3205/
↪primary/acisf03205N006_full_img2.fits'
```

A more typical way of using this function would be this:

[46]:
```
ch.get_image_path('3205', Quantity(0.5, 'keV'), Quantity(7, 'keV'), 'ACIS-I')
```

[46]:
```
'/Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_output/chandra_raw/3205/
↪primary/acisf03205N006_full_img2.fits'
```

The get methods for exposure and background map paths operate identically to the image path method, though will show an error for Chandra:

[47]:
```
ch.get_expmap_path('3205', Quantity(0.5, 'keV'), Quantity(7, 'keV'), 'ACIS-I')
```

```
---------------------------------------------------------------------------
PreProcessedNotSupportedError             Traceback (most recent call last)
Cell In [47], line 1
----> 1 ch.get_expmap_path('3205', Quantity(0.5, 'keV'), Quantity(7, 'keV'), 'ACIS-I')

File ~/code/DAXA/daxa/mission/base.py:2034, in BaseMission.get_expmap_path(self, obs_id,␣
↪lo_en, hi_en, inst)
   2022 """
   2023 A get method that provides the path to a downloaded pre-generated exposure map␣
↪for the current mission (if
   2024 available). This method will not work if pre-processed data have not been␣
↪downloaded.
   (...)
   2031 :rtype: str
   2032 """
   2033 if self._template_exp_name is None:
-> 2034     raise PreProcessedNotSupportedError("This mission ({m}) does not support the␣
↪download of pre-processed "
   2035                                         "exposure maps, so a path cannot be "
   2036                                         "provided.".format(m=self.pretty_name))
   2038 if lo_en is not None:
   2039     # We make sure that the provided energy bounds are in keV
   2040     lo_en = lo_en.to('keV')

PreProcessedNotSupportedError: This mission (Chandra) does not support the download of␣
↪pre-processed exposure maps, so a path cannot be provided.
```

3.1.7 Saving mission state

It is possible to save the current 'state' of a mission object to a file - which in turn can be used to reload a mission class and have it be in that same state. This is primarily used by internal DAXA methods, to allow saved archives to reinstate their constituent mission classes when they are loaded back in.

However, it is also possible to do this manually, in order to reload missions as they were when they were saved. The save file could also be shared with other users, to allow them to create a mission in the same state.

The state of the mission includes the filtering procedures performed upon it, and the current filtered data - this means that the observations selected when the mission was saved will be reloaded exactly, and the same filtering steps can be re-applied on reload to update the mission (if more observations have become available).

Saving a mission state is simple - call the `save()` method and pass a root storage directory and optionally a file name (if this is not supplied the save file will be the mission name + '_state.json'):

[48]:
```
xm.save("demo_save/", "demo_xmm_save.json")
```

A mission can then be reinstated from the file on declaration:

[49]:
```
reinstated_xm = XMMPointed(save_file_path="demo_save/demo_xmm_save.json")
reinstated_xm.filtered_obs_info
```

```
/Users/dt237/code/DAXA/daxa/mission/xmm.py:83: UserWarning: 140 of the 17701␣
↪observations located for this mission have been removed due to NaN RA or Dec values
  self._fetch_obs_info()
```

[49]:
```
              ra      dec       ObsID               start  science_usable  \
3527  149.592285 -11.0597  0201903501 2004-06-17 10:38:28            True

            duration proprietary_end_date  revolution  proprietary_usable  \
3527 0 days 04:08:33           2004-06-17         828                True

                     end
3527 2004-06-17 14:47:01
```

If some time has passed since the mission state was saved, and you wish to re-run the original filtering procedures to check if new data are available, you can call the `update_filtering()` method.

3.2 Data Acquisition Case Studies

3.2.1 Observations of Messier 51 from many telescope archives

This case study searches for X-ray observations of the Messier 51 galaxy, taken by a variety of X-ray telescopes. We chose this galaxy because it is active (and thus likely X-ray bright), and nearby - this makes it a priority target for many X-ray observatories. This process could equally be applied to any named object (or object at a specific set of coordinates).

Import Statements

[1]:
```
from warnings import warn

from daxa.mission import XMMPointed, Chandra, ASCA, Suzaku, ROSATPointed, ROSATAllSky
from daxa.archive import Archive
from daxa.exceptions import NoObsAfterFilterError
```

Other Tutorials

These case studies are meant to be highly specific examples of how you might acquire data for a particular science case, they do not provide general instruction on how to use DAXA missions or archives. We instead direct you to:

- Using DAXA missions - Here we explain what DAXA mission classes are and how to use them to select only the data you need.
- Creating a DAXA archive - This explains how to create an archive, load an existing archive, and the various properties and features of DAXA archives.
- Processing telescope data - The processing tutorials for different missions are presented here, though there may not yet be processing support for all missions.

Reading through these should give you a good understanding of how DAXA can be used to acquire, organise, and process multi-mission X-ray datasets for your specific use case.

Defining missions

We create instances of the XMM, Chandra, ASCA, Suzaku, ROSAT Pointed, and ROSAT All-Sky missions in order to search their archives - other missions are supported by DAXA (and can be found in the missions tutorial), but these are a subset likely to have observations of M51:

[2]:
```
xm = XMMPointed()
ch = Chandra()
asc = ASCA()
su = Suzaku()
rp = ROSATPointed()
ra = ROSATAllSky()
```

```
/Users/dt237/code/DAXA/daxa/mission/xmm.py:83: UserWarning: 140 of the 17697␣
↪observations located for this mission have been removed due to NaN RA or Dec values
  self._fetch_obs_info()
/Users/dt237/code/DAXA/daxa/mission/asca.py:91: UserWarning: 5 of the 3079 observations␣
↪located for ASCA have been removed due to all instrument exposures being zero.
  self._fetch_obs_info()
/Users/dt237/code/DAXA/daxa/mission/suzaku.py:96: UserWarning: 14 of the 3055␣
↪observations located for Suzaku have been removed due to all instrument exposures␣
↪being zero.
  self._fetch_obs_info()
```

Searching for observations

In this instance we use the `filter_on_name` method to search for observations of M51 - this will use the Sesame name resolver to look-up the coordinates for the object. Alternatively, we could use the `filter_on_positions` method and supply the coordinate ourselves:

[3]:
```
xm.filter_on_name("M51")
ch.filter_on_name("M51")
asc.filter_on_name("M51")
rp.filter_on_name("M51")
ra.filter_on_name("M51")
```

```
/Users/dt237/code/DAXA/daxa/mission/base.py:1075: UserWarning: Chandra FoV are difficult␣
↪to define, as they can be strongly dependant on observation mode; as such take these␣
↪as very approximate.
  fov = self.fov
/Users/dt237/code/DAXA/daxa/mission/base.py:97: UserWarning: There are multiple chosen␣
↪instruments SIS0, SIS1, GIS2, GIS3 for asca with different FoVs, but they observe␣
↪simultaneously. As such the search distance has been set to the largest FoV of the␣
↪chosen instruments.
  any_ret = change_func(*args, **kwargs)
```

We have deliberately separated the Suzaku search, as we are aware that it will not find any matching data - as such we'll use this to highlight that the standard Python exception-catching statements can be used to stop a failure to find data derailing your script (for instance you might wish to iterate through a list of missions and have a try-except statement like this:

[4]:
```
try:
    su.filter_on_name("M51")
```

(continues on next page)

(continued from previous page)

```
except NoObsAfterFilterError as err:
    warn(err.message, stacklevel=2)
```

```
/opt/anaconda3/envs/daxa_dev/lib/python3.9/site-packages/IPython/core/interactiveshell.
↪py:3433: UserWarning: The positional search has returned no Suzaku observations.
  exec(code_obj, self.user_global_ns, self.user_ns)
```

Example of observations identified from filtering

We can use the `filtered_obs_info` property to retrieve the information table describing the accepted observations:

[5]:
```
rp.filtered_obs_info
```

[5]:
```
          ra   dec        ObsID  science_usable                      start  \
368   202.47  47.2  RH600601N00            True 1994-06-18 13:11:33.000000
917   202.47  47.2  RP600158N00            True 1991-11-28 16:07:59.999998
2904  202.47  47.2  RH600062A03            True 1994-05-22 05:24:01.000002
3096  202.47  47.2  RH600062A01            True 1992-05-22 23:44:46.000000
3174  202.47  47.2  RH600062A00            True 1991-12-07 09:46:57.000003
3303  202.47  47.2  RH601115N00            True 1997-12-26 23:37:19.000001

                          end        duration instrument with_filter  \
368  1994-06-24 08:04:24.000001 0 days 10:05:23        HRI           N
917  1991-12-13 17:50:20.999999 0 days 06:39:16      PSPCB           N
2904 1994-05-23 07:16:07.000000 0 days 02:36:32        HRI           N
3096 1992-06-05 21:50:42.999999 0 days 02:27:11        HRI           N
3174 1992-01-10 05:10:12.999996 0 days 02:22:21        HRI           N
3303 1997-12-30 07:46:24.000001 0 days 02:15:04        HRI           N

     target_category  target_name  proc_rev  fits_type
368              NGS         M 51         2    RDF 3_4
917              NGS  N5194/N5195         2    RDF 3_4
2904             NGS          M51         2    RDF 3_4
3096             NGS          M51         2  RFITS V3.
3174             NGS          M51         2  RFITS V3.
3303             NGS                      2    RDF 4_2
```

Defining an Archive

The filtered missions can then be used to define an archive containing the selected data:

[6]:
```
arch = Archive('M51', [xm, ch, asc, rp, ra])
arch.info()
```

```
Downloading XMM-Newton Pointed data: 100%|| 16/16 [07:37<00:00, 28.60s/it]
Downloading Chandra data: 100%|| 28/28 [02:57<00:00,  6.32s/it]
Downloading ASCA data: 100%|| 2/2 [01:00<00:00, 30.23s/it]
Downloading ROSAT Pointed data: 100%|| 6/6 [00:11<00:00,  1.93s/it]
Downloading RASS data: 100%|| 2/2 [00:09<00:00,  4.55s/it]


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

(continues on next page)

(continued from previous page)

```
Number of missions - 5
Total number of observations - 54
Beginning of earliest observation - 1990-07-11 00:00:00
End of latest observation - 2022-01-08 17:51:21

-- XMM-Newton Pointed --
   Internal DAXA name - xmm_pointed
   Chosen instruments - M1, M2, PN
   Number of observations - 16
   Fully Processed - False

-- Chandra --
   Internal DAXA name - chandra
   Chosen instruments - ACIS-I, ACIS-S, HRC-I, HRC-S
   Number of observations - 28
   Fully Processed - False

-- ASCA --
   Internal DAXA name - asca
   Chosen instruments - SIS0, SIS1, GIS2, GIS3
   Number of observations - 2
   Fully Processed - False

-- ROSAT Pointed --
   Internal DAXA name - rosat_pointed
   Chosen instruments - PSPCB, PSPCC, HRI
   Number of observations - 6
   Fully Processed - False

-- RASS --
   Internal DAXA name - rosat_all_sky
   Chosen instruments - PSPC
   Number of observations - 2
   Fully Processed - False
------------------------------------------------------
```

3.2.2 X-ray observations of a sample of clusters selected from eFEDS

This case study searches for X-ray observations of the eFEDS sample of galaxy clusters, and is intended to show how DAXA missions can be used to create multi-mission archives of data for large samples of objects; this case study would apply just as well to any type of X-ray source. The clusters we use will **all** have eFEDS observations, as they were selected from that survey, but many should also have serendipitous XMM and Chandra observations.

Import Statements

[1]:
```
import numpy as np
import pandas as pd
from astropy.coordinates import SkyCoord

from daxa.mission import XMMPointed, Chandra, eROSITACalPV
from daxa.archive import Archive
```

Other Tutorials

These case studies are meant to be highly specific examples of how you might acquire data for a particular science case, they do not provide general instruction on how to use DAXA missions or archives. We instead direct you to:

- Using DAXA missions - Here we explain what DAXA mission classes are and how to use them to select only the data you need.
- Creating a DAXA archive - This explains how to create an archive, load an existing archive, and the various properties and features of DAXA archives.
- Processing telescope data - The processing tutorials for different missions are presented here, though there may not yet be processing support for all missions.

Reading through these should give you a good understanding of how DAXA can be used to acquire, organise, and process multi-mission X-ray datasets for your specific use case.

Sample

We read in the sample of galaxy clusters we'll be searching for observations of - they were selected from the eROSITA Final-Equatorial Depth Survey (eFEDS), and many should have XMM and Chandra observations:

[2]:
```
samp = pd.read_csv("samp_files/efeds_xray_cluster_candidates.csv")
samp.head(5)
```

[2]:

	ID	ID_SRC	RA	DEC	EXT_LIKE	DET_LIKE	\
0	eFEDS J082626.6-003429	28993	126.610799	-0.574787	8.486203	5.029723	
1	eFEDS J082751.8-002853	11248	126.965471	-0.481638	12.791595	27.865910	
2	eFEDS J082808.8-001003	4800	127.036645	-0.167715	28.492811	62.512480	
3	eFEDS J082820.6-000721	4169	127.085556	-0.122752	42.376125	81.378350	
4	eFEDS J082840.6-000500	7991	127.169202	-0.083552	18.438711	37.515427	

	z	z_type	T_300kpc	T_300kpc_L	...	L_500kpc_L	L_500kpc_U	\
0	0.161110	0	-1.000000	-1.000000	...	-1.000000e+00	3.924900e+42	
1	0.257160	0	-1.000000	-1.000000	...	-1.000000e+00	1.061100e+43	
2	0.076155	0	0.885294	0.786329	...	2.339100e+42	3.424700e+42	
3	0.844900	0	-1.000000	-1.000000	...	1.890600e+44	2.762200e+44	
4	0.319705	0	-1.000000	-1.000000	...	1.419400e+43	2.065400e+43	

	Lbol_500kpc	Lbol_500kpc_L	Lbol_500kpc_U	F_500kpc	F_500kpc_L	\
0	-1.000000e+00	-1.000000e+00	1.782000e+43	-1.000000e+00	-1.000000e+00	
1	-1.000000e+00	-1.000000e+00	1.796300e+43	-1.000000e+00	-1.000000e+00	
2	4.402300e+42	3.640200e+42	5.339000e+42	1.976100e-13	1.624400e-13	
3	6.207600e+44	5.089800e+44	8.013600e+44	7.796600e-14	6.555100e-14	
4	3.993600e+43	3.175500e+43	4.931700e+43	5.510200e-14	4.443600e-14	

(continues on next page)

(continued from previous page)

```
     F_500kpc_U  SNR_MAX  R_SNR_MAX_ARCMIN
0  6.040400e-14     1.32            0.8011
1  4.903300e-14     3.07            0.7393
2  2.382600e-13     8.78            2.6312
3  8.961400e-14     7.30            1.4667
4  6.466500e-14     5.61            1.3388

[5 rows x 34 columns]
```

[3]:
```
coords = SkyCoord(samp['RA'].values, samp['DEC'].values, unit='deg')
```

Defining missions

[4]:
```
er = eROSITACalPV()
xm = XMMPointed()
ch = Chandra()
```

```
/Users/dt237/code/DAXA/daxa/mission/xmm.py:83: UserWarning: 140 of the 17697␣
↪observations located for this mission have been removed due to NaN RA or Dec values
  self._fetch_obs_info()
```

Searching for observations

We search for observations around the coordinates of our cluster sample - it is worth noting that we are using the default FoV radius/half-width multiplied by a factor of 1.2 as a search radius. You may also set this value yourself, for each instrument individually (for missions like Chandra) or for all instruments, using the `search_distance` argument and an astropy quantity in units convertible to degrees.

Also, if `return_obs_info` is set to True, a dataframe is returned from the method to allow the user to link specific ObsIDs to particular entries in our original sample table. The dataframe contains a 'pos_ind' column, which contains indexes corresponding to the input positions (i.e. the 4th entry of pos would have index 3), it also contains ObsIDs matched to that coordinate.

[5]:
```
er_assoc = er.filter_on_positions(coords, return_pos_obs_info=True)
xm_assoc = xm.filter_on_positions(coords, return_pos_obs_info=True)
ch_assoc = ch.filter_on_positions(coords, return_pos_obs_info=True)
```

```
/Users/dt237/code/DAXA/daxa/mission/base.py:1095: UserWarning: A field-of-view cannot be␣
↪easily defined for eROSITACalPV and this number is the approximate half-length of an␣
↪eFEDS section, the worst case separation - this is unnecessarily large for pointed␣
↪observations, and you should make your own judgement on a search distance.
  fov = self.fov
/Users/dt237/code/DAXA/daxa/mission/base.py:1095: UserWarning: Chandra FoV are difficult␣
↪to define, as they can be strongly dependant on observation mode; as such take these␣
↪as very approximate.
  fov = self.fov
```

Exploring the selected data

We can examine the `filtered_obs_info` property (see the missions tutorial for a fuller explanation):

[6]:
```
er.filtered_obs_info
```

[6]:
```
       ra  dec   ObsID  science_usable               start  \
0  129.55  1.5  300007            True 2019-11-03 02:42:50
1  133.86  1.5  300008            True 2019-11-04 03:49:16
2  138.14  1.5  300009            True 2019-11-05 05:29:18
3  142.45  1.5  300010            True 2019-11-06 07:24:46

                  end  duration Field_Name Field_Type
0 2019-11-04 03:36:37   89627.0      EFEDS     SURVEY
1 2019-11-05 05:16:39   91643.0      EFEDS     SURVEY
2 2019-11-06 06:40:06   90648.0      EFEDS     SURVEY
3 2019-11-07 08:20:08   89722.0      EFEDS     SURVEY
```

[7]:
```
xm.filtered_obs_info
```

[7]:
```
              ra       dec       ObsID               start  science_usable  \
3629   130.198335  0.763056  0202940101 2004-05-08 16:32:31            True
3630   130.198335  0.763056  0202940201 2004-10-26 17:36:30            True
5137   136.602495  0.965556  0402780801 2007-04-20 15:52:25            True
7355   138.022125  0.483667  0602340201 2009-12-01 01:27:07            True
7385   138.723585  4.442889  0602830401 2009-11-16 01:38:06            True
...           ...       ...         ...                 ...             ...
17163  140.539000  3.775000  0920001301 2023-10-27 11:23:39            True
17164  140.539000  3.775000  0920002501 2023-10-27 07:26:59            True
17166  129.890833 -1.679000  0920000901 2023-10-29 11:14:11            True
17167  129.890833 -1.679000  0920002601 2023-10-29 07:19:11            True
17285  139.106000  1.385000  0920980301 2023-11-30 05:12:14            True

             duration proprietary_end_date  revolution  proprietary_usable  \
3629  0 days 08:03:28           2006-03-04         808                True
3630  0 days 06:28:36           2006-03-04         894                True
5137  0 days 07:33:33           2008-05-20        1348                True
7355  0 days 07:11:13           2010-12-17        1827                True
7385  0 days 02:17:00           2010-12-01        1820                True
...               ...                  ...         ...                 ...
17163 0 days 03:50:00           2024-11-13        4374               False
17164 0 days 03:56:40           2024-11-13        4374               False
17166 0 days 04:46:40           2024-11-27        4375               False
17167 0 days 03:55:00           2024-11-27        4375               False
17285 0 days 12:55:00           2024-12-28        4391               False

                      end
3629  2004-05-09 00:35:59
3630  2004-10-27 00:05:06
5137  2007-04-20 23:25:58
7355  2009-12-01 08:38:20
7385  2009-11-16 03:55:06
...                   ...
17163 2023-10-27 15:13:39
```

(continues on next page)

(continued from previous page)

```
17164 2023-10-27 11:23:39
17166 2023-10-29 16:00:51
17167 2023-10-29 11:14:11
17285 2023-11-30 18:07:14

[113 rows x 10 columns]
```

[8]:
```
ch.filtered_obs_info
```

[8]:
```
              ra      dec  ObsID  science_usable  proprietary_usable  \
1702   136.91667 -0.69994  17084            True                True
2148   144.43375  2.76094  20348            True                True
3684   140.94625  4.04850  23835            True                True
3885   137.31625  3.91186  23162            True                True
4605   137.86458  5.84778  14958            True                True
...          ...      ...    ...             ...                 ...
20574  145.23833  3.40033  11451            True                True
20890  135.83750  4.96056  11448            True                True
21229  143.81125  3.59603   5705            True                True
21409  130.27833  3.20189  13347            True                True
21559  137.35000  0.03639   5703            True                True

                          start                       end        duration  \
1702  2015-01-10 00:32:52.999996 2015-01-10 16:07:22.999996 0 days 15:34:30
2148  2018-01-22 17:22:56.000004 2018-01-23 07:18:06.000004 0 days 13:55:10
3684  2022-05-16 13:22:45.999995 2022-05-17 00:28:45.999995 0 days 11:06:00
3885  2020-02-20 03:27:14.999999 2020-02-20 14:10:44.999999 0 days 10:43:30
4605  2012-12-30 10:27:00.999997 2012-12-30 19:54:40.999997 0 days 09:27:40
...                          ...                       ...             ...
20574 2010-01-04 14:03:12.999998 2010-01-04 14:39:12.999998 0 days 00:36:00
20890 2010-01-04 15:01:54.999999 2010-01-04 15:35:44.999999 0 days 00:33:50
21229 2005-03-07 04:46:24.000001 2005-03-07 05:16:24.000001 0 days 00:30:00
21409 2011-12-11 12:21:05.000000 2011-12-11 12:47:05.000000 0 days 00:26:00
21559 2004-12-28 09:24:24.999999 2004-12-28 09:46:54.999999 0 days 00:22:30

      proprietary_end_date target_category instrument grating data_mode
1702            2016-01-12             AGN     ACIS-S    NONE  TE_006C8
2148            2019-01-23             NGS     ACIS-S    NONE  TE_005C6
3684            2023-05-17             AGN     ACIS-S    NONE  TE_007F2
3885            2021-02-20             AGN     ACIS-S    NONE  TE_0065E
4605            2013-12-31             AGN     ACIS-S    NONE  TE_009C8
...                    ...             ...        ...     ...       ...
20574           2011-01-05             AGN     ACIS-S    NONE  TE_008FC
20890           2011-01-05             AGN     ACIS-S    NONE  TE_008FC
21229           2006-03-08             AGN     ACIS-S    NONE  TE_006B0
21409           2012-12-12             AGN     ACIS-S    NONE  TE_0076A
21559           2005-12-29             AGN     ACIS-S    NONE  TE_006B0

[90 rows x 13 columns]
```

We can also use the tables that were returned from the search methods to match observations to specific objects:

[9]:
```
xm_assoc['name'] = samp.loc[xm_assoc['pos_ind'].values.astype(int), 'ID'].values
xm_assoc
```

[9]:
```
    pos_ind              pos_ra              pos_dec                ObsIDs  \
0        45   129.3486854874346     1.40366630348438            0903700101
1        47  129.48802147587307  -1.7049345640737046  0920001601,0920002401
2        52   129.5393896036211  -2.0807673161720084  0920001601,0920002401
3        54  129.54957579257595  -1.9929684395330654  0920001601,0920002401
4        56   129.5732303438282   -2.284546510019436  0920001601,0920002401
..      ...                 ...                  ...                   ...
110     469  141.93603300405417    4.941812984556808  0901870201,0901871201
111     491  142.47325179257808   0.4670133405850062            0802220601
112     521  143.75324046077222   0.9047856577567157  0920000801,0920002301
113     522  143.80457579850201   0.7993807870395315  0920000801,0920002301
114     523   143.8362603912554   0.5801534818576921  0920000801,0920002301

                       name
0    eFEDS J083723.7+012413
1    eFEDS J083757.2-014217
2    eFEDS J083809.5-020450
3    eFEDS J083812.0-015934
4    eFEDS J083817.6-021704
..                      ...
110  eFEDS J092744.6+045631
111  eFEDS J092953.6+002801
112  eFEDS J093500.8+005417
113  eFEDS J093513.1+004757
114  eFEDS J093520.7+003448

[115 rows x 5 columns]
```

[10]:
```
ch_assoc['name'] = samp.loc[ch_assoc['pos_ind'].values.astype(int), 'ID'].values
ch_assoc
```

[10]:
```
    pos_ind              pos_ra              pos_dec       ObsIDs  \
0         9  127.79401199149834   1.9378392793445995        19734
1        12  127.85804170868693    1.925897930055847        19734
2        16  127.97336088473476   1.4252588642845447        19734
3        18  128.11694107938197  -0.1156644071035969        23614
4        22  128.31476848695075    0.106508629638464        23614
..      ...                 ...                  ...          ...
187     537  144.43405705683216    2.760045545939909        20348
188     538   144.6271718639683    4.256395791583636        18099
189     539    144.909595641361    4.371716376951857        26035
190     540   145.0245934774472    3.224725957866028  22270,11451
191     541  145.03072779231115    3.965204479523988  26035,22270

                       name
0    eFEDS J083110.6+015616
1    eFEDS J083125.9+015533
2    eFEDS J083153.6+012531
3    eFEDS J083228.1-000656
4    eFEDS J083315.6+000623
```

(continues on next page)

(continued from previous page)

```
..                      ...
187  eFEDS J093744.2+024536
188  eFEDS J093830.5+041523
189  eFEDS J093938.3+042218
190  eFEDS J094005.9+031329
191  eFEDS J094007.3+035755

[192 rows x 5 columns]
```

[11]:
```
er_assoc['name'] = samp.loc[er_assoc['pos_ind'].values.astype(int), 'ID'].values
er_assoc
```

[11]:
```
    pos_ind              pos_ra               pos_dec  ObsIDs  \
0         0  126.61080512258118   -0.5747925179258094  300007
1         1  126.96547689559516   -0.4816435127681215  300007
2         2  127.03665115606483   -0.1677211251616217  300007
3         3  127.08556215338794   -0.1227575917789436  300007
4         4   127.1692087960137  -0.08355838266078222  300007
..      ...                 ...                  ...     ...
537     537  144.43406327204823    2.7600402745537114  300010
538     538  144.62717799852624     4.256390530427462  300010
539     539  144.90960176777745    4.3717111308643295  300010
540     540   145.0245996650002    3.2247207179507202  300010
541     541  145.03073393985625    3.9651992399385168  300010

                      name
0    eFEDS J082626.6-003429
1    eFEDS J082751.8-002853
2    eFEDS J082808.8-001003
3    eFEDS J082820.6-000721
4    eFEDS J082840.6-000500
..                      ...
537  eFEDS J093744.2+024536
538  eFEDS J093830.5+041523
539  eFEDS J093938.3+042218
540  eFEDS J094005.9+031329
541  eFEDS J094007.3+035755

[542 rows x 5 columns]
```

Defining an archive

The filtered missions can then be used to define an archive containing the selected data:

[12]:
```
arch = Archive('eFEDS_clusters', [er, xm, ch])
arch.info()
```

```
/Users/dt237/code/DAXA/daxa/archive/base.py:133: UserWarning: The raw data for this␣
↪mission have already been downloaded.
  mission.download()
/Users/dt237/code/DAXA/daxa/archive/base.py:133: UserWarning: Proprietary data have been␣
↪selected, but no credentials provided; as such the proprietary data have been excluded␣
↪from download and further processing.
```

(continues on next page)

(continued from previous page)

```
  mission.download()
Downloading XMM-Newton Pointed data: 100%|| 100/100 [00:18<00:00,  5.38it/s]
Downloading Chandra data: 100%|| 90/90 [03:01<00:00,  2.02s/it]
```

```
------------------------------------------------------
Number of missions - 3
Total number of observations - 194
Beginning of earliest observation - 1999-11-02 17:31:43.000001
End of latest observation - 2023-02-26 11:24:55.000001

-- eROSITACalPV --
   Internal DAXA name - erosita_calpv
   Chosen instruments - TM1, TM2, TM3, TM4, TM5, TM6, TM7
   Number of observations - 4
   Fully Processed - False

-- XMM-Newton Pointed --
   Internal DAXA name - xmm_pointed
   Chosen instruments - M1, M2, PN
   Number of observations - 100
   Fully Processed - False

-- Chandra --
   Internal DAXA name - chandra
   Chosen instruments - ACIS-I, ACIS-S, HRC-I, HRC-S
   Number of observations - 90
   Fully Processed - False
------------------------------------------------------
```

3.2.3 Finding X-ray observations for a sample of SNe 30 days either side of discovery

This case study searches for X-ray observations (from XMM, Chandra, and Swift; but it is applicable to all DAXA missions) of Type 1a Supernovae locations, within a time window around each SNe's discovery date. This process is applicable to any search for an observation of a specific place at a specific time (or a sample of places at a set of different times).

Import Statements

[1]:
```
import numpy as np
import pandas as pd
from astropy.coordinates import SkyCoord
from datetime import datetime, timedelta

from daxa.mission import XMMPointed, Chandra, Swift
from daxa.archive import Archive
```

Other Tutorials

These case studies are meant to be highly specific examples of how you might acquire data for a particular science case, they do not provide general instruction on how to use DAXA missions or archives. We instead direct you to:

- Using DAXA missions - Here we explain what DAXA mission classes are and how to use them to select only the data you need.
- Creating a DAXA archive - This explains how to create an archive, load an existing archive, and the various properties and features of DAXA archives.
- Processing telescope data - The processing tutorials for different missions are presented here, though there may not yet be processing support for all missions.

Reading through these should give you a good understanding of how DAXA can be used to acquire, organise, and process multi-mission X-ray datasets for your specific use case.

Sample

We read in a demonstrative sample, which consists of 1000 randomly selected Type 1a Supernovae from the transient name server (TNS); there are many columns, but the only ones we require are position (Right-Ascension and Declination) and the discovery date (which we will search around):

[2]:
```
samp = pd.read_csv("samp_files/sn1a_samp.csv")
samp.head(5)
```

[2]:
```
    objid name_prefix     name          ra  declination  redshift  typeid  \
0  133295          SN  2023ock  232.474130    66.075011     0.039     3.0
1   74545          SN   2021dj  208.777227    54.304902     0.070     3.0
2   56011          SN  2020enj  155.237795     0.573764     0.104     3.0
3   74416          SN    2021E   62.978428    16.803008     0.050     3.0
4   49744          SN  2019xck   97.070105    23.608591     0.035     3.0

    type  reporting_groupid reporting_group  ...  source_group  \
0  SN Ia               48.0             ZTF  ...           ZTF
1  SN Ia               48.0             ZTF  ...           ZTF
2  SN Ia               18.0           ATLAS  ...         ATLAS
3  SN Ia               74.0          ALeRCE  ...           ZTF
4  SN Ia               74.0          ALeRCE  ...           ZTF

             discoverydate discoverymag  discmagfilter  filter  \
0  2023-07-26 08:16:47.000      19.2400          110.0       g
1  2021-01-01 11:24:00.000      19.2500          111.0       r
2  2020-03-16 10:59:31.200      19.2350           72.0  orange
3  2021-01-01 03:34:56.997      19.9476          110.0       g
4  2019-12-19 09:51:42.000      19.3648          110.0       g

                                        reporters        time_received  \
0  C. Fremling (Caltech) on behalf of the Zwicky ...  2023-07-28 07:24:04
1  C. Fremling (Caltech) on behalf of the Zwicky ...  2021-01-03 22:01:11
2  J. Tonry, L. Denneau, A. Heinze, H. Weiland, H...  2020-03-16 18:48:30
3  F. Forster, F.E. Bauer, A. Munoz-Arancibia, G...  2021-01-01 16:43:48
4  F. Forster, F.E. Bauer, G. Pignata, J. Arredon...  2019-12-19 16:00:58

                    internal_names         creationdate         lastmodified
```

(continues on next page)

(continued from previous page)

```
0           ZTF23aauqmys, ATLAS23pvm  2023-07-28 07:24:05  2023-08-06 05:49:07
1   ZTF21aaabucr, ATLAS21aza, PS21xg  2021-01-03 22:01:13  2021-02-07 11:04:35
2  ATLAS20hza, PS20agg, ZTF20aaubotx  2020-03-16 18:48:32  2020-03-16 18:48:32
3           ZTF21aaaalaf, ATLAS21ash  2021-01-01 16:43:53  2021-01-13 00:38:47
4          ZTF19aczeomw, ATLAS19bdqm  2019-12-19 16:01:09  2019-12-19 16:01:09

[5 rows x 21 columns]
```

We read out the coordinates into an astropy coordinate object, and set up the time windows we will be searching for each SNe - for this demonstration we will search for X-ray observations 30 days either side of the disovery date:

[3]:
```
pos = SkyCoord(samp['ra'].values, samp['declination'].values, unit='deg')
start_times = np.array([datetime.strptime(dd, "%Y-%m-%d %H:%M:%S.%f") -␣
↪timedelta(days=30)
                        for dd in samp['discoverydate'].values])
end_times = np.array([datetime.strptime(dd, "%Y-%m-%d %H:%M:%S.%f") + timedelta(days=30)
                      for dd in samp['discoverydate'].values])
```

Defining missions

We create instances of the XMM, Chandra, and Swift missions in order to search their archives - Swift is the most likely to have many matching observations, as it acts as a transient follow-up telescope, but XMM and Chandra are workhorses and may have some observations that we might want to explore:

[4]:
```
xm = XMMPointed()
ch = Chandra()
sw = Swift()
```

```
/Users/dt237/code/DAXA/daxa/mission/xmm.py:83: UserWarning: 140 of the 17697␣
↪observations located for this mission have been removed due to NaN RA or Dec values
  self._fetch_obs_info()
/Users/dt237/code/DAXA/daxa/mission/swift.py:101: UserWarning: 598 of the 353616␣
↪observations located for Swift have been removed due to all instrument exposures being␣
↪zero.
  self._fetch_obs_info()
/Users/dt237/code/DAXA/daxa/mission/swift.py:101: UserWarning: 17 of the 353616␣
↪observations located for Swift have been removed due to all chosen instrument (XRT,␣
↪BAT) exposures being zero.
  self._fetch_obs_info()
```

Searching for observations

We will make use of the DAXA filtering method that allows us to search for observations of a particular coordinate, within a particular time frame, for a whole sample. It is called the same way for the three missions we are using, we pass the positions, start times, and end times, and the keyword arguments have the following meanings:

- **return_obs_info** - If True, a dataframe is returned from the method to allow the user to link specific ObsIDs to particular entries in our original sample table. The dataframe contains a 'pos_ind' column, which contains indexes corresponding to the input positions (i.e. the 4th entry of pos would have index 3), it also contains ObsIDs matched to that coordinate and time window.

- **over_run** - If True, observations that start or end outside of the specified time window are accepted. If False, they are not.

[5]:
```
xm_assoc = xm.filter_on_positions_at_time(pos, start_times, end_times, return_obs_
↪info=True, over_run=True)
ch_assoc = ch.filter_on_positions_at_time(pos, start_times, end_times, return_obs_
↪info=True, over_run=True)
sw_assoc = sw.filter_on_positions_at_time(pos, start_times, end_times, return_obs_
↪info=True, over_run=True)
```

```
/Users/dt237/code/DAXA/daxa/mission/base.py:1389: UserWarning: Every value in the filter␣
↪array is False, meaning that no observations remain.
  self.filter_array = np.full(self.filter_array.shape, False)
/Users/dt237/code/DAXA/daxa/mission/base.py:1075: UserWarning: Chandra FoV are difficult␣
↪to define, as they can be strongly dependant on observation mode; as such take these␣
↪as very approximate.
  fov = self.fov
/Users/dt237/code/DAXA/daxa/mission/base.py:1389: UserWarning: Every value in the filter␣
↪array is False, meaning that no observations remain.
  self.filter_array = np.full(self.filter_array.shape, False)
/Users/dt237/code/DAXA/daxa/mission/base.py:97: UserWarning: There are multiple chosen␣
↪instruments XRT, BAT for swift with different FoVs, but they observe simultaneously.␣
↪As such the search distance has been set to the largest FoV of the chosen instruments.
  any_ret = change_func(*args, **kwargs)
/Users/dt237/code/DAXA/daxa/mission/base.py:1389: UserWarning: Every value in the filter␣
↪array is False, meaning that no observations remain.
  self.filter_array = np.full(self.filter_array.shape, False)
```

Identified observations

We will now use the returns from the filtering methods to highlight the observations which have been identified as fulfilling our criteria:

[6]:
```
xm_assoc['sn_name'] = samp.loc[xm_assoc['pos_ind'].values.astype(int), 'name'].values
xm_assoc
```

[6]:

	pos_ind	pos_ra	pos_dec	ObsIDs	sn_name
30	414	158.42781249083353	39.49061071686476	0824030101	2018hus

[7]:
```
ch_assoc['sn_name'] = samp.loc[ch_assoc['pos_ind'].values.astype(int), 'name'].values
ch_assoc
```

[7]:

	pos_ind	pos_ra	pos_dec	ObsIDs	sn_name
10	45	219.741280784	51.0827058066	21697	2020hyi
41	142	234.553792	39.732811		
44	160	161.2694701	2.31901835		
114	429	186.495176661	7.23543298389		
166	664	206.843520969	26.384687742		
174	706	222.24025	18.326689		
196	783	193.82175	2.897311		
219	894	222.905086576	18.9261358766		

(continues on next page)

(continued from previous page)

```
41                                                  22528  2019rmq
44                                                  22494  2020kxf
114                                                 23771  2021ita
166  27023,27026,27021,27022,27024,27029,27020,27806  2023hrk
174                                                 22659  2021min
196                                                 23563  2021pkz
219                                                 26962  2023jgq
```

[8]:
```
sw_assoc['sn_name'] = samp.loc[sw_assoc['pos_ind'].values.astype(int), 'name'].values
sw_assoc
```

[8]:
```
    pos_ind              pos_ra             pos_dec  \
1         1  208.77722506346083   54.30490205010956
2         2  155.23780171544678  0.5737592936995243
7         7  0.8142941780660411   16.14571038076381
10       10  250.60306782108816   78.91498537452249
17       17  219.49858839000424   9.388540565458925
..      ...                 ...                 ...
873     955   267.9480465961222   44.90420978372445
879     961   295.4186780442415  -21.26268560353943
887     970   39.01071994163291  43.472059484607946
905     990   347.3879247185516   15.65927267376877
909     995  355.94687610671747  51.241493712166886

                                                ObsIDs  sn_name
1    00013608061,00013608056,00013608059,0001360805...   2021dj
2                             00095611001,00075041008  2020enj
7    00014427002,00014427004,00014427001,0001442700...  2021rhu
10                                        00095179001  2019kyz
17     00013556003,00013556005,00013556002,00013556004  2020lil
..                                                ...      ...
873               00074924063,00074924061,00075754012  2022ucs
879    00016144004,00016144003,00016144001,00016144002  2023mvl
887  00010346005,00010346003,00010346008,0001034601...  2017hjy
905  00013713001,00013713003,00013713002,0001371300...  2020szr
909  00081310002,00035031184,00035031190,0003503118...  2020rsi

[119 rows x 5 columns]
```

Defining an archive

The filtered missions can then be used to define an archive containing the selected data:

[9]:
```
arch = Archive('sne_search', [xm, ch, sw])
arch.info()
```

```
Downloading XMM-Newton Pointed data: 100%|| 1/1 [00:22<00:00, 22.25s/it]
Downloading Chandra data: 100%|| 15/15 [00:38<00:00,  2.56s/it]
Downloading Swift data: 100%|| 454/454 [07:33<00:00,  1.00it/s]


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

(continues on next page)

(continued from previous page)

```
Number of missions - 3
Total number of observations - 470
Beginning of earliest observation - 2016-02-21 03:25:58
End of latest observation - 2024-01-05 07:08:51

-- XMM-Newton Pointed --
   Internal DAXA name - xmm_pointed
   Chosen instruments - M1, M2, PN
   Number of observations - 1
   Fully Processed - False

-- Chandra --
   Internal DAXA name - chandra
   Chosen instruments - ACIS-I, ACIS-S, HRC-I, HRC-S
   Number of observations - 15
   Fully Processed - False

-- Swift --
   Internal DAXA name - swift
   Chosen instruments - XRT, BAT
   Number of observations - 454
   Fully Processed - False
-----------------------------------------------------
```

3.2.4 Observations within the XXL-North region

This case study demonstrates how we can use a DAXA filtering method to identify all observations whose coordinate falls within a rectangular region - we apply this to the task of selecting all the XMM and Chandra observations within the XXL-North (XXL is a project utilising the largest contiguous region observed by XMM, one in the north and one in the south).

The filtering method we demonstrate here could just as easily be applied to other wide regions of interest (the Lockman hole for example), and will work with any DAXA mission, not just XMM and Chandra.

Import Statements

[1]:
```
from astropy.coordinates import SkyCoord
from astropy.units import hourangle

from daxa.mission import XMMPointed, Chandra
```

Other Tutorials

These case studies are meant to be highly specific examples of how you might acquire data for a particular science case, they do not provide general instruction on how to use DAXA missions or archives. We instead direct you to:

- Using DAXA missions - Here we explain what DAXA mission classes are and how to use them to select only the data you need.
- Creating a DAXA archive - This explains how to create an archive, load an existing archive, and the various properties and features of DAXA archives.
- Processing telescope data - The processing tutorials for different missions are presented here, though there may not yet be processing support for all missions.

Reading through these should give you a good understanding of how DAXA can be used to acquire, organise, and process multi-mission X-ray datasets for your specific use case.

Defining missions

The XXL project used XMM, so we knows there will be plenty of observations of the XXL region with XMM, but we will also search for any Chandra observations that fall within the same region:

[2]:
```
xm = XMMPointed()
ch = Chandra()
```

```
/Users/dt237/code/DAXA/daxa/mission/xmm.py:83: UserWarning: 140 of the 17697␣
↪observations located for this mission have been removed due to NaN RA or Dec values
  self._fetch_obs_info()
```

Searching for observations

We are going to use a DAXA filtering method that allows for the selection of all observations whose central coordinate falls within a rectangle - for that search we need to define the lower-left and upper-right coordinates of the rectangular region. We define these coordinates with convention that RA increases from right-to-left (i.e. the upper-right RA is greater than the lower-left RA), but the search method can also handle rectangles that have RA increasing from left-to-right.

The coordinates we've created broadly cover the XXL-North region:

[3]:
```
ll = SkyCoord("2h36m0s", -8., unit=(hourangle, 'deg'))
ur = SkyCoord("2h00m0s", -2., unit=(hourangle, 'deg'))
```

The `filter_on_rect_region` method is used, and we simply need to pass the corner coordinates we have already defined:

[4]:
```
xm.filter_on_rect_region(ll, ur)
ch.filter_on_rect_region(ll, ur)
```

```
/Users/dt237/code/DAXA/daxa/mission/base.py:107: UserWarning: The passed corner␣
↪coordinates are defined with RA increasing from right to left (upper-right RA is less␣
↪than lower-left; we reversed this.
  any_ret = change_func(*args, **kwargs)
```

Examining the available observations

As expected, we have selected many XMM observations - and there is also a not-insignificant number of Chandra observations.

[5]:
```
xm.filtered_obs_info
```

[5]:
```
              ra       dec       ObsID               start  science_usable  \
234    35.666670 -3.833333  0037980101 2002-01-11 04:21:02            True
235    36.000000 -3.833333  0037980201 2002-01-11 14:20:11            True
236    36.333330 -3.833333  0037980301 2002-01-11 23:37:59            True
237    36.666660 -3.833333  0037980401 2002-01-26 02:24:12            True
238    37.000005 -3.833333  0037980501 2002-01-25 20:51:08            True
...          ...       ...         ...                 ...             ...
14759  31.332917 -2.551611  0870850101 2020-07-04 21:25:41            True
14778  38.402083 -5.507694  0862670101 2020-07-10 03:03:35            True
14846  38.402083 -5.507722  0862670201 2020-08-16 15:57:07            True
14847  38.402083 -5.507722  0862670401 2020-08-16 13:39:53            True
17409  36.930375 -6.091694  0920220301 2024-01-25 15:58:15            True

             duration proprietary_end_date  revolution  proprietary_usable  \
234   0 days 04:12:52           2003-02-13         383                True
235   0 days 03:52:55           2003-02-13         383                True
236   0 days 03:54:31           2003-02-08         383                True
237   0 days 04:23:51           2003-02-08         390                True
238   0 days 04:53:35           2003-02-08         390                True
...               ...                  ...         ...                 ...
14759 0 days 10:06:40           2021-07-27        3767                True
14778 1 days 04:03:20           2021-09-17        3770                True
14846 1 days 00:03:19           2021-09-17        3789                True
14847 0 days 02:17:14           2021-09-17        3789                True
17409 0 days 06:23:20           2025-02-08        4419               False

                     end
234   2002-01-11 08:33:54
235   2002-01-11 18:13:06
236   2002-01-12 03:32:30
237   2002-01-26 06:48:03
238   2002-01-26 01:44:43
...                   ...
14759 2020-07-05 07:32:21
14778 2020-07-11 07:06:55
14846 2020-08-17 16:00:26
14847 2020-08-16 15:57:07
17409 2024-01-25 22:21:35

[422 rows x 10 columns]
```

[6]:
```
ch.filtered_obs_info
```

[6]:
```
             ra      dec  ObsID  science_usable  proprietary_usable  \
77     31.18436 -5.09273   4129            True                True
711    34.58875 -5.17417  12882            True                True
886    35.04167 -6.04167  14972            True                True
891    35.02417 -5.14100  13374            True                True
913    36.68333 -4.69583   9368            True                True
...         ...      ...    ...             ...                 ...
17733  33.34208 -6.09800  16574            True                True
17885  31.54917 -6.19300  16575            True                True
18900  37.90667 -7.48181   3030            True                True
19149  33.62208 -5.29569   4767            True                True
20831  35.27350 -4.68375  23741            True                True

                           start                       end        duration  \
77    2003-06-13 01:01:52.000003 2003-06-14 22:01:52.000003 1 days 21:00:00
711   2010-09-27 04:39:01.999996 2010-09-28 04:25:31.999996 0 days 23:46:30
886   2013-09-26 12:25:33.000004 2013-09-27 09:50:03.000004 0 days 21:24:30
891   2011-10-07 05:41:46.000003 2011-10-08 02:59:26.000003 0 days 21:17:40
913   2007-11-23 16:55:21.999996 2007-11-24 14:01:11.999996 0 days 21:05:50
...                          ...                       ...             ...
17733 2015-08-05 17:22:51.000001 2015-08-05 18:47:21.000001 0 days 01:24:30
17885 2015-06-24 05:19:51.999997 2015-06-24 06:44:01.999997 0 days 01:24:10
18900 2002-09-27 23:17:28.000003 2002-09-28 00:28:18.000003 0 days 01:10:50
19149 2003-11-26 16:45:20.999998 2003-11-26 17:53:30.999998 0 days 01:08:10
20831 2020-12-06 12:06:00.000003 2020-12-06 12:40:20.000003 0 days 00:34:20

      proprietary_end_date target_category instrument grating data_mode
77              2004-06-27             AGN     ACIS-I    NONE  TE_002AC
711             2011-09-29             GCL     ACIS-S    NONE  TE_0046E
886             2014-10-02             AGN     ACIS-S    NONE  TE_00AD8
891             2012-10-11             GCL     ACIS-I    NONE  TE_00AB4
913             2008-11-26             GCL     ACIS-S    NONE  TE_0046E
...                    ...             ...        ...     ...       ...
17733           2016-08-06             GCL     ACIS-S    NONE  TE_0046E
17885           2016-06-25             GCL     ACIS-S    NONE  TE_0046E
18900           2003-10-03             AGN     ACIS-S    NONE  TE_002A2
19149           2004-12-01             AGN     ACIS-S    NONE  TE_002A2
20831           2021-12-07             AGN     ACIS-S    NONE  TE_004A6

[84 rows x 13 columns]
```

3.3 Creating and interacting with a DAXA Archive

This tutorial will explain the basic concepts behind the second type of important class in DAXA, the Archive class (with mission classes being the first type, see the missions tutorial). DAXA Archives are what manage the datasets that we download from various missions, enabling easy access and greatly simplifying processing/reduction - they allow you to stop thinking about all the files and settings that any large dataset entails.

We will cover the following:

- Setting up an Archive from scratch, using filtered DAXA missions.
- Loading an existing Archive from disk.
- The properties of an Archive.
- Accessing processing logs and success information (though we do not cover processing in this part of the documentation).

3.3.1 Import Statements

[1]:
```
from daxa.mission import XMMPointed, Chandra, eRASS1DE, ROSATPointed
from daxa.archive import Archive

import os
```

3.3.2 What is a DAXA archive?

DAXA Archives take a set of filtered missions, make sure that their data are downloaded, and enable easy access and organisation of all data files and processing functions. Key functionality includes:

- Storing the logs and errors of all processing steps (if run).
- Allowing for their easy retrieval.
- Managing the myriad files produced during the processing.
- Keeping track of which processes failed for which data, ensuring that any further processing only runs on data that have successfully passed through the earlier processes.

Archives can also be loaded back into DAXA at a later date, so that the processing logs of data that has since been found to be problematic can be easily inspected, or indeed so that processing steps can be re-run with different settings; this also allows for archives to be updated, if more data become available.

3.3.3 Creating a new archive

Here we will demonstrate how to set up a new DAXA Archive from scratch - this information can be combined with the the missions tutorial and the case studies to create an archive from any dataset you might be using.

Step 1 - Set up and filter missions

The first thing we have to do is to select the observations that we wish to include in the archive (and indeed the missions that we wish to include). The missions all have different characteristics, so your choice of which to include will be heavily dependent on your science case.

Here we will create an archive of XMM, Chandra, eROSITA All-Sky DR1, and ROSAT pointed observations of a famous galaxy cluster (though the archive would behave the same if it held data for a large sample of objects).

First of all, we define instances of the mission classes that we wish to include:

[2]:
```
xm = XMMPointed()
ch = Chandra()
er = eRASS1DE()
rp = ROSATPointed()
```

```
/Users/dt237/code/DAXA/daxa/mission/xmm.py:83: UserWarning: 140 of the 17701␣
↪observations located for this mission have been removed due to NaN RA or Dec values
  self._fetch_obs_info()
```

Then we filter them to only include observations of our cluster:

[3]:
```
xm.filter_on_name("A3667")
ch.filter_on_name("A3667")
er.filter_on_name("A3667")
rp.filter_on_name("A3667")
```

```
/Users/dt237/code/DAXA/daxa/mission/base.py:1335: UserWarning: Chandra FoV are difficult␣
↪to define, as they can be strongly dependant on observation mode; as such take these␣
↪as very approximate.
  fov = self.fov
```

We then download the available data (though the declaration of an Archive would also trigger this, we do it this way because we wish to download pre-generated products for Chandra and ROSAT pointed observations):

[4]:
```
xm.download()
ch.download(download_products=True)
er.download()
rp.download(download_products=True)
```

```
Downloading XMM-Newton Pointed data: 100%|| 8/8 [02:18<00:00, 17.33s/it]
Downloading Chandra data: 100%|| 12/12 [02:58<00:00, 14.86s/it]
Downloading eRASS DE:1 data: 100%|| 1/1 [01:18<00:00, 78.21s/it]
Downloading ROSAT Pointed data: 100%|| 3/3 [00:17<00:00,  5.77s/it]
```

Step 2 - Setting up an Archive object

Now we create the actual DAXA Archive instance - we can pass the following arguments:

- `archive_name` - The name to be given to the archive (used to load it back in at a later date, if necessary). The only input required
- `missions` - The filtered missions that we have already created (can be left as None if loading in an existing archive).
- `clobber` - Will overwrite an existing archive if the passed `archive_name` has already been used. Default is False.

- `download_products` - If the missions have not already had downloads triggered, this controls whether pre-processed products should be downloaded or not. Default is True, a dictionary with mission names as keys and True/False as values can be passed to provide more nuanced control.
- `use_preprocessed` - Whether pre-processed data (for those missions that provide it) should be automatically imported into the archive processed data structure. Default is False, a dictionary with mission names as keys and True/False as values can be passed to provide more nuanced control.

We demonstrate how to set up an archive using the pre-processed data that is downloadable from the Chandra, eRASS1DE, and ROSAT-Pointed online datasets, as well as the raw data available from the XMM online dataset:

[5]:
```
arch = Archive("A3667", [xm, ch, er, rp], clobber=True,
               use_preprocessed={'xmm_pointed': False, 'chandra': True,
                                 'erosita_all_sky_de_dr1': True,
                                 'rosat_pointed': True})
```

```
Including pre-processed Chandra data in the archive: 100%|| 12/12 [00:00<00:00, 24.73it/
↪s]
Including pre-processed eRASS DE:1 data in the archive: 100%|| 1/1 [00:00<00:00,  1.14it/
↪s]
Including pre-processed ROSAT Pointed data in the archive: 100%|| 3/3 [00:00<00:00, 16.
↪83it/s]
```

Now we've declared it, we can use the `info()` method to get a summary of its current status, including the amount of data available:

[6]:
```
arch.info()
```

```
--------------------------------------------------
Number of missions - 4
Total number of observations - 24
Beginning of earliest observation - 1992-04-14 18:55:38.000003
End of latest observation - 2020-04-20 12:23:50

-- XMM-Newton Pointed --
   Internal DAXA name - xmm_pointed
   Chosen instruments - M1, M2, PN
   Number of observations - 8
   Fully Processed - False

-- Chandra --
   Internal DAXA name - chandra
   Chosen instruments - ACIS-I, ACIS-S, HRC-I, HRC-S
   Number of observations - 12
   Fully Processed - True

-- eRASS DE:1 --
   Internal DAXA name - erosita_all_sky_de_dr1
   Chosen instruments - TM1, TM2, TM3, TM4, TM5, TM6, TM7
   Number of observations - 1
   Fully Processed - True

-- ROSAT Pointed --
   Internal DAXA name - rosat_pointed
   Chosen instruments - PSPCB, PSPCC, HRI
```

(continues on next page)

(continued from previous page)

```
   Number of observations - 3
   Fully Processed - True
------------------------------------------------------
```

Step 3 - Processing the Archive

We're not actually going to cover *how* to process things here, as each telescope tends to have its own backend software with a unique way of doing things; they each have their own processing tutorials, which will demonstrate both a one-line processing method, and how to control the reduction in more detail. Any processing method will take the archive object as an argument, and act on the data stored within it.

So instead we include this step here to highlight that the next logical step after the creation of a new archive is to run processing and reduction routines, if raw data have been downloaded. The successful completion of this step will leave you with an archive of data that you can easily manage, access, and use for your scientific analyses.

If you elected to download existing products (most missions support this), then only one processing step is necessary - this reorganises the downloaded data so that it is compatible with DAXA storage and file naming conventions. **It will have run automatically on declaration**

Step 4 - Using the data

Once an archive has been constructed, whether of raw data to be processed locally, pre-processed data products, or a combination of both, the end result will be a set of X-ray data that can be used for scientific analysis.

The storage and organisation of the archive's data is entirely consistent between different telescopes; the archive's processed data (the 'processed_data' directory within the overall archive path, which is identified using the `archive_path` property) is organised like this:

- **Mission Name** - For instance, 'xmm_pointed' or 'chandra'
  - *ObsID* - Each ObsID of a mission gets a sub-directory.
    - events - Where event lists (uncleaned, cleaned, and final) are stored.
    - images - Where all images and exposure maps are stored.
    - background - Where any background maps or models are stored.
    - cleaning - Where any by-products of the cleaning processes are stored.
    - logs - Where logs from all processing steps are stored.

To provide an example, we show you the contents of an eRASS1DE directory that contains pre-processed data:

[7]:
```
demo_pth = arch.get_current_data_path('erosita_all_sky_de_dr1', '304147')

os.listdir(demo_pth)
```

[7]:
```
['background', 'images', 'logs', 'events', 'cleaning']
```

Now the contents of the images sub-directory:

[8]:
```
os.listdir(os.path.join(demo_pth, 'images'))
```

[8]:
```
['obsid304147-instTM1_TM2_TM3_TM4_TM5_TM6_TM7-subexpNone-en0.2_0.5keV-image.fits',
 'obsid304147-instTM1_TM2_TM3_TM4_TM5_TM6_TM7-subexpNone-en0.2_2.3keV-image.fits',
 'obsid304147-instTM1_TM2_TM3_TM4_TM5_TM6_TM7-subexpNone-en0.6_2.3keV-image.fits',
 'obsid304147-instTM1_TM2_TM3_TM4_TM5_TM6_TM7-subexpNone-en2.3_5.0keV-image.fits',
 'obsid304147-instTM1_TM2_TM3_TM4_TM5_TM6_TM7-subexpNone-en0.2_0.6keV-image.fits',
 'obsid304147-instTM1_TM2_TM3_TM4_TM5_TM6_TM7-subexpNone-en0.5_1.0keV-expmap.fits',
 'obsid304147-instTM1_TM2_TM3_TM4_TM5_TM6_TM7-subexpNone-en0.2_2.3keV-expmap.fits',
 'obsid304147-instTM1_TM2_TM3_TM4_TM5_TM6_TM7-subexpNone-en1.0_2.0keV-image.fits',
 'obsid304147-instTM1_TM2_TM3_TM4_TM5_TM6_TM7-subexpNone-en1.0_2.0keV-expmap.fits',
 'obsid304147-instTM1_TM2_TM3_TM4_TM5_TM6_TM7-subexpNone-en2.3_5.0keV-expmap.fits',
 'obsid304147-instTM1_TM2_TM3_TM4_TM5_TM6_TM7-subexpNone-en0.2_0.6keV-expmap.fits',
 'obsid304147-instTM1_TM2_TM3_TM4_TM5_TM6_TM7-subexpNone-en0.5_1.0keV-image.fits',
 'obsid304147-instTM1_TM2_TM3_TM4_TM5_TM6_TM7-subexpNone-en0.6_2.3keV-expmap.fits',
 'obsid304147-instTM1_TM2_TM3_TM4_TM5_TM6_TM7-subexpNone-en0.2_0.5keV-expmap.fits']
```

As well as the events sub-directory:

[9]:
```
os.listdir(os.path.join(demo_pth, 'events'))
```

[9]:
```
['obsid304147-instTM1_TM2_TM3_TM4_TM5_TM6_TM7-subexpNone-finalevents.fits']
```

The common format of the filenames used by DAXA should be fairly evident by this point; it extends across all data products produced by DAXA and is shared across all the different supported missions. Some elements may differ between data products, but the overall structure is the same.

Each name contains summary information that allow it to be uniquely identified, both programatically and by a user; different pieces of information start with an identifier (for instance obsid to denote where the relevant ObsID begins), and different pieces are separated by a '-'. If there are multiple bits of information connected to a particular category then they are separated by '_' (see 'inst' part of eROSITA file names above).

The following can be contained in a filename:

- **ObsID** - Denoted by 'obsid', this will be present in the name of every data file.
- **Instrument** - Denoted by 'inst', this should be present in the name of every data file.
- **Sub-exposure ID** - Denoted by 'subexp'; some missions support multiple sub-exposures per ObsID instrument, in which case the sub-exposure ID will be included here. It is also possible for 'None' to be the value if there are no sub-exposures, or 'ALL' if the file is a combination of sub-exposures.
- **Lower and Upper Energy** - Denoted by 'en', this will be of the form en{lower}_{upper}{unit} (en0.2_10.0keV for instance). This is for cases where the product is energy bound. It is also possible for the value to be 'None' if the 'en' identifier is included in the filename but no energy limits were applied.
- **The type of product** - This will always be at the end of the filename, and indicates what the file contains. For instance 'finalevents' would represent the final event file for use in an analysis, 'image' would be an image, etc.

Note on saving Archives

Archive instances can be saved and loaded back in (as you'll see in the next section). This can be triggered manually by running the `save()` method, but **this shouldn't be generally necessary** - this is because the archive is automatically saved upon first setup, and after every processing step.

3.3.4 Loading an existing archive

As we have intimated, previously created archives can be loaded back in to memory in exactly the same state as when they were saved. We will demonstrate this here with an archive we prepared earlier - it has had XMM processing applied, which will allow us to demonstrate the logging and management functionality.

Reloading an archive has a number of possible applications:

- Access to archive data management functions - e.g. locating specific data files, identifying what observations are available.
- Checking processing logs - e.g. finding errors or warnings in the processing of data that has since been identified as problematic.
- Updating the archive - either adding another mission, or using the archive to check for new data matching your original mission filtering operations (these are stored in the mission saves, so can be re-run automatically).

All you need to do is set up an Archive instance and pass the name of an existing archive - this assumes your code is running in the same directory as it was originally, as Archives are stored in 'daxa_output' (if the DAXA configuration file hasn't been altered). The configuration can also be altered so that all DAXA outputs are stored in an absolute path, in which case defining an Archive object with the name of an existing dataset would work from any directory).

Loading in an archive (note that you don't need to pass any missions, loading the archive back in will also reinstate the missions as they were when the Archive was last saved):

[10]:
```
prev_arch = Archive("PHL1811_made_earlier")
```

```
/Users/dt237/code/DAXA/daxa/mission/xmm.py:83: UserWarning: 140 of the 17701␣
↪observations located for this mission have been removed due to NaN RA or Dec values
  self._fetch_obs_info()
```

Once again, we will run the `info()` method, but note that for this archive the XMM-Newton Pointed mission is marked as 'fully processed':

[11]:
```
prev_arch.info()
```

```
------------------------------------------------------
Number of missions - 4
Total number of observations - 10
Beginning of earliest observation - 1990-10-31 00:00:00
End of latest observation - 2015-11-30 04:11:08.184002

-- XMM-Newton Pointed --
   Internal DAXA name - xmm_pointed
   Chosen instruments - M1, M2, PN
   Number of observations - 5
   Fully Processed - False

-- Chandra --
   Internal DAXA name - chandra
   Chosen instruments - ACIS-I, ACIS-S, HRC-I, HRC-S
   Number of observations - 3
   Fully Processed - False

-- NuSTAR Pointed --
   Internal DAXA name - nustar_pointed
```

(continues on next page)

(continued from previous page)

```
   Chosen instruments - FPMA, FPMB
   Number of observations - 1
   Fully Processed - False

-- RASS --
   Internal DAXA name - rosat_all_sky
   Chosen instruments - PSPC
   Number of observations - 1
   Fully Processed - False
-------------------------------------------------------
```

We note that it *is* possible to declare an Archive with a previously used name and overwrite it - you just have to pass `clobber=True` when you declare the Archive instance. We print the docstring of the Archive class here for reference:

[12]:
```
print(Archive.__doc__)
```

```
    The Archive class, which is to be used to consolidate and provide some interface␣
↪with a set
    of mission's data. Archives can be passed to processing and cleaning functions in␣
↪DAXA, and also
    contain convenience functions for accessing summaries of the available data.

    :param str archive_name: The name to be given to this archive - it will be used for␣
↪storage
        and identification. If an existing archive with this name exists it will be read␣
↪in, unless clobber=True.
    :param List[BaseMission]/BaseMission missions: The mission, or missions, which are␣
↪to be included
        in this archive - any setup processes (i.e. the filtering of data to be␣
↪acquired) should be
        performed prior to creating an archive. The default value is None, but this␣
↪should be set for any new
        archives, it can only be left as None if an existing archive is being read back␣
↪in.
    :param bool clobber: If an archive named 'archive_name' already exists, then setting␣
↪clobber to True
        will cause it to be deleted and overwritten.
    :param bool/dict download_products: Controls whether pre-processed products should␣
↪be downloaded for missions
        that offer it (assuming downloading was not triggered when the missions were␣
↪declared). Default is
        True, but False may also be passed, as may a dictionary of DAXA mission names␣
↪with True/False values.
    :param bool/dict use_preprocessed: Whether pre-processed data products should be␣
↪used rather than re-processing
        locally with DAXA. If True then what pre-processed data products are available␣
↪will be automatically
        re-organised into the DAXA processed data structure during the setup of this␣
↪archive. If False (the default)
        then this will not automatically be applied. Just as with 'download_products', a␣
↪dictionary may be passed for
```

(continues on next page)

(continued from previous page)

```
        more nuanced control, with mission names as keys and True/False as values.
```

3.3.5 Accessing component missions

The missions that were used to create an archive can be retrieved, giving access to their information tables - note that you cannot just use the filtering methods of a mission to change the data in the archive; altering the observations in an archive requires using the archive `update()` method.

To retrieve a mission you can either address the archive with the DAXA internal name of the mission, or get the whole list using the `missions` property:

[13]:
```
prev_arch['xmm_pointed']
```

[13]:
```
<daxa.mission.xmm.XMMPointed at 0x7f77aafeedf0>
```

[14]:
```
prev_arch.missions
```

[14]:
```
[<daxa.mission.xmm.XMMPointed at 0x7f77aafeedf0>,
 <daxa.mission.chandra.Chandra at 0x7f77aad6f220>,
 <daxa.mission.nustar.NuSTARPointed at 0x7f7778575fd0>,
 <daxa.mission.rosat.ROSATAllSky at 0x7f77985d8790>]
```

[15]:
```
prev_arch['xmm_pointed'].filtered_obs_info
```

[15]:
```
               ra        dec       ObsID               start  science_usable  \
922    157.746300  31.048890  0102041001 2000-12-07 04:57:14            True
1044    67.555425 -61.350560  0105261001 2000-09-27 06:56:36            True
3802   328.756200  -9.373528  0204310101 2004-11-01 09:06:42            True
6014   234.896250 -83.593060  0502671101 2008-04-01 17:24:48            True
12006  328.756250  -9.373333  0761910201 2015-11-29 09:38:07            True

             duration proprietary_end_date  revolution  proprietary_usable  \
922   0 days 01:30:06  2002-04-06 00:00:00         182                True
1044  0 days 04:16:54  2002-08-25 00:00:00         147                True
3802  0 days 09:08:39  2005-12-01 00:00:00         897                True
6014  0 days 05:25:20  2009-05-29 00:00:00        1522                True
12006 0 days 16:30:00  2016-12-11 23:00:00        2925                True

                      end
922   2000-12-07 06:27:20
1044  2000-09-27 11:13:30
3802  2004-11-01 18:15:21
6014  2008-04-01 22:50:08
12006 2015-11-30 02:08:07
```

3.3.6 Archive general properties

Here we run through the general properties of the archive class, summarising their meaning and content.

Name

The `archive_name` class returns the name that was given to the archive on creation - this cannot be changed.

[16]:
```
prev_arch.archive_name
```

[16]:
```
'PHL1811_made_earlier'
```

Archive Path

This property (`archive_path`) returns the absolute path to the top level of this archive's storage directory:

[17]:
```
prev_arch.archive_path
```

[17]:
```
'/Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_output/archives/PHL1811_
↪made_earlier/'
```

Mission Names

In addition to the `missions` property discussed earlier, we include a `mission_names` property which lists the internal names of the mission classes associated with the archive:

[18]:
```
prev_arch.mission_names
```

[18]:
```
['xmm_pointed', 'chandra', 'nustar_pointed', 'rosat_all_sky']
```

3.3.7 Processing-related Archive properties

This section deals with Archive properties that are related to processing of the available data into something scientifically useful - this is why we loaded an existing archive with processing applied, to demonstrate the contents of these properties:

Process Success

The `process_success` property is very important - it is a nested dictionary which records which processing steps were 'successful' (usually defined as no errors being detected, and expected files being found) for which data. Those that were successful have a boolean value of True, those that weren't have a boolean value of False.

Top level keys will always be mission name, the next level down will be the process name, and the layer below that will be the unique IDs of the data the process acted on. This is often an ObsID, but can also be ObsID + instrument name, or ObsID + instrument name + sub-exposure ID.

This allows you (but more importantly the archive itself) to know which stages have failed for which data - that in turn means any processing that is dependent on a previous stage can know which data to skip. All this ensures no interruptions when reducing large sets of data.

In this case we've run all processing steps on the XMM data in the archive, note the following entry:

- `espfilt` - 0502671101PNS003

It failed safely, and was not considered for the next processing stages. **Also note that the ObsID 0102041001 does not appear** after the `odf_ingest` step, as all of its data was taken in CalClosed mode and can't be used for the study of the target objects:

[19]:
```
prev_arch.process_success
```

[19]:
```
{'xmm_pointed': {'cif_build': {'0502671101': True,
   '0105261001': True,
   '0102041001': True,
   '0761910201': True,
   '0204310101': True},
  'odf_ingest': {'0102041001': True,
   '0105261001': True,
   '0502671101': True,
   '0204310101': True,
   '0761910201': True},
  'epchain': {'0105261001PNS003': True,
   '0105261001PNU002': True,
   '0502671101PNS003': True,
   '0204310101PNS003': True,
   '0761910201PNS003': True},
  'emchain': {'0105261001M2U002': True,
   '0105261001M1S001': True,
   '0105261001M1U002': True,
   '0105261001M2S002': True,
   '0502671101M1S001': True,
   '0502671101M2S002': True,
   '0204310101M1S001': True,
   '0204310101M2S002': True,
   '0761910201M1S001': True,
   '0761910201M2S002': True},
  'espfilt': {'0105261001M1S001': True,
   '0105261001M2S002': True,
   '0105261001M2U002': True,
   '0105261001M1U002': True,
   '0204310101M2S002': True,
   '0204310101M1S001': True,
   '0761910201M1S001': True,
   '0502671101M1S001': True,
   '0502671101M2S002': True,
   '0761910201M2S002': True,
   '0502671101PNS003': False,
   '0105261001PNS003': True,
   '0105261001PNU002': True,
   '0204310101PNS003': True,
   '0761910201PNS003': True},
  'cleaned_evt_lists': {'0105261001M1S001': True,
   '0105261001M2S002': True,
   '0105261001M1U002': True,
   '0105261001M2U002': True,
   '0761910201M1S001': True,
   '0204310101M1S001': True,
   '0204310101M2S002': True,
   '0502671101M1S001': True,
```

(continues on next page)

(continued from previous page)

```
   '0502671101M2S002': True,
   '0761910201M2S002': True,
   '0105261001PNS003': True,
   '0105261001PNU002': True,
   '0204310101PNS003': True,
   '0761910201PNS003': True},
  'merge_subexposures': {'0502671101M2': True,
   '0502671101M1': True,
   '0204310101M1': True,
   '0204310101M2': True,
   '0761910201M2': True,
   '0761910201M1': True,
   '0204310101PN': True,
   '0761910201PN': True,
   '0105261001M2': True,
   '0105261001M1': True,
   '0105261001PN': True}},
 'chandra': {'preprocessed_events': {'2958ACIS-S': True,
   '2957ACIS-S': True,
   '15357ACIS-S': True},
  'preprocessed_images': {'2958ACIS-S': True,
   '2957ACIS-S': True,
   '15357ACIS-S': True}},
 'nustar_pointed': {'preprocessed_events': {'60101004002FPMA': True,
   '60101004002FPMB': True},
  'preprocessed_images': {'60101004002FPMA': True, '60101004002FPMB': True}},
 'rosat_all_sky': {'preprocessed_events': {'RS931959N00PSPC': True},
  'preprocessed_images': {'RS931959N00PSPC': True},
  'preprocessed_expmaps': {'RS931959N00PSPC': True},
  'preprocessed_backmaps': {'RS931959N00PSPC': True}}}
```

Process Names

This property (`process_names`) stores a list of the processes that have been run on each mission:

[20]:
```
prev_arch.process_names
```

[20]:
```
{'xmm_pointed': ['cif_build',
  'odf_ingest',
  'epchain',
  'emchain',
  'espfilt',
  'cleaned_evt_lists',
  'merge_subexposures'],
 'chandra': ['preprocessed_events', 'preprocessed_images'],
 'nustar_pointed': ['preprocessed_events', 'preprocessed_images'],
 'rosat_all_sky': ['preprocessed_events',
  'preprocessed_images',
  'preprocessed_expmaps',
  'preprocessed_backmaps']}
```

Process Logs (stdout)

All of the logs for all processes run by DAXA are stored, and can be accessed through the `process_logs` property - this is structured in the exact same way as `process_success`, as a nested dictionary. The only difference here is that the final values are strings rather than booleans.

We show the log for a single process applied to a single piece of data, otherwise this tutorial document would be very long indeed - this particular process worked perfectly:

[21]:
```
print(prev_arch.process_logs['xmm_pointed']['espfilt']['0204310101M1S001'])
```

```
espfilt:- Executing (routine): espfilt eventfile=/Users/dt237/code/DAXA/docs/source/
↪notebooks/tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_
↪pointed/0204310101/events/obsid0204310101-instM1-subexpS001-events.fits withoot=no␣
↪ootfile=dataset method=histogram withsmoothing=yes smooth=51 withbinning=yes␣
↪binsize=60 ratio=1.2 withlongnames=yes elow=2500 ehigh=8500 rangescale=6 allowsigma=3␣
↪limits='0.1 6.5' keepinterfiles=no  -w 1 -V 4
espfilt:- espfilt (espfilt-4.3)  [xmmsas_20211130_0941-20.0.0] started:  2024-04-24T19:
↪20:01.000
espfilt:-  ESPFILT: Processing eventlist: /Users/dt237/code/DAXA/docs/source/notebooks/
↪tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/
↪0204310101/events/obsid0204310101-instM1-subexpS001-events.fits
espfilt:-  *FOV IMAGE* = mos1S001-fovim-2500-8500.fits
evselect:- Executing (routine): evselect table=/Users/dt237/code/DAXA/docs/source/
↪notebooks/tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_
↪pointed/0204310101/events/obsid0204310101-instM1-subexpS001-events.fits␣
↪filteredset=filtered.fits withfilteredset=no keepfilteroutput=no flagcolumn=EVFLAG␣
↪flagbit=-1 destruct=yes dssblock='' expression=true filtertype=expression cleandss=no␣
↪updateexposure=yes filterexposure=yes writedss=yes blockstocopy='' attributestocopy=''␣
↪energycolumn=PHA zcolumn=WEIGHT zerrorcolumn=EWEIGHT withzerrorcolumn=no␣
↪withzcolumn=no ignorelegallimits=no imageset=mos1S001-fovim-2500-8500.fits␣
↪xcolumn=DETX ycolumn=DETY ximagebinsize=1 yimagebinsize=1 squarepixels=no␣
↪ximagesize=600 yimagesize=600 imagebinning=imageSize ximagemin=1 ximagemax=640␣
↪withxranges=no yimagemin=1 yimagemax=640 withyranges=no imagedatatype=Real64␣
↪withimagedatatype=no raimagecenter=0 decimagecenter=0 withcelestialcenter=no␣
↪withimageset=yes spectrumset=spectrum.fits spectralbinsize=5 specchannelmin=0␣
↪specchannelmax=11999 withspecranges=no nonStandardSpec=no withspectrumset=no␣
↪rateset=rate.fits timecolumn=TIME timebinsize=1 timemin=0 timemax=1000␣
↪withtimeranges=no maketimecolumn=no makeratecolumn=no withrateset=no␣
↪histogramset=histo.fits histogramcolumn=TIME histogrambinsize=1 histogrammin=0␣
↪histogrammax=1000 withhistoranges=no withhistogramset=no  -w 1 -V 4
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] started:  2024-04-
↪24T19:20:01.000
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] ended:    2024-04-
↪24T19:20:01.000
espfilt:-  Running evselect to create *FOV LIGHTCURVE*
espfilt:-  *FOV LIGHTCURVE * = mos1S001-fovlc-2500-8500.fits
evselect:- Executing (routine): evselect table=/Users/dt237/code/DAXA/docs/source/
↪notebooks/tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_
↪pointed/0204310101/events/obsid0204310101-instM1-subexpS001-events.fits:EVENTS␣
↪filteredset=filtered.fits withfilteredset=yes keepfilteroutput=no flagcolumn=EVFLAG␣
↪flagbit=-1 destruct=yes dssblock='' expression='(PATTERN<=12)&&(PI in [2500:8500])&&(
↪#XMMEA_EM)' filtertype=expression cleandss=no updateexposure=yes filterexposure=yes␣
↪writedss=yes blockstocopy='' attributestocopy='' energycolumn=PHA zcolumn=WEIGHT␣
↪zerrorcolumn=EWEIGHT withzerrorcolumn=no withzcolumn=no ignorelegallimits=no␣
↪imageset=image.fits xcolumn=RAWX ycolumn=RAWY ximagebinsize=1 yimagebinsize=1␣
↪squarepixels=no ximagesize=600 yimagesize=600 imagebinning=imageSize ximagemin=1␣
↪ximagemax=640 withxranges=no yimagemin=1 yimagemax=640 withyranges=no␣
↪imagedatatype=Real64 withimagedatatype=no raimagecenter=0 decimagecenter=0␣
↪withcelestialcenter=no withimageset=no spectrumset=spectrum.fits spectralbinsize=5␣
↪specchannelmin=0 specchannelmax=11999 withspecranges=no nonStandardSpec=no␣
```

(continues on next page)

(continued from previous page)

```
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] started:  2024-04-
↪24T19:20:02.000
evselect:- selected 31988 rows from the input table.
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] ended:    2024-04-
↪24T19:20:02.000
espfilt:-  Running evselect to create *CORNER EVENTLIST*
espfilt:-  *CORNER EVENTLIST * = mos1S001-corev-2500-8500.fits
evselect:- Executing (routine): evselect table=/Users/dt237/code/DAXA/docs/source/
↪notebooks/tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_
↪pointed/0204310101/events/obsid0204310101-instM1-subexpS001-events.fits:EVENTS␣
↪filteredset=mos1S001-corev-2500-8500.fits withfilteredset=yes keepfilteroutput=yes␣
↪flagcolumn=EVFLAG flagbit=-1 destruct=yes dssblock='' expression='(PATTERN<=12)&&
↪((FLAG & 0x766aa000)==0)&&!(CIRCLE(100,-200,17700,DETX,DETY)||CIRCLE(834,135,17100,
↪DETX,DETY)||CIRCLE(770,-803,17100,DETX,DETY)||BOX(-20,-17000,6500,500,0,DETX,
↪DETY)||BOX(5880,-20500,7500,1500,10,DETX,DETY)||BOX(-5920,-20500,7500,1500,350,DETX,
↪DETY)||BOX(-20,-20000,5500,500,0,DETX,DETY)||BOX(-12900,16000,250,4000,0,DETX,
↪DETY)||BOX(80,18600,150,1300,0,DETX,DETY)||BOX(-10,-18800,125,1500,0,DETX,DETY))'␣
↪filtertype=expression cleandss=no updateexposure=yes filterexposure=yes writedss=yes␣
↪blockstocopy='' attributestocopy='' energycolumn=PHA zcolumn=WEIGHT␣
↪zerrorcolumn=EWEIGHT withzerrorcolumn=no withzcolumn=no ignorelegallimits=no␣
↪imageset=image.fits xcolumn=RAWX ycolumn=RAWY ximagebinsize=1 yimagebinsize=1␣
↪squarepixels=no ximagesize=600 yimagesize=600 imagebinning=imageSize ximagemin=1␣
↪ximagemax=640 withxranges=no yimagemin=1 yimagemax=640 withyranges=no␣
↪imagedatatype=Real64 withimagedatatype=no raimagecenter=0 decimagecenter=0␣
↪withcelestialcenter=no withimageset=no spectrumset=spectrum.fits spectralbinsize=5␣
↪specchannelmin=0 specchannelmax=11999 withspecranges=no nonStandardSpec=no␣
↪withspectrumset=no rateset=rate.fits timecolumn=TIME timebinsize=1 timemin=0␣
↪timemax=1000 withtimeranges=no maketimecolumn=no makeratecolumn=no withrateset=no␣
↪histogramset=histo.fits histogramcolumn=TIME histogrambinsize=1 histogrammin=0␣
↪histogrammax=1000 withhistoranges=no withhistogramset=no  -w 1 -V 4
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] started:  2024-04-
↪24T19:20:02.000
evselect:- selected 14626 rows from the input table.
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] ended:    2024-04-
↪24T19:20:03.000
espfilt:-  Running evselect to create *CORNER IMAGE*
espfilt:-  *CORNER IMAGE * = mos1S001-corim-2500-8500.fits
evselect:- Executing (routine): evselect table=mos1S001-corev-2500-8500.fits␣
↪filteredset=filtered.fits withfilteredset=no keepfilteroutput=no flagcolumn=EVFLAG␣
↪flagbit=-1 destruct=yes dssblock='' expression=true filtertype=expression cleandss=no␣
↪updateexposure=yes filterexposure=yes writedss=yes blockstocopy='' attributestocopy=''␣
↪energycolumn=PHA zcolumn=WEIGHT zerrorcolumn=EWEIGHT withzerrorcolumn=no␣
↪withzcolumn=no ignorelegallimits=no imageset=mos1S001-corim-2500-8500.fits␣
↪xcolumn=DETX ycolumn=DETY ximagebinsize=1 yimagebinsize=1 squarepixels=no␣
↪ximagesize=600 yimagesize=600 imagebinning=imageSize ximagemin=1 ximagemax=640␣
↪withxranges=no yimagemin=1 yimagemax=640 withyranges=no imagedatatype=Real64␣
↪withimagedatatype=no raimagecenter=0 decimagecenter=0 withcelestialcenter=no␣
↪withimageset=yes spectrumset=spectrum.fits spectralbinsize=5 specchannelmin=0␣
↪specchannelmax=11999 withspecranges=no nonStandardSpec=no withspectrumset=no␣
↪rateset=rate.fits timecolumn=TIME timebinsize=1 timemin=0 timemax=1000␣
↪withtimeranges=no maketimecolumn=no makeratecolumn=no withrateset=no␣
↪histogramset=histo.fits histogramcolumn=TIME histogrambinsize=1 histogrammin=0␣
↪histogrammax=1000 withhistoranges=no withhistogramset=no  -w 1 -V 4
```

(continues on next page)

(continued from previous page)

```
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] started:  2024-04-
↪24T19:20:03.000
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] ended:    2024-04-
↪24T19:20:03.000
espfilt:-  Running evselect to create *CORNER LIGHTCURVE*
espfilt:-  *CORNER LIGHTCURVE * = mos1S001-corlc-2500-8500.fits
evselect:- Executing (routine): evselect table=/Users/dt237/code/DAXA/docs/source/
↪notebooks/tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_
↪pointed/0204310101/events/obsid0204310101-instM1-subexpS001-events.fits:EVENTS␣
↪filteredset=filtered.fits withfilteredset=yes keepfilteroutput=no flagcolumn=EVFLAG␣
↪flagbit=-1 destruct=yes dssblock='' expression='(PATTERN<=12)&&(PI in [2500:8500])&&
↪((FLAG & 0x766aa000)==0)&&!(CIRCLE(100,-200,17700,DETX,DETY)||CIRCLE(834,135,17100,
↪DETX,DETY)||CIRCLE(770,-803,17100,DETX,DETY)||BOX(-20,-17000,6500,500,0,DETX,
↪DETY)||BOX(5880,-20500,7500,1500,10,DETX,DETY)||BOX(-5920,-20500,7500,1500,350,DETX,
↪DETY)||BOX(-20,-20000,5500,500,0,DETX,DETY)||BOX(-12900,16000,250,4000,0,DETX,
↪DETY)||BOX(80,18600,150,1300,0,DETX,DETY)||BOX(-10,-18800,125,1500,0,DETX,DETY))'␣
↪filtertype=expression cleandss=no updateexposure=yes filterexposure=yes writedss=yes␣
↪blockstocopy='' attributestocopy='' energycolumn=PHA zcolumn=WEIGHT␣
↪zerrorcolumn=EWEIGHT withzerrorcolumn=no withzcolumn=no ignorelegallimits=no␣
↪imageset=image.fits xcolumn=RAWX ycolumn=RAWY ximagebinsize=1 yimagebinsize=1␣
↪squarepixels=no ximagesize=600 yimagesize=600 imagebinning=imageSize ximagemin=1␣
↪ximagemax=640 withxranges=no yimagemin=1 yimagemax=640 withyranges=no␣
↪imagedatatype=Real64 withimagedatatype=no raimagecenter=0 decimagecenter=0␣
↪withcelestialcenter=no withimageset=no spectrumset=spectrum.fits spectralbinsize=5␣
↪specchannelmin=0 specchannelmax=11999 withspecranges=no nonStandardSpec=no␣
↪withspectrumset=no rateset=mos1S001-corlc-2500-8500.fits timecolumn=TIME timebinsize=1␣
↪timemin=0 timemax=1000 withtimeranges=no maketimecolumn=yes makeratecolumn=yes␣
↪withrateset=yes histogramset=histo.fits histogramcolumn=TIME histogrambinsize=1␣
↪histogrammin=0 histogrammax=1000 withhistoranges=no withhistogramset=no  -w 1 -V 4
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] started:  2024-04-
↪24T19:20:04.000
evselect:- selected 4962 rows from the input table.
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] ended:    2024-04-
↪24T19:20:04.000
espfilt:-  ESPFILT: Processing using the HISTOGRAM method.
espfilt:-  CLEAN_LC: There are        32670  rows in the FOV LC
espfilt:-  Starting Multidimensional minimzation of fit_gauss (amoeba)
espfilt:- plim(1:2) =   0.1000   6.5000
espfilt:- **** sortarray max bins    =   663 histo binwidth =    0.0196
espfilt:- **** sortarray actual bins (nverls) =   658
espfilt:- min LC value=    0.000 max LC value =    3.373 limits(2) =    6.500
espfilt:- ***** pre-amoeba: low =   30 peak =    35 high =    42 delta =    13
espfilt:- ***** pre-amoeba: cnts(low) = 1162 cnts(peak) =  1304 cnts(high) =   865
espfilt:-  pre-amoeba parms =  1304.000    0.767    0.100
espfilt:-
espfilt:- 1st rl =  1257.566    0.754    0.170 / rh =  1257.566    0.754    0.170
espfilt:- 1st cl =  2470.722 / ch =  2470.722
espfilt:- ***** 2nd amoeba: low =   29 peak =    34 high =    41
espfilt:- ***** 2nd amoeba: cnts(low) = 1013 cnts(peak)=  1247 cnts(high) =   903
espfilt:-  2nd parms =  1247.000    0.747    0.100
espfilt:- 2nd rl =  1261.334    0.755    0.164 / rh =  1261.334    0.755    0.164
espfilt:- 2nd cl =  2463.076 / ch =  2463.076
```

(continues on next page)

(continued from previous page)

```
espfilt:- ***** 3rd amoeba: low =   29 peak =    34 high =    41
espfilt:- ***** 3rd amoeba: cnts(low) = 1013 cnts(peak)=  1247 cnts(high) =   903
espfilt:-  3rd parms =  1247.000    0.747    0.100
espfilt:- 3rd rl =  1261.334    0.755    0.164 / rh =  1261.334    0.755    0.164
espfilt:- 3rd cl =  2463.076 / ch =  2463.076
espfilt:- H_IMME =  0.627     H_TOTL = 0.2520     L_IMME = 0.6781     L_TOTL = -.8937E-01
espfilt:- WRITE_QDP_GTI: Norm =  1261.334 Width =     0.164 Center =     0.755
gtibuild:- Executing (routine): gtibuild file=mos1S001-gti-2500-8500.txt table=mos1S001-
↪gti-2500-8500.fits  -w 1 -V 4
gtibuild:- gtibuild (gtibuild-1.5)  [xmmsas_20211130_0941-20.0.0] started:  2024-04-
↪24T19:20:05.000
gtibuild:- gtibuild (gtibuild-1.5)  [xmmsas_20211130_0941-20.0.0] ended:    2024-04-
↪24T19:20:05.000
espfilt:-  HISTOGRAM_METHOD: *CLEANED (flare-free) FOV EVENTLIST FILE* = mos1S001-allevc-
↪2500-8500.fits
espfilt:-  *Flare free eventlist evselect command =
evselect:- Executing (routine): evselect table=/Users/dt237/code/DAXA/docs/source/
↪notebooks/tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_
↪pointed/0204310101/events/obsid0204310101-instM1-subexpS001-events.fits␣
↪filteredset=mos1S001-allevc-2500-8500.fits withfilteredset=yes keepfilteroutput=yes␣
↪flagcolumn=EVFLAG flagbit=-1 destruct=yes dssblock='' expression='(PATTERN<=12)&&
↪GTI(mos1S001-gti-2500-8500.fits,TIME)&&((FLAG & 0x766aa000)==0)' filtertype=expression␣
↪cleandss=no updateexposure=yes filterexposure=yes writedss=yes blockstocopy=''␣
↪attributestocopy='' energycolumn=PHA zcolumn=WEIGHT zerrorcolumn=EWEIGHT␣
↪withzerrorcolumn=no withzcolumn=no ignorelegallimits=no imageset=image.fits␣
↪xcolumn=RAWX ycolumn=RAWY ximagebinsize=1 yimagebinsize=1 squarepixels=no␣
↪ximagesize=600 yimagesize=600 imagebinning=imageSize ximagemin=1 ximagemax=640␣
↪withxranges=no yimagemin=1 yimagemax=640 withyranges=no imagedatatype=Real64␣
↪withimagedatatype=no raimagecenter=0 decimagecenter=0 withcelestialcenter=no␣
↪withimageset=no spectrumset=spectrum.fits spectralbinsize=5 specchannelmin=0␣
↪specchannelmax=11999 withspecranges=no nonStandardSpec=no withspectrumset=no␣
↪rateset=rate.fits timecolumn=TIME timebinsize=1 timemin=0 timemax=1000␣
↪withtimeranges=no maketimecolumn=no makeratecolumn=no withrateset=no␣
↪histogramset=histo.fits histogramcolumn=TIME histogrambinsize=1 histogrammin=0␣
↪histogrammax=1000 withhistoranges=no withhistogramset=no  -w 1 -V 4
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] started:  2024-04-
↪24T19:20:05.000
evselect:- selected 89183 rows from the input table.
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] ended:    2024-04-
↪24T19:20:05.000
espfilt:-  HISTOGRAM_METHOD: *CLEANED (flare-free) FOV IMAGE FILE* = mos1S001-allimc-
↪2500-8500.fits
espfilt:-  *Flare free FOV image evselect command =
evselect:- Executing (routine): evselect table=mos1S001-allevc-2500-8500.fits␣
↪filteredset=filtered.fits withfilteredset=no keepfilteroutput=no flagcolumn=EVFLAG␣
↪flagbit=-1 destruct=yes dssblock='' expression=true filtertype=expression cleandss=no␣
↪updateexposure=yes filterexposure=yes writedss=yes blockstocopy='' attributestocopy=''␣
↪energycolumn=PHA zcolumn=WEIGHT zerrorcolumn=EWEIGHT withzerrorcolumn=no␣
↪withzcolumn=no ignorelegallimits=no imageset=mos1S001-allimc-2500-8500.fits␣
↪xcolumn=DETX ycolumn=DETY ximagebinsize=1 yimagebinsize=1 squarepixels=no␣
↪ximagesize=600 yimagesize=600 imagebinning=imageSize ximagemin=1 ximagemax=640␣
↪withxranges=no yimagemin=1 yimagemax=640 withyranges=no imagedatatype=Real64␣
↪withimagedatatype=no raimagecenter=0 decimagecenter=0 withcelestialcenter=no␣
↪withimageset=yes spectrumset=spectrum.fits spectralbinsize=5 specchannelmin=0␣
↪specchannelmax=11999 withspecranges=no nonStandardSpec=no withspectrumset=no␣
↪rateset=rate.fits timecolumn=TIME timebinsize=1 timemin=0 timemax=1000␣
↪withtimeranges=no maketimecolumn=no makeratecolumn=no withrateset=no␣
↪histogramset=histo.fits histogramcolumn=TIME histogrambinsize=1 histogrammin=0␣
↪histogrammax=1000 withhistoranges=no withhistogramset=no  -w 1 -V 4
```

(continues on next page)

(continued from previous page)

```
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] started:  2024-04-
↪24T19:20:06.000
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] ended:    2024-04-
↪24T19:20:06.000
espfilt:-  HISTOGRAM_METHOD: *CLEANED (flare-free) CORNER EVENTLIST FILE* = mos1S001-
↪corevc-2500-8500.fits
espfilt:-  *Flare free corner eventlist evselect command =
evselect:- Executing (routine): evselect table=/Users/dt237/code/DAXA/docs/source/
↪notebooks/tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_
↪pointed/0204310101/events/obsid0204310101-instM1-subexpS001-events.fits:EVENTS␣
↪filteredset=mos1S001-corevc-2500-8500.fits withfilteredset=yes keepfilteroutput=yes␣
↪flagcolumn=EVFLAG flagbit=-1 destruct=yes dssblock='' expression='(PATTERN<=12)&&
↪GTI(mos1S001-gti-2500-8500.fits,TIME)&&((FLAG & 0x766aa000)==0)&&!(CIRCLE(100,-200,
↪17700,DETX,DETY)||CIRCLE(834,135,17100,DETX,DETY)||CIRCLE(770,-803,17100,DETX,
↪DETY)||BOX(-20,-17000,6500,500,0,DETX,DETY)||BOX(5880,-20500,7500,1500,10,DETX,
↪DETY)||BOX(-5920,-20500,7500,1500,350,DETX,DETY)||BOX(-20,-20000,5500,500,0,DETX,
↪DETY)||BOX(-12900,16000,250,4000,0,DETX,DETY)||BOX(80,18600,150,1300,0,DETX,
↪DETY)||BOX(-10,-18800,125,1500,0,DETX,DETY))' filtertype=expression cleandss=no␣
↪updateexposure=yes filterexposure=yes writedss=yes blockstocopy='' attributestocopy=''␣
↪energycolumn=PHA zcolumn=WEIGHT zerrorcolumn=EWEIGHT withzerrorcolumn=no␣
↪withzcolumn=no ignorelegallimits=no imageset=image.fits xcolumn=RAWX ycolumn=RAWY␣
↪ximagebinsize=1 yimagebinsize=1 squarepixels=no ximagesize=600 yimagesize=600␣
↪imagebinning=imageSize ximagemin=1 ximagemax=640 withxranges=no yimagemin=1␣
↪yimagemax=640 withyranges=no imagedatatype=Real64 withimagedatatype=no raimagecenter=0␣
↪decimagecenter=0 withcelestialcenter=no withimageset=no spectrumset=spectrum.fits␣
↪spectralbinsize=5 specchannelmin=0 specchannelmax=11999 withspecranges=no␣
↪nonStandardSpec=no withspectrumset=no rateset=rate.fits timecolumn=TIME timebinsize=1␣
↪timemin=0 timemax=1000 withtimeranges=no maketimecolumn=no makeratecolumn=no␣
↪withrateset=no histogramset=histo.fits histogramcolumn=TIME histogrambinsize=1␣
↪histogrammin=0 histogrammax=1000 withhistoranges=no withhistogramset=no  -w 1 -V 4
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] started:  2024-04-
↪24T19:20:06.000
evselect:- selected 11558 rows from the input table.
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] ended:    2024-04-
↪24T19:20:07.000
espfilt:-  HISTOGRAM_METHOD: *CLEANED (flare-free) CORNER IMAGE FILE* = mos1S001-corimc-
↪2500-8500.fits
espfilt:-  *Flare free corner image evselect command =
evselect:- Executing (routine): evselect table=mos1S001-corevc-2500-8500.fits␣
↪filteredset=filtered.fits withfilteredset=no keepfilteroutput=no flagcolumn=EVFLAG␣
↪flagbit=-1 destruct=yes dssblock='' expression=true filtertype=expression cleandss=no␣
↪updateexposure=yes filterexposure=yes writedss=yes blockstocopy='' attributestocopy=''␣
↪energycolumn=PHA zcolumn=WEIGHT zerrorcolumn=EWEIGHT withzerrorcolumn=no␣
↪withzcolumn=no ignorelegallimits=no imageset=mos1S001-corimc-2500-8500.fits␣
↪xcolumn=DETX ycolumn=DETY ximagebinsize=1 yimagebinsize=1 squarepixels=no␣
↪ximagesize=600 yimagesize=600 imagebinning=imageSize ximagemin=1 ximagemax=640␣
↪withxranges=no yimagemin=1 yimagemax=640 withyranges=no imagedatatype=Real64␣
↪withimagedatatype=no raimagecenter=0 decimagecenter=0 withcelestialcenter=no␣
↪withimageset=yes spectrumset=spectrum.fits spectralbinsize=5 specchannelmin=0␣
↪specchannelmax=11999 withspecranges=no nonStandardSpec=no withspectrumset=no␣
↪rateset=rate.fits timecolumn=TIME timebinsize=1 timemin=0 timemax=1000␣
↪withtimeranges=no maketimecolumn=no makeratecolumn=no withrateset=no␣
↪histogramset=histo.fits histogramcolumn=TIME histogrambinsize=1 histogrammin=0␣
↪histogrammax=1000 withhistoranges=no withhistogramset=no  -w 1 -V 4
```

(continues on next page)

(continued from previous page)

```
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] started:  2024-04-
↪24T19:20:07.000
evselect:- evselect (evselect-3.71.1)  [xmmsas_20211130_0941-20.0.0] ended:    2024-04-
↪24T19:20:07.000
espfilt:-  HISTOGRAM_METHOD: Adding residuals to cleaned event lists
espfilt:- Normalization =  1261.334
espfilt:- Width         =     0.164
espfilt:- Center        =     0.755
espfilt:-  HISTOGRAM_METHOD: Return with no errors
espfilt:-  ESPFILT: **** keepinterfiles NOT set
espfilt:-  ESPFILT: **** Deleting these intermediary files:
espfilt:-  Unfiltered FOV image: mos1S001-fovim-2500-8500.fits
espfilt:-  Corner-only image: mos1S001-corim-2500-8500.fits
espfilt:-  GTI text file for gtibuild: mos1S001-gti-2500-8500.txt
espfilt:- espfilt (espfilt-4.3)  [xmmsas_20211130_0941-20.0.0] ended:    2024-04-24T19:
↪20:07.000
```

Process Raw Errors (stderr)

Related to the previous section is the `raw_process_errors` property, which stores the stderr output of any process that generated one - note that if no stderr was produced, then we do not create an entry. DAXA makes a distinction between 'raw process errors' and the 'process errors' you'll see in the next section - this is because we attempt to parse any stderr and extract particular errors, versus this property which is just the raw text:

[22]:
```
prev_arch.raw_process_errors
```

[22]:
```
{'xmm_pointed': {'cif_build': {},
  'odf_ingest': {},
  'epchain': {},
  'emchain': {},
  'espfilt': {'0502671101PNS003': '** evselect: warning (NoWCS), No WCS information␣
↪available for image column DETX.\n** evselect: warning (SummaryOfWarnings), \n  ␣
↪warning NoWCS silently occurred 1 times\n** evselect: warning (NoWCS), No WCS␣
↪information available for image column DETX.\n** evselect: warning (SummaryOfWarnings),
↪ \n   warning NoWCS silently occurred 1 times\n** espfilt: error (noCounts), All histo␣
↪counts are zero! Check your FOV Lightcurve!\nmv: rename pnS003-gti-2500-8500.fits to /
↪Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_output/archives/PHL1811_
↪made_earlier/processed_data/xmm_pointed/0502671101/cleaning/obsid0502671101-instPN-
↪subexpS003-en2500_8500keV-gti.fits: No such file or directory\nmv: rename pnS003-
↪allevc-2500-8500.fits to /Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_
↪output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/0502671101/cleaning/
↪PNS003-allevc-2500-8500.fits: No such file or directory\nmv: rename pnS003-hist-2500-
↪8500.qdp to /Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_output/
↪archives/PHL1811_made_earlier/processed_data/xmm_pointed/0502671101/cleaning/
↪obsid0502671101-instPN-subexpS003-en2500_8500keV-hist.qdp: No such file or directory\n
↪'},
  'cleaned_evt_lists': {},
  'merge_subexposures': {}},
 'chandra': {'preprocessed_events': {}, 'preprocessed_images': {}},
 'nustar_pointed': {'preprocessed_events': {}, 'preprocessed_images': {}},
```

(continues on next page)

(continued from previous page)

```
 'rosat_all_sky': {'preprocessed_events': {},
  'preprocessed_images': {},
  'preprocessed_expmaps': {},
  'preprocessed_backmaps': {}}}
```

Process Parsed Errors (parsed from stderr)

We attempt to extract the pertinent information from the raw error outputs, and this is what gets stored in `process_errors` - again with the same storage structure. Here we just show a single entry, for one of the pieces of data that we noted had failed a processing step in the `process_success` section of this tutorial:

[23]:
```
prev_arch.process_errors['xmm_pointed']['espfilt']['0502671101PNS003']
```

[23]:
```
['noCounts raised by espfilt - All histo counts are zero! Check your FOV Lightcurve!']
```

Process Parsed Warnings (parsed from stderr)

We make a distinction between errors and warnings, as do many pieces of backend software. The `process_warnings` property acts exactly as the `process_errors` property, but it is warnings that have been extracted rather than errors:

[24]:
```
prev_arch.process_warnings['xmm_pointed']['epchain']["0502671101PNS003"][109]
```

[24]:
```
{'originator': 'epframes',
 'name': 'notHKconstant',
 'message': 'HK parameter F1725 is not constant, range = [           0 ,         256 ].␣
↪Use default:          256'}
```

Process Extra Information

The `process_extra_info` property is something of a catch-all for any information passed into, or produced by, a processing step that the archive might need access to later on. This can include things like the paths to the key output files of a process. This isn't really meant to be useful to the user, but can still be accessed.

We only show the contents for one processing step (`cleaned_evt_lists`) so as not to make the tutorial too long:

[25]:
```
prev_arch.process_extra_info['xmm_pointed']['cleaned_evt_lists']
```

[25]:
```
{'0105261001M1S001': {'evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/
↪tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/
↪0105261001/events/obsid0105261001-instM1-subexpS001-en-cleanevents.fits',
  'en_key': ''},
 '0105261001M2S002': {'evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/
↪tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/
↪0105261001/events/obsid0105261001-instM2-subexpS002-en-cleanevents.fits',
  'en_key': ''},
 '0105261001M1U002': {'evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/
↪tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/
↪0105261001/events/obsid0105261001-instM1-subexpU002-en-cleanevents.fits',
  'en_key': ''},
 '0105261001M2U002': {'evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/
↪tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/
↪0105261001/events/obsid0105261001-instM2-subexpU002-en-cleanevents.fits',
```

(continues on next page)

(continued from previous page)

```
  'en_key': ''},
 '0761910201M1S001': {'evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/
↪tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/
↪0761910201/events/obsid0761910201-instM1-subexpS001-en-cleanevents.fits',
  'en_key': ''},
 '0204310101M1S001': {'evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/
↪tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/
↪0204310101/events/obsid0204310101-instM1-subexpS001-en-cleanevents.fits',
  'en_key': ''},
 '0204310101M2S002': {'evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/
↪tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/
↪0204310101/events/obsid0204310101-instM2-subexpS002-en-cleanevents.fits',
  'en_key': ''},
 '0502671101M1S001': {'evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/
↪tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/
↪0502671101/events/obsid0502671101-instM1-subexpS001-en-cleanevents.fits',
  'en_key': ''},
 '0502671101M2S002': {'evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/
↪tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/
↪0502671101/events/obsid0502671101-instM2-subexpS002-en-cleanevents.fits',
  'en_key': ''},
 '0761910201M2S002': {'evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/
↪tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/
↪0761910201/events/obsid0761910201-instM2-subexpS002-en-cleanevents.fits',
  'en_key': ''},
 '0105261001PNS003': {'evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/
↪tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/
↪0105261001/events/obsid0105261001-instPN-subexpS003-en-cleanevents.fits',
  'en_key': '',
  'oot_evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_
↪output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/0105261001/events/
↪obsid0105261001-instPN-subexpS003-en-cleanootevents.fits'},
 '0105261001PNU002': {'evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/
↪tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/
↪0105261001/events/obsid0105261001-instPN-subexpU002-en-cleanevents.fits',
  'en_key': '',
  'oot_evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_
↪output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/0105261001/events/
↪obsid0105261001-instPN-subexpU002-en-cleanootevents.fits'},
 '0204310101PNS003': {'evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/
↪tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/
↪0204310101/events/obsid0204310101-instPN-subexpS003-en-cleanevents.fits',
  'en_key': '',
  'oot_evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_
↪output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/0204310101/events/
↪obsid0204310101-instPN-subexpS003-en-cleanootevents.fits'},
 '0761910201PNS003': {'evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/
↪tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/
↪0761910201/events/obsid0761910201-instPN-subexpS003-en-cleanevents.fits',
  'en_key': '',
  'oot_evt_clean_path': '/Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_
↪output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/0761910201/events/
↪obsid0761910201-instPN-subexpS003-en-cleanootevents.fits'}}
```

(continues on next page)

(continued from previous page)

Final process success

This property (`final_process_success`) is the final arbiter of whether a particular ObsID of a mission can be used by the end user - this is only decided when whatever DAXA defines as the 'final processing step' for a particular mission is run (for XMM this is the assembly of cleaned event lists).

If it is False, then none of the data for that ObsID reached the final processing step - note here that 0102041001 is False, this is because all the instruments were in calibration mode with the filter closed. As such, this ObsID is moved from the 'processed_data' directory in the archive storage structure, to the 'failed_data' directory.

[26]:
```
prev_arch.final_process_success
```

[26]:
```
{'xmm_pointed': {'0102041001': False,
  '0105261001': True,
  '0204310101': True,
  '0502671101': True,
  '0761910201': True},
 'chandra': {},
 'nustar_pointed': {},
 'rosat_all_sky': {}}
```

Observation Summaries

This property (`observation_summaries`) contains a summary of the exact data available for particular observations of a mission, and is meant to allow DAXA processes to access whether they can run on a particular ObsID (not really relevant to end users). The property is populated in different ways for each mission - the XMMPointed mission, for instance, parses the output summary file from `odf_ingest` and turns it into an extremely detailed summary of the state of each instrument on XMM for an observation.

For clarity we only show the output for one ObsID:

[27]:
```
prev_arch.observation_summaries['xmm_pointed']['0204310101']
```

[27]:
```
{'M1': {'active': True,
  'num_exp': 1,
  'exposures': {'S001': {'scheduled': True,
    'type': 'SCIENCE',
    'mode': 'PrimeFullWindow',
    'filter': 'Thin1',
    'ccd_modes': {'1': 'Imaging',
     '2': 'Imaging',
     '3': 'Imaging',
     '4': 'Imaging',
     '5': 'Imaging',
     '6': 'Imaging',
     '7': 'Imaging'}}}},
 'M2': {'active': True,
  'num_exp': 1,
  'exposures': {'S002': {'scheduled': True,
    'type': 'SCIENCE',
```

(continues on next page)

(continued from previous page)

```
    'mode': 'PrimeFullWindow',
    'filter': 'Thin1',
    'ccd_modes': {'1': 'Imaging',
     '2': 'Imaging',
     '3': 'Imaging',
     '4': 'Imaging',
     '5': 'Imaging',
     '6': 'Imaging',
     '7': 'Imaging'}}},
 'PN': {'active': True,
  'num_exp': 14,
  'exposures': {'S003': {'scheduled': True,
    'type': 'SCIENCE',
    'mode': 'PrimeFullWindow',
    'filter': 'Thin1',
    'ccd_modes': {'1': 'Imaging',
     '2': 'Imaging',
     '3': 'Imaging',
     '4': 'Imaging',
     '5': 'Imaging',
     '6': 'Imaging',
     '7': 'Imaging',
     '8': 'Imaging',
     '9': 'Imaging',
     '10': 'Imaging',
     '11': 'Imaging',
     '12': 'Imaging'}},
   'U002': {'scheduled': False,
    'type': 'SCIENCE',
    'mode': 'Diagnostic',
    'filter': None,
    'ccd_modes': {'1': 'DiscardedLinesData'}},
   'U003': {'scheduled': False,
    'type': 'SCIENCE',
    'mode': 'Diagnostic',
    'filter': None,
    'ccd_modes': {'2': 'DiscardedLinesData'}},
   'U004': {'scheduled': False,
    'type': 'SCIENCE',
    'mode': 'Diagnostic',
    'filter': None,
    'ccd_modes': {'3': 'DiscardedLinesData'}},
   'U005': {'scheduled': False,
    'type': 'SCIENCE',
    'mode': 'Diagnostic',
    'filter': None,
    'ccd_modes': {'4': 'DiscardedLinesData'}},
   'U006': {'scheduled': False,
    'type': 'SCIENCE',
    'mode': 'Diagnostic',
    'filter': None,
    'ccd_modes': {'5': 'DiscardedLinesData'}},
```

(continues on next page)

(continued from previous page)

```
  'U007': {'scheduled': False,
   'type': 'SCIENCE',
   'mode': 'Diagnostic',
   'filter': None,
   'ccd_modes': {'6': 'DiscardedLinesData'}},
  'U008': {'scheduled': False,
   'type': 'SCIENCE',
   'mode': 'Diagnostic',
   'filter': None,
   'ccd_modes': {'7': 'DiscardedLinesData'}},
  'U009': {'scheduled': False,
   'type': 'SCIENCE',
   'mode': 'Diagnostic',
   'filter': None,
   'ccd_modes': {'8': 'DiscardedLinesData'}},
  'U010': {'scheduled': False,
   'type': 'SCIENCE',
   'mode': 'Diagnostic',
   'filter': None,
   'ccd_modes': {'9': 'DiscardedLinesData'}},
  'U011': {'scheduled': False,
   'type': 'SCIENCE',
   'mode': 'Diagnostic',
   'filter': None,
   'ccd_modes': {'10': 'DiscardedLinesData'}},
  'U012': {'scheduled': False,
   'type': 'SCIENCE',
   'mode': 'Diagnostic',
   'filter': None,
   'ccd_modes': {'11': 'DiscardedLinesData'}},
  'U013': {'scheduled': False,
   'type': 'SCIENCE',
   'mode': 'Diagnostic',
   'filter': None,
   'ccd_modes': {'12': 'DiscardedLinesData'}},
  'U101': {'scheduled': False,
   'type': 'SCIENCE',
   'mode': 'Offset',
   'filter': 'Thin1',
   'ccd_modes': {'1': 'OffsetData',
    '2': 'OffsetData',
    '3': 'OffsetData',
    '4': 'OffsetData',
    '5': 'OffsetData',
    '6': 'OffsetData',
    '7': 'OffsetData',
    '8': 'OffsetData',
    '9': 'OffsetData',
    '10': 'OffsetData',
    '11': 'OffsetData',
    '12': 'OffsetData'}}}}}
```

3.3.8 Data management Archive functions

This section introduces the built-in methods that allow you to manage the archive and its data:

Get path to data

One of the most useful methods of the archive class is `get_current_data_path()`, which allows you to programmatically retrieve the current path to a particular ObsID's data in the archive storage structure (this takes into account the `final_process_success` flag - remember that entirely failed data are moved to a 'failed_data' directory.

All we need to do is pass the mission name and the ObsID, and the top-level data path will be returned. Here we show two examples, one for an ObsID with a True final success flag, and one with a False final success flag:

```
[28]: prev_arch.get_current_data_path('xmm_pointed', '0204310101')
```

```
[28]: '/Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_output/archives/PHL1811_
      ↪made_earlier/processed_data/xmm_pointed/0204310101/'
```

```
[29]: prev_arch.get_current_data_path('xmm_pointed', '0102041001')
```

```
[29]: '/Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_output/archives/PHL1811_
      ↪made_earlier/failed_data/xmm_pointed/0102041001/'
```

Get failed processes

Often we will wish to know exactly which data failed a particular processing step - this could be inferred from the `process_success` property, but we also provide a convenient get method (`get_failed_processes()`) that will retrieve the unique identifer of each piece of data that failed a specified processing step:

```
[30]: prev_arch.get_failed_processes('espfilt')
```

```
[30]: {'xmm_pointed': ['0502671101PNS003']}
```

Get logs

The `get_process_logs()` method provides a more convenient way of accessing specific logs stored in the `process_logs` property. It allows for the retrieval of logs based on several criteria, with the only requirement being the passing of a process name.

Beyond that you can specify the mission name, ObsID, and instrument to retrieve logs for - as for some processes there are sub-exposures of a given instrument, giving this information can still result in multiple logs being returned. You can also pass **lists** of mission name, ObsID, and instrument and retrieve sets of logs that way.

It is also possible to specify an exact unique identifier (0502671101M2S002 for instance).

We demonstrate fetching the `espfilt` logs for a single PN instrument of 0502671101:

```
[31]: prev_arch.get_process_logs('espfilt', obs_id='0502671101', inst='PN')
```

```
[31]: {'xmm_pointed': {'0502671101PNS003': "espfilt:- Executing (routine): espfilt eventfile=/
      ↪Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_output/archives/PHL1811_
      ↪made_earlier/processed_data/xmm_pointed/0502671101/events/obsid0502671101-instPN-
      ↪subexpS003-events.fits withoot=yes ootfile=/Users/dt237/code/DAXA/docs/source/
      ↪notebooks/tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_
      ↪pointed/0502671101/events/obsid0502671101-instPN-subexpS003-ootevents.fits␣
      ↪method=histogram withsmoothing=yes smooth=51 withbinning=yes binsize=60 withoutput=yes␣
      ↪withlongnames=yes elow=2500 ehigh=8500 rangescale=15 allowsigma=3 limits='0.1 6.5'␣
      ↪keepinterfiles=no  -w 1 -V 4\nespfilt:- espfilt (espfilt-4.3)  [xmmsas_20211130_0941-
      ↪20.0.0] started:  2024-04-24T19:20:07.000\nespfilt:-  ESPFILT: Processing eventlist: /
      ↪Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_output/archives/PHL1811_
      ↪made_earlier/processed_data/xmm_pointed/0502671101/events/obsid0502671101-instPN-
      ↪subexpS003-events.fits\nespfilt:-  *FOV IMAGE* = pnS003-fovim-2500-8500.fits\nevselect:
```

(continues on next page)

(continued from previous page)

Get raw errors

There is also an equivalent method, `get_process_raw_error_logs()` that can fetch raw error logs. It behaves exactly the same as `get_process_logs()`, see the last section for information on the possible arguments:

[32]:
```
prev_arch.get_process_raw_error_logs('espfilt', obs_id='0502671101', inst='PN')
```

[32]:
```
{'xmm_pointed': {'0502671101PNS003': '** evselect: warning (NoWCS), No WCS information␣
↪available for image column DETX.\n** evselect: warning (SummaryOfWarnings), \n  ␣
↪warning NoWCS silently occurred 1 times\n** evselect: warning (NoWCS), No WCS␣
↪information available for image column DETX.\n** evselect: warning (SummaryOfWarnings),
↪  \n   warning NoWCS silently occurred 1 times\n** espfilt: error (noCounts), All histo␣
↪counts are zero! Check your FOV Lightcurve!\nmv: rename pnS003-gti-2500-8500.fits to /
↪Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_output/archives/PHL1811_
↪made_earlier/processed_data/xmm_pointed/0502671101/cleaning/obsid0502671101-instPN-
↪subexpS003-en2500_8500keV-gti.fits: No such file or directory\nmv: rename pnS003-
↪allevc-2500-8500.fits to /Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_
↪output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/0502671101/cleaning/
↪PNS003-allevc-2500-8500.fits: No such file or directory\nmv: rename pnS003-hist-2500-
↪8500.qdp to /Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_output/
↪archives/PHL1811_made_earlier/processed_data/xmm_pointed/0502671101/cleaning/
↪obsid0502671101-instPN-subexpS003-en2500_8500keV-hist.qdp: No such file or directory\n
↪'}}
```

Get failed logs

There is one final method that allows for log retrieval - `get_failed_logs()`. This only takes a process name as an argument, and will retrieve the logs and raw error logs for every piece of data that failed the specified processing step. It returns them as a tuple, with the first entry being the dictionary of logs and the second being the dictionary of raw errors:

[33]:
```
logs, errors = prev_arch.get_failed_logs('espfilt')
```

We only show 0502671101M2S002:

[34]:
```
logs['xmm_pointed']['0502671101PNS003']
```

[34]:
```
"espfilt:- Executing (routine): espfilt eventfile=/Users/dt237/code/DAXA/docs/source/
↪notebooks/tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_
↪pointed/0502671101/events/obsid0502671101-instPN-subexpS003-events.fits withoot=yes␣
↪ootfile=/Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_output/archives/
↪PHL1811_made_earlier/processed_data/xmm_pointed/0502671101/events/obsid0502671101-
↪instPN-subexpS003-ootevents.fits method=histogram withsmoothing=yes smooth=51␣
↪withbinning=yes binsize=60 ratio=1.2 withlongnames=yes elow=2500 ehigh=8500␣
↪rangescale=15 allowsigma=3 limits='0.1 6.5' keepinterfiles=no  -w 1 -V 4\nespfilt:-␣
↪espfilt (espfilt-4.3)  [xmmsas_20211130_0941-20.0.0] started:  2024-04-24T19:20:07.000\
↪nespfilt:-  ESPFILT: Processing eventlist: /Users/dt237/code/DAXA/docs/source/
↪notebooks/tutorials/daxa_output/archives/PHL1811_made_earlier/processed_data/xmm_
↪pointed/0502671101/events/obsid0502671101-instPN-subexpS003-events.fits\nespfilt:- ␣
↪*FOV IMAGE* = pnS003-fovim-2500-8500.fits\nevselect:- Executing (routine): evselect␣
↪table=/Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_output/archives/
↪PHL1811_made_earlier/processed_data/xmm_pointed/0502671101/events/obsid0502671101-
↪instPN-subexpS003-events.fits filteredset=filtered.fits withfilteredset=no␣
↪keepfilteroutput=no flagcolumn=EVFLAG flagbit=-1 destruct=yes dssblock=''␣
↪expression=true filtertype=expression cleandss=no updateexposure=yes␣
↪filterexposure=yes writedss=yes blockstocopy='' attributestocopy='' energycolumn=PHA␣
↪zcolumn=WEIGHT zerrorcolumn=EWEIGHT withzerrorcolumn=no withzcolumn=no␣
```

(continues on next page)

(continued from previous page)

[35]:
```
errors
```

[35]:
```
{'xmm_pointed': {'0502671101PNS003': '** evselect: warning (NoWCS), No WCS information␣
↪available for image column DETX.\n** evselect: warning (SummaryOfWarnings), \n  ␣
↪warning NoWCS silently occurred 1 times\n** evselect: warning (NoWCS), No WCS␣
↪information available for image column DETX.\n** evselect: warning (SummaryOfWarnings),
↪ \n   warning NoWCS silently occurred 1 times\n** espfilt: error (noCounts), All histo␣
↪counts are zero! Check your FOV Lightcurve!\nmv: rename pnS003-gti-2500-8500.fits to /
↪Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_output/archives/PHL1811_
↪made_earlier/processed_data/xmm_pointed/0502671101/cleaning/obsid0502671101-instPN-
↪subexpS003-en2500_8500keV-gti.fits: No such file or directory\nmv: rename pnS003-
↪allevc-2500-8500.fits to /Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_
↪output/archives/PHL1811_made_earlier/processed_data/xmm_pointed/0502671101/cleaning/
↪PNS003-allevc-2500-8500.fits: No such file or directory\nmv: rename pnS003-hist-2500-
↪8500.qdp to /Users/dt237/code/DAXA/docs/source/notebooks/tutorials/daxa_output/
↪archives/PHL1811_made_earlier/processed_data/xmm_pointed/0502671101/cleaning/
↪obsid0502671101-instPN-subexpS003-en2500_8500keV-hist.qdp: No such file or directory\n
↪'}}
```

3.3.9 Adding region files to the Archive (optional)

Region files are created by running source detection algorithms on images generated from X-ray observations, and DAXA **does not yet have the ability to generate them itself**. However, they are a crucial part of many analyses, and as such a crucial part of data archives.

If you have created region files (**in the DS9 format**), you can add them to the archive so that they are included in the storage structure. This process verifies that the passed region files are in RA-Dec coordinates (rather than defined in the pixel coordinates of the image they were detected in), by requiring that images or WCS information be passed in those circumstances.

- {'mission_name': {'ObsID': 'path to regions'}}
- {'mission_name': {'ObsID': [list of region objects]}}
- {'mission_name': {'ObsID': {'region': 'path to regions'}}}
- {'mission_name': {'ObsID': {'region': [list of region objects]}}}
- {'mission_name': {'ObsID': {'region': ..., 'wcs_src': 'path to image'}}}
- {'mission_name': {'ObsID': {'region': ..., 'wcs_src': XGA Image}}}
- {'mission_name': {'ObsID': {'region': ..., 'wcs_src': Astropy WCS object}}}

For instance, we have used a tool to generate source regions for the observations we have processed, and want to add those regions to the archive. As the region files are in pixel coordinates we pass the paths to the images we used (unnecessary if regions are already in RA-Dec coordinates):

[37]:
```
# Listing the region files in the test directory
reg_files = os.listdir('region_files')

# Setting up the structure of the dictionary we will pass to the archive at the end of␣
↪this
reg_paths = {'xmm_pointed': {}}
```

(continues on next page)

(continued from previous page)

```
# Iterating through the ObsIDs in the XMMPointed mission
for oi in prev_arch['xmm_pointed'].filtered_obs_ids:
    # Checking to see which have a corresponding region file
    if any([oi in rf for rf in reg_files]):
        # Generating the path to the image we need for pixel to RA-Dec conversion
        im_pth = prev_arch.get_current_data_path('xmm_pointed', oi) + \
        'images/obsid{}-instM2-subexpALL-en0.5_2.0keV-image.fits'.format(oi)
        # Setting up the entry in the final dictionary, with the path to the regions and␣
↪the image
        reg_paths['xmm_pointed'][oi] = {'regions': 'region_files/{}.reg'.format(oi),
↪'wcs_src': im_pth}

# Adding the regions to the archive
prev_arch.source_regions = reg_paths
```

```
/var/folders/td/gw9qkx6d3szb1nkt_cfvcbzm000vzl/T/ipykernel_30819/3922313599.py:17:␣
↪UserWarning: The source_regions property already had an entry for 0105261001 under xmm_
↪pointed, this has been overwritten!
  prev_arch.source_regions = reg_paths
/var/folders/td/gw9qkx6d3szb1nkt_cfvcbzm000vzl/T/ipykernel_30819/3922313599.py:17:␣
↪UserWarning: The source_regions property already had an entry for 0204310101 under xmm_
↪pointed, this has been overwritten!
  prev_arch.source_regions = reg_paths
/var/folders/td/gw9qkx6d3szb1nkt_cfvcbzm000vzl/T/ipykernel_30819/3922313599.py:17:␣
↪UserWarning: The source_regions property already had an entry for 0502671101 under xmm_
↪pointed, this has been overwritten!
  prev_arch.source_regions = reg_paths
/var/folders/td/gw9qkx6d3szb1nkt_cfvcbzm000vzl/T/ipykernel_30819/3922313599.py:17:␣
↪UserWarning: The source_regions property already had an entry for 0761910201 under xmm_
↪pointed, this has been overwritten!
  prev_arch.source_regions = reg_paths
```

Once added to the archive, you can also retrieve the regions through a property (we did not discuss it earlier in this tutorial) - `source_regions`. It returns them as Python 'regions' module objects:

[38]:
```
prev_arch.source_regions
```

[38]:
```
{'xmm_pointed': {'0105261001': [<EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
        (67.59133516, -61.53306334)>, width=0.007545166292795352 deg, height=0.
↪007545166292795352 deg, angle=186.39335083127384 deg)>,
   <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
        (67.178534, -61.37451667)>, width=0.008274114359963867 deg, height=0.
↪008274114359963867 deg, angle=466.9975179351541 deg)>,
   <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
        (67.46879016, -61.36577221)>, width=0.00850673285572214 deg, height=0.
↪00850673285572214 deg, angle=140.94986422162475 deg)>,
   <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
        (68.01317832, -61.35516589)>, width=0.007519977428013009 deg, height=0.
↪007519977428013009 deg, angle=261.2582496754984 deg)>,
   <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
        (68.00486199, -61.33624502)>, width=0.013147992908181612 deg, height=0.
↪013147992908181612 deg, angle=265.95154602021546 deg)>,
   <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
```

(continues on next page)

(continued from previous page)

```
      (67.41445681, -61.28706975)>, width=0.007512080273205201 5 deg, height=0.
↪007512080273205201 5 deg, angle=419.76253296921374 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.52454168, -61.26549895)>, width=0.007888040300578339 deg, height=0.
↪007888040300578339 deg, angle=368.13995128822404 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.38459977, -61.24728435)>, width=0.007642156673586847 deg, height=0.
↪007642156673586847 deg, angle=405.1377277122432 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.66282973, -61.13583214)>, width=0.007484864530890935 deg, height=0.
↪007484864530890935 deg, angle=342.51362589811765 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.67081204, -61.41366132)>, width=0.007664217192863797 deg, height=0.
↪007664217192863797 deg, angle=214.13162270745238 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.44099122, -61.2872128)>, width=0.008544191122124041 deg, height=0.
↪008544191122124041 deg, angle=413.6812532986161 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.47859205, -61.17991392)>, width=0.008220698389583923 deg, height=0.
↪008220698389583923 deg, angle=372.0472646434019 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.62366961, -61.57349813)>, width=0.007591811484076978 deg, height=0.
↪007591811484076978 deg, angle=188.7899825817821 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.46197943, -61.31728401)>, width=0.009600351102606488 deg, height=0.
↪009600351102606488 deg, angle=441.75783953763363 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.109084, -61.3151228)>, width=0.007726056638463273 deg, height=0.
↪007726056638463273 deg, angle=448.4264497362917 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.19833167, -61.30776102)>, width=0.00700383303154554 deg, height=0.
↪00700383303154554 deg, angle=445.20514853614804 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.30056555, -61.23205705)>, width=0.007642141740875588 deg, height=0.
↪007642141740875588 deg, angle=412.2134543327273 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.63170002, -61.15814151)>, width=0.009628376655476337 deg, height=0.
↪009628376655476337 deg, angle=345.1369371937843 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.63227828, -61.15627305)>, width=0.009228444139956988 deg, height=0.
↪009228444139956988 deg, angle=345.2044298700188 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.67485433, -61.37499077)>, width=0.002790520972276921 6 deg, height=0.
↪001611109591257416 2 deg, angle=44.883076244857286 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.69462508, -61.33509251)>, width=0.031793397493982 03 deg, height=0.
↪022402867284854 13 deg, angle=44.63583059287267 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.92311013, -61.3253819)>, width=0.009572126121424652 deg, height=0.
↪002836789681040755 6 deg, angle=-257.05172906999195 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.57937638, -61.39856243)>, width=0.06787196199238821 deg, height=0.
↪022450986528974 39 deg, angle=-19.414157271481155 deg)>,
```

(continues on next page)

(continued from previous page)

```
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.85008628, -61.42796549)>, width=0.1397191347246724 deg, height=0.
↪04780714778773332 deg, angle=-237.4726619426717 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (67.42445028, -61.17772595)>, width=0.030515763924281118 deg, height=0.
↪01992090470829036 deg, angle=4.480105303940038 deg)>],
 '0204310101': [<EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.68906505, -9.57605556)>, width=0.007494624973566207 deg, height=0.
↪007494624973566207 deg, angle=514.3266958726614 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.68959451, -9.56149785)>, width=0.006759131697903771 deg, height=0.
↪006759131697903771 deg, angle=512.5796094746042 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.88756277, -9.52098468)>, width=0.007452465671294455 deg, height=0.
↪007452465671294455 deg, angle=579.5203052526585 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.80541496, -9.51795062)>, width=0.008375859232780183 deg, height=0.
↪008375859232780183 deg, angle=551.5787100398195 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.72559081, -9.51892627)>, width=0.007444719230222005 deg, height=0.
↪007444719230222005 deg, angle=517.5127356369848 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.55371487, -9.49046251)>, width=0.007642096373174 6 deg, height=0.
↪007642096373174 6 deg, angle=474.0587823060275 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.56596717, -9.4880535)>, width=0.007737029310810679 deg, height=0.
↪007737029310810679 deg, angle=474.7287829609348 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.82139591, -9.47361547)>, width=0.007737085424898903 deg, height=0.
↪007737085424898903 deg, angle=566.8961022892662 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.67183058, -9.44703777)>, width=0.0071365428053444206 deg, height=0.
↪0071365428053444206 deg, angle=477.7355081764938 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.74299637, -9.44486753)>, width=0.007846231696543896 deg, height=0.
↪007846231696543896 deg, angle=506.2988954235226 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.76086754, -9.41124584)>, width=0.008050640320955097 deg, height=0.
↪008050640320955097 deg, angle=497.7129792302257 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.8464958, -9.40473667)>, width=0.007737097719562047 deg, height=0.
↪007737097719562047 deg, angle=258.67400652493257 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.80730171, -9.39688777)>, width=0.007737102356468667 deg, height=0.
↪007737102356468667 deg, angle=258.914402158858 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.70381145, -9.38479759)>, width=0.007249993801450 57 deg, height=0.
↪007249993801450 57 deg, angle=444.90628957212795 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.83026421, -9.38399593)>, width=0.007865255978998582 deg, height=0.
↪007865255978998582 deg, angle=278.38565516990224 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
```

(continues on next page)

(continued from previous page)

```
      (328.8120376, -9.38092408)>, width=0.007338448567576843 deg, height=0.
↪007338448567576843 deg, angle=287.9436293684878 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.67039651, -9.37572647)>, width=0.007646681844692583 deg, height=0.
↪007646681844692583 deg, angle=441.6907421859443 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.75681337, -9.37342961)>, width=0.007919729453573972 deg, height=0.
↪007919729453573972 deg, angle=410.70964078256986 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.70810058, -9.37211098)>, width=0.00966665832004527 deg, height=0.
↪00966665832004527 deg, angle=434.593589673473 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.72034748, -9.37150905)>, width=0.009437367340938366 deg, height=0.
↪009437367340938366 deg, angle=431.04359123537307 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.89561695, -9.37075174)>, width=0.00773387258312214 deg, height=0.
↪00773387258312214 deg, angle=280.1779909796026 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.67993575, -9.36243761)>, width=0.00854418282413368 deg, height=0.
↪00854418282413368 deg, angle=433.5141855894659 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.89314428, -9.3557064)>, width=0.00828300911798855 deg, height=0.
↪00828300911798855 deg, angle=287.59639446126425 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.57318109, -9.35301779)>, width=0.007552153247472566 deg, height=0.
↪007552153247472566 deg, angle=439.34160578877743 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.78984701, -9.34039916)>, width=0.00908259926340609 deg, height=0.
↪00908259926340609 deg, angle=348.28825039618556 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.80239825, -9.33163815)>, width=0.007737094243088075 deg, height=0.
↪007737094243088075 deg, angle=338.9892023029214 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.57572829, -9.31540772)>, width=0.0076731460209169145 deg, height=0.
↪0076731460209169145 deg, angle=429.2991901378363 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.83465169, -9.31090206)>, width=0.007815190929166629 deg, height=0.
↪007815190929166629 deg, angle=325.7609392262642 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.62179047, -9.30706564)>, width=0.00744031732502786 deg, height=0.
↪00744031732502786 deg, angle=421.5136736975769 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.93982357, -9.30419317)>, width=0.006817785524070047 deg, height=0.
↪006817785524070047 deg, angle=298.78277735769643 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.5824604, -9.2981542)>, width=0.007498418709017829 deg, height=0.
↪007498418709017829 deg, angle=424.3720915847924 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.61992022, -9.27890238)>, width=0.007409588481195356 deg, height=0.
↪007409588481195356 deg, angle=414.4219788807195 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.73873297, -9.24004532)>, width=0.006809609883438864 deg, height=0.
↪006809609883438864 deg, angle=374.75959111728724 deg)>,
```

(continues on next page)

(continued from previous page)

```
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.84875786, -9.2334555)>, width=0.007624181442133438 deg, height=0.
↪007624181442133438 deg, angle=336.4766373980855 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.70002037, -9.22635466)>, width=0.007312596562708717 deg, height=0.
↪007312596562708717 deg, angle=385.32690819492217 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.73053419, -9.21549118)>, width=0.007092895516765346 deg, height=0.
↪007092895516765346 deg, angle=375.2559289849012 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.66945038, -9.21538118)>, width=0.007507169766322179 deg, height=0.
↪007507169766322179 deg, angle=391.60689657572266 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.92113086, -9.51503501)>, width=0.00920233061969241 deg, height=0.
↪00920233061969241 deg, angle=588.5428276309194 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.76718805, -9.46938864)>, width=0.008479874183753903 deg, height=0.
↪008479874183753903 deg, angle=531.4119478557901 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.97067115, -9.44025128)>, width=0.007134318556827573 deg, height=0.
↪007134318556827573 deg, angle=255.47074364670326 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.60991084, -9.35741364)>, width=0.007591343401669003 deg, height=0.
↪007591343401669003 deg, angle=438.52061221514987 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.58434174, -9.35144598)>, width=0.007715982802198556 deg, height=0.
↪007715982802198556 deg, angle=438.29078458752 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.75127407, -9.32831492)>, width=0.007877356155183543 deg, height=0.
↪007877356155183543 deg, angle=383.5305446771204 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.83988442, -9.24448306)>, width=0.0074608492348664825 deg, height=0.
↪0074608492348664825 deg, angle=337.7900853561976 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.67936205, -9.22831308)>, width=0.007642103323825737 deg, height=0.
↪007642103323825737 deg, angle=391.13527919824 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.62917513, -9.21923338)>, width=0.007642082016264236 deg, height=0.
↪007642082016264236 deg, angle=400.7084686717307 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.79871532, -9.16670201)>, width=0.012440372645364971 deg, height=0.
↪012440372645364971 deg, angle=355.0788032786662 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.84029561, -9.54883107)>, width=0.007667803827869568 deg, height=0.
↪007667803827869568 deg, angle=560.96001825782 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.8978813, -9.37772185)>, width=0.007862583932634504 deg, height=0.
↪007862583932634504 deg, angle=276.65062148184717 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.89105993, -9.54884994)>, width=0.011787833479783082 deg, height=0.
↪011787833479783082 deg, angle=575.0197345874648 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
```

(continues on next page)

(continued from previous page)

```
      (328.97061681, -9.43944536)>, width=0.006871217861284668 deg, height=0.
↪006871217861284668 deg, angle=255.69675250947418 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.74727506, -9.53826212)>, width=0.004185759438683856 deg, height=0.
↪003417651254745010 5 deg, angle=0.00519723029853467 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.65521757, -9.25972201)>, width=0.005085643146284783 deg, height=0.
↪004979068953513777 deg, angle=-89.97978092689743 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.74532898, -9.1401906)>, width=0.008910391116877896 deg, height=0.
↪005099599293395697 deg, angle=-4.3431752110019035 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.84740316, -9.12401742)>, width=0.010836210345668586 deg, height=0.
↪004752678895310275 deg, angle=22.76255845772804 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.66155125, -9.25468904)>, width=0.007461134604631991 deg, height=0.
↪006089759412612027 deg, angle=-243.26581444855768 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.87250749, -9.17123746)>, width=0.015571927865370996 deg, height=0.
↪012841259287035708 deg, angle=-5.447628043271801 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.76800879, -9.58453019)>, width=0.04900630168181078 deg, height=0.
↪041851841281045 4 deg, angle=14.492613927564252 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.73703671, -9.42752247)>, width=0.02737187371185334 deg, height=0.
↪025547049908862033 deg, angle=-234.62113208461741 deg)>],
 '0502671101': [<EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (235.90154368, -83.80147385)>, width=0.00660837290419497 deg, height=0.
↪00660837290419497 deg, angle=202.4868703807935 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (234.49541433, -83.68304058)>, width=0.007100641472299129 deg, height=0.
↪007100641472299129 deg, angle=157.52414219789802 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (233.74507431, -83.51250489)>, width=0.00714719294578614 deg, height=0.
↪00714719294578614 deg, angle=430.5717571135679 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (233.94244606, -83.35916674)>, width=0.007460790554446995 deg, height=0.
↪007460790554446995 deg, angle=390.6036279032619 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (234.20203946, -83.63612767)>, width=0.011149504631671156 deg, height=0.
↪011149504631671156 deg, angle=491.55066920006726 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (235.09173966, -83.3819441)>, width=0.007336634582089766 deg, height=0.
↪007336634582089766 deg, angle=354.7415736534235 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (235.27920679, -83.62139769)>, width=0.009230085090930708 deg, height=0.
↪009230085090930708 deg, angle=212.69728753428527 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (236.23212762, -83.57686191)>, width=0.007570885109952949 deg, height=0.
↪007570885109952949 deg, angle=263.42136755600364 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (232.78254537, -83.64354873)>, width=0.007287453342756499 deg, height=0.
↪007287453342756499 deg, angle=471.22423270980056 deg)>,
```

(continues on next page)

(continued from previous page)

```
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (236.73121362, -83.53946232)>, width=0.028088980637951882 deg, height=0.
↪015714807252284024 deg, angle=65.80502135791009 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (234.90593367, -83.588805)>, width=0.060431397862485636 deg, height=0.
↪05796591495634242 deg, angle=8.671039046651588 deg)>],
 '0761910201': [<EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.68937419, -9.56078069)>, width=0.007605173317941238 deg, height=0.
↪007605173317941238 deg, angle=512.5237110475313 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.77253784, -9.53257425)>, width=0.007225087744081948 deg, height=0.
↪007225087744081948 deg, angle=537.2291426911661 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.77991627, -9.52102755)>, width=0.007733317091148604 deg, height=0.
↪007733317091148604 deg, angle=540.1609388825366 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.88736665, -9.52040684)>, width=0.007642123949260695 deg, height=0.
↪007642123949260695 deg, angle=579.1564080819821 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.80523125, -9.51758771)>, width=0.007509971224357759 deg, height=0.
↪007509971224357759 deg, angle=551.2198086846786 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.72564001, -9.51836509)>, width=0.007901393639954415 deg, height=0.
↪007901393639954415 deg, angle=517.526794183721 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.92044482, -9.51495767)>, width=0.009964105840928699 deg, height=0.
↪009964105840928699 deg, angle=587.989011277084 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.67642934, -9.51242001)>, width=0.006911430923404150 5 deg, height=0.
↪006911430923404150 5 deg, angle=500.3228230353246 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.56531187, -9.4881104)>, width=0.007708270005386793 deg, height=0.
↪007708270005386793 deg, angle=474.9739522459599 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.55365585, -9.48913623)>, width=0.007266936661629105 6 deg, height=0.
↪007266936661629105 6 deg, angle=474.05785398408636 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.76595056, -9.47897744)>, width=0.007737084485437304 deg, height=0.
↪007737084485437304 deg, angle=531.4412174213251 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.82192403, -9.47346034)>, width=0.007405325670107783 deg, height=0.
↪007405325670107783 deg, angle=566.5750852172926 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.67191667, -9.44653819)>, width=0.008188997983536301 deg, height=0.
↪008188997983536301 deg, angle=478.0965592859854 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.74315096, -9.44429672)>, width=0.007494703478737374 deg, height=0.
↪007494703478737374 deg, angle=506.57593722899287 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.98165457, -9.43094931)>, width=0.008713355304176376 deg, height=0.
↪008713355304176376 deg, angle=258.3220246762259 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
```

(continues on next page)

(continued from previous page)

```
      (328.7608298, -9.41117408)>, width=0.008338247600854148 deg, height=0.
↪008338247600854148 deg, angle=499.03105288380016 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.70626465, -9.40879626)>, width=0.0077696729844267025 deg, height=0.
↪0077696729844267025 deg, angle=464.653955323828 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.84643592, -9.40421552)>, width=0.007794735186659312 deg, height=0.
↪007794735186659312 deg, angle=257.75908603777935 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.80710602, -9.39717254)>, width=0.0077371023425100985 deg, height=0.
↪0077371023425100985 deg, angle=255.05350280846199 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (329.00979931, -9.38984914)>, width=0.007642116597171322 deg, height=0.
↪007642116597171322 deg, angle=269.96043273636946 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.70467401, -9.38436828)>, width=0.006738410167270855 deg, height=0.
↪006738410167270855 deg, angle=445.7222148366446 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.83070165, -9.38367648)>, width=0.007874826013197438 deg, height=0.
↪007874826013197438 deg, angle=277.03565312549773 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.81200351, -9.38025547)>, width=0.007045742026326322 deg, height=0.
↪007045742026326322 deg, angle=286.77870377092853 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.67005046, -9.37491259)>, width=0.008233181457226636 deg, height=0.
↪008233181457226636 deg, angle=442.1198637685323 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.75649891, -9.37322249)>, width=0.007933552857170845 deg, height=0.
↪007933552857170845 deg, angle=413.71675948051967 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.70796622, -9.37173101)>, width=0.008681480850895254 deg, height=0.
↪008681480850895254 deg, angle=435.5766777103424 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.8957851, -9.37103292)>, width=0.007538171566229098 deg, height=0.
↪007538171566229098 deg, angle=279.32203458597206 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.72058903, -9.37144816)>, width=0.009346645864457591 deg, height=0.
↪009346645864457591 deg, angle=432.3986182648724 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.68086324, -9.36191652)>, width=0.007842746784627016 deg, height=0.
↪007842746784627016 deg, angle=433.9790996303439 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.89339805, -9.35562868)>, width=0.007894676140839866 deg, height=0.
↪007894676140839866 deg, angle=286.9454240409902 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.61028928, -9.35544602)>, width=0.0076466605615973645 deg, height=0.
↪0076466605615973645 deg, angle=438.3786140008967 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.57316187, -9.35288901)>, width=0.0074928486894210375 deg, height=0.
↪0074928486894210375 deg, angle=439.74467148930773 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (329.01076915, -9.35201347)>, width=0.009124096971773004 deg, height=0.
↪009124096971773004 deg, angle=279.37627450237886 deg)>,
```

(continues on next page)

(continued from previous page)

```
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.78943218, -9.33982115)>, width=0.009362956833258915 deg, height=0.
↪009362956833258915 deg, angle=348.97538633888524 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.85669407, -9.32949866)>, width=0.008796769907473507 deg, height=0.
↪008796769907473507 deg, angle=308.4194123975086 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.7503647, -9.32688732)>, width=0.007820733673050688 deg, height=0.
↪007820733673050688 deg, angle=384.55676029198094 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.57621029, -9.31464728)>, width=0.007424424945689914 deg, height=0.
↪007424424945689914 deg, angle=429.485174123743 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.83474633, -9.31090046)>, width=0.007501871457599387 deg, height=0.
↪007501871457599387 deg, angle=325.4059932279165 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.79077339, -9.31158361)>, width=0.00797343822778213 deg, height=0.
↪00797343822778213 deg, angle=351.94516751782294 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.93981651, -9.3040177)>, width=0.007445535674519624 deg, height=0.
↪007445535674519624 deg, angle=298.46985071857654 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.58249999, -9.29807621)>, width=0.0071450812701841235 deg, height=0.
↪0071450812701841235 deg, angle=424.7641479315843 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.6189924, -9.27921549)>, width=0.008684419291152426 deg, height=0.
↪008684419291152426 deg, angle=415.0951940564478 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.64197997, -9.23477804)>, width=0.0074608267523772415 deg, height=0.
↪0074608267523772415 deg, angle=401.22044355499946 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.84886845, -9.23328715)>, width=0.008116936777280776 deg, height=0.
↪008116936777280776 deg, angle=336.38868916703535 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.67951663, -9.22868351)>, width=0.007729466376465991 deg, height=0.
↪007729466376465991 deg, angle=391.50031933343274 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.70024959, -9.22699216)>, width=0.0076718807023934054 deg, height=0.
↪0076718807023934054 deg, angle=385.6729366953833 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.66815329, -9.21530287)>, width=0.007642091503343634 deg, height=0.
↪007642091503343634 deg, angle=392.2191732974662 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.6767295, -9.19053487)>, width=0.007045646840351006 deg, height=0.
↪007045646840351006 deg, angle=386.9967740636772 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.58244675, -9.4627311)>, width=0.0073499475291128385 deg, height=0.
↪0073499475291128385 deg, angle=470.57915661676134 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.77709921, -9.39113194)>, width=0.007642176657240857 deg, height=0.
↪007642176657240857 deg, angle=476.75139849239014 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
```

(continues on next page)

(continued from previous page)

```
      (328.68704345, -9.52917156)>, width=0.009760532586818882 deg, height=0.
↪009760532586818882 deg, angle=506.7890913118479 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.70787811, -9.50168701)>, width=0.01039442263290166 deg, height=0.
↪01039442263290166 deg, angle=507.7046920804661 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.76719065, -9.46949572)>, width=0.008898874358419878 deg, height=0.
↪008898874358419878 deg, angle=531.2970150964355 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.96751421, -9.44376535)>, width=0.0074926116789422875 deg, height=0.
↪0074926116789422875 deg, angle=253.76133173052233 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.87544149, -9.42650864)>, width=0.008883965127052801 deg, height=0.
↪008883965127052801 deg, angle=608.831353701835 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.87027321, -9.41502843)>, width=0.009208357013779252 deg, height=0.
↪009208357013779252 deg, angle=254.30719692583813 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.63355, -9.2672484)>, width=0.007570125451236267 deg, height=0.
↪007570125451236267 deg, angle=409.62181893037854 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.8818919, -9.13298913)>, width=0.007606100056132776 deg, height=0.
↪007606100056132776 deg, angle=338.51230130245455 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.8430574, -9.32774321)>, width=0.009316088407206 37 deg, height=0.
↪009316088407206 37 deg, angle=314.75663725655045 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.6213323, -9.3083263)>, width=0.007494668592118892 deg, height=0.
↪007494668592118892 deg, angle=422.44481229159595 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.58389853, -9.40170782)>, width=0.007762796464037069 deg, height=0.
↪007762796464037069 deg, angle=453.54391975536015 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.943744, -9.49940368)>, width=0.007266861168549896 deg, height=0.
↪007266861168549896 deg, angle=595.9032099359723 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.65523524, -9.25937826)>, width=0.0069789185302487015 deg, height=0.
↪006771908415006877 deg, angle=38.18458091605515 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.74495384, -9.13987952)>, width=0.010799658012802556 deg, height=0.
↪005065284194131101 deg, angle=-3.41899691429815 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.8472948, -9.12389621)>, width=0.011241188923289154 deg, height=0.
↪005825221156390184 deg, angle=22.931945913121808 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.66737543, -9.43260352)>, width=0.022303894441709094 deg, height=0.
↪011907153729425595 deg, angle=51.78980020234835 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.57712017, -9.51571182)>, width=0.014725846448283647 deg, height=0.
↪01185632766887667 deg, angle=69.67002567183334 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.75689362, -9.57390549)>, width=0.06516474075786996 deg, height=0.
↪03133683329802987 deg, angle=-12.004004917278055 deg)>,
```

(continues on next page)

(continued from previous page)

```
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.7342496, -9.42368759)>, width=0.03320522012875693 deg, height=0.
↪02804757347416249 3 deg, angle=54.06498950775162 deg)>,
  <EllipseSkyRegion(<SkyCoord (ICRS): (ra, dec) in deg
      (328.80555656, -9.33028177)>, width=0.017124914642677427 deg, height=0.
↪014535991053605816 deg, angle=-44.42214439125614 deg)>]},
 'chandra': {},
 'nustar_pointed': {},
 'rosat_all_sky': {}}
```

3.4 Processing Tutorials

3.4.1 Processing raw XMM-Newton Pointed data

This tutorial will teach you how to use DAXA to process raw XMM-Newton data into a science ready state using one line of Python code (or several lines, if you wish to have more control over the settings for each step). **This relies on there being an initialised (either manually before launching Python, or in your bash profile/rc) backend installation of the XMM Science Analysis System (SAS), including accessible calibration files** - DAXA will check for such an installation, and will not allow processing to start without it.

All DAXA processing steps will parallelise as much as possible - processes running on different ObsIDs/instruments/sub-exposures will be run simultaneously (if cores are available)

Import Statements

[29]:
```
from daxa.mission import XMMPointed
from daxa.archive import Archive
from daxa.process.simple import full_process_xmm
from daxa.process.xmm.setup import cif_build, odf_ingest
from daxa.process.xmm.assemble import emchain, epchain, cleaned_evt_lists, merge_
↪subexposures, \
    rgs_events, rgs_angles, cleaned_rgs_event_lists
from daxa.process.xmm.check import emanom
from daxa.process.xmm.clean import espfilt
from daxa.process.xmm.generate import generate_images_expmaps
```

An Archive to be processed

Every processing function implemented in DAXA takes an Archive instance as its first argument; if you don't already know what that is then you should go back and read the following tutorials:

- Creating a DAXA archive - This explains how to create an archive, load an existing archive, and the various properties and features of DAXA archives.
- Using DAXA missions - Here we explain what DAXA mission classes are and how to use them to select only the data you need.

Here we create an archive of XMM observations of the galaxy cluster Abell 907:

[5]:
```
xm = XMMPointed()
xm.filter_on_name('A907')
arch = Archive('Abell907', xm)
```

```
/Users/dt237/code/DAXA/daxa/mission/xmm.py:83: UserWarning: 140 of the 17689␣
↪observations located for this mission have been removed due to NaN RA or Dec values
  self._fetch_obs_info()
Downloading XMM-Newton Pointed data: 100%|| 3/3 [00:36<00:00, 12.05s/it]
```

One-line solution

Though we provide individual functions that wrap the various steps required to reduce and prepare XMM data, and they can be used separately for greater control over the configuration parameters, we also include a one-line solution which executes the processing steps with default configuration.

We believe that the default parameters are adequate for most use cases, and this allows for users unfamiliar with the intricacies of XMM data to easily start working with it. Executing the following will automatically generate cleaned event lists for MOS1, MOS2, PN, RGS1, and RGS2 (if they were selected during mission creation), as well as images and exposure maps for MOS1, MOS2, and PN:

[6]:
```
full_process_xmm(arch)
```

```
XMM-Newton Pointed - Generating calibration files: 100%|| 3/3 [00:19<00:00,  6.51s/it]
XMM-Newton Pointed - Generating ODF summary files: 100%|| 3/3 [00:03<00:00,  1.25s/it]
XMM-Newton Pointed - Assembling PN and PN-OOT event lists: 100%|| 3/3 [03:37<00:00, 72.
↪36s/it]
XMM-Newton Pointed - Assembling MOS event lists: 100%|| 6/6 [01:22<00:00, 13.70s/it]
XMM-Newton Pointed - Finding PN/MOS soft-proton flares: 100%|| 9/9 [00:17<00:00,  1.93s/
↪it]
XMM-Newton Pointed - Generating cleaned PN/MOS event lists: 100%|| 8/8 [00:03<00:00,  2.
↪62it/s]
XMM-Newton Pointed - Generating final PN/MOS event lists: 100%|| 8/8 [00:00<00:00, 120.
↪67it/s]
Generating products of type(s) ccf: 100%|| 3/3 [00:19<00:00,  6.37s/it]
Generating products of type(s) image: 100%|| 8/8 [00:01<00:00,  7.19it/s]
Generating products of type(s) expmap: 100%|| 8/8 [00:22<00:00,  2.84s/it]
Generating products of type(s) image: 100%|| 8/8 [00:01<00:00,  7.01it/s]
Generating products of type(s) expmap: 100%|| 8/8 [00:22<00:00,  2.84s/it]
```

General arguments for all processing functions

There are some arguments that all processing functions will take - they don't control the behaviour of the processing step itself, but instead how the commands are executed:

- **num_cores** - The number of cores that the function is allowed to use. This is set globally by the daxa.NUM_CORES constant (if set before any other DAXA function is imported), and by default is 90% of the cores available on the system.
- **timeout** - This controls how long an individual instance of the process is allowed to run before it is cancelled; the whole function may run for longer depending how many pieces of data need processing and how many cores are allocated. The default is generally null, but it can be set using an astropy quantity with time units.

Breaking down the XMM processing steps

Setup steps

These functions set up the necessary environment/files for the further processing/reduction of XMM data.

Building calibration files (cif_build)

The cif_build function function is a wrapper for the XMM SAS tool of the same name - it assembles a CIF, which points other SAS tools to the current calibration files required for whatever XMM instrument they are working on.

Only the `analysis_date` parameter of this function affects the files produced by this function (other things can be controlled, such as the number of cores or timeout) - this date determines which calibration files are selected for the generated CIF, and the default is the current date - another date can be passed as a Python datetime object.

All other processing steps depend on this one

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

```
Help on function cif_build in module daxa.process.xmm.setup:

cif_build(obs_archive: daxa.archive.base.Archive, num_cores: int = 9, disable_progress:␣
↪bool = False, analysis_date: Union[str, datetime.datetime] = 'now', timeout: astropy.
↪units.quantity.Quantity = None) -> Tuple[dict, dict, dict, str, int, bool, astropy.
↪units.quantity.Quantity]
    A DAXA Python interface for the SAS cifbuild command, used to generate calibration␣
↪files for XMM observations
    prior to processing. The observation date is supplied by the XMM mission instance(s),
↪ and is the date when the
    observation was started (as acquired from the XSA).

    :param Archive obs_archive: An Archive instance containing XMM mission instances for␣
↪which observation calibration
        files should be generated. This function will fail if no XMM missions are␣
↪present in the archive.
    :param int num_cores: The number of cores to use, default is set to 90% of available.
    :param bool disable_progress: Setting this to true will turn off the SAS generation␣
↪progress bar.
    :param str/datetime analysis_date: The analysis date for which to generate␣
↪calibration file. The default is
        'now', but this parameter can be used to create calibration files as they would␣
↪have been on a past date.
    :param Quantity timeout: The amount of time each individual process is allowed to␣
↪run for, the default is None.
        Please note that this is not a timeout for the entire cif_build process, but a␣
↪timeout for individual
        ObsID processes.
    :return: Information required by the SAS decorator that will run commands. Top level␣
↪keys of any dictionaries are
        internal DAXA mission names, next level keys are ObsIDs. The return is a tuple␣
↪containing a) a dictionary of
        bash commands, b) a dictionary of final output paths to check, c) a dictionary␣
↪of extra info (in this case
        obs and analysis dates), d) a generation message for the progress bar, e) the␣
↪number of cores allowed, and
```

(continues on next page)

(continued from previous page)

```
        f) whether the progress bar should be hidden or not.
    :rtype: Tuple[dict, dict, dict, str, int, bool, Quantity]
```

Summarising the available data files (odf_ingest)

The odf_ingest function function simply examines the observation data file (ODF) directory, and determines the instruments (including observing modes and sub-exposures) that have data present - it then creates a SAS summary file.

Our implementation of this function wraps the original SAS tool, and then adds a second step - this parses the SAS summary file and extracts the information that is relevant to whether we can use a particular instrument (and sub-exposure of an instrument) for astrophysics. This is what populates the XMM entry in the `observation_summaries` property of the archive class.

All subsequent steps depend on this one - and observation_summaries will have no XMM entry until this is run

[34]:
```
help(odf_ingest)
```

```
Help on function odf_ingest in module daxa.process.xmm.setup:

odf_ingest(obs_archive: daxa.archive.base.Archive, num_cores: int = 9, disable_progress:␣
↪bool = False, timeout: astropy.units.quantity.Quantity = None)
    This function runs the SAS odfingest task, which creates a summary of the raw data␣
↪available in the ODF
    directory, and is used by many SAS processing tasks.

    :param Archive obs_archive: An Archive instance containing XMM mission instances for␣
↪which observation summary
        files should be generated. This function will fail if no XMM missions are␣
↪present in the archive.
    :param int num_cores: The number of cores to use, default is set to 90% of available.
    :param bool disable_progress: Setting this to true will turn off the SAS generation␣
↪progress bar.
    :param Quantity timeout: The amount of time each individual process is allowed to␣
↪run for, the default is None.
        Please note that this is not a timeout for the entire odf_ingest process, but a␣
↪timeout for individual
        ObsID processes.
    :return: Information required by the SAS decorator that will run commands. Top level␣
↪keys of any dictionaries are
        internal DAXA mission names, next level keys are ObsIDs. The return is a tuple␣
↪containing a) a dictionary of
        bash commands, b) a dictionary of final output paths to check, c) a dictionary␣
↪of extra info (in this case
        obs and analysis dates), d) a generation message for the progress bar, e) the␣
↪number of cores allowed, and
        f) whether the progress bar should be hidden or not.
    :rtype: Tuple[dict, dict, dict, str, int, bool, Quantity]
```

EPIC Cameras (CCD)

These functions are specifically for the preparation of data from the three EPIC cameras on XMM; PN, MOS1, and MOS2.

Assembling whole-camera MOS initial event lists (emchain)

The emchain function is what assembles the raw, separate-CCD-level, event lists and files into a single raw events list for a whole XMM MOS camera exposure. If there are **multiple sub-exposures** which DAXA has identified as being usable, then a separate raw event list is created for each of them - they are combined in a later processing step.

We have implemented this method so that MOS1 and MOS2 (and sub-exposures, if present) data are processed in parallel for a given observation (and all observations are processed in parallel as well).

Only the `process_unscheduled` argument passed to emchain can change the files produced, as it controls whether unscheduled sub-exposures should be processed or not - the default is True.

[35]:
```
help(emchain)
```

```
Help on function emchain in module daxa.process.xmm.assemble:

emchain(obs_archive: daxa.archive.base.Archive, process_unscheduled: bool = True, num_
↪cores: int = 9, disable_progress: bool = False, timeout: astropy.units.quantity.
↪Quantity = None)
    This function runs the emchain SAS process on XMM missions in the passed archive,␣
↪which assembles the
    MOS-specific ODFs into combined photon event lists - rather than the per CCD files␣
↪that existed before. The
    emchain manual can be found here (https://xmm-tools.cosmos.esa.int/external/sas/
↪current/doc/emchain.pdf) and
    gives detailed explanations of the process.

    The DAXA wrapper does not allow emchain to automatically loop through all the sub-
↪exposures for a given
    ObsID-MOSX combo, but rather creates a separate process call for each of them. This␣
↪allows for greater
    parallelisation (if on a system with a significant core count), but also allows the␣
↪same level of granularity
    in the logging of processing of different sub-exposures as in DAXA's epchain␣
↪implementation.

    The particular CCDs to be processed are not specified in emchain, unlike in epchain,␣
↪because it can sometimes
    have unintended consequences. For instance processing a MOS observation in␣
↪FastUncompressed mode, with timing
    on CCD 1 and imaging everywhere else, can cause emchain to fail (even though no␣
↪actual failure occurs) because
    the submode is set to Unknown, rather than FastUncompressed.

    :param Archive obs_archive: An Archive instance containing XMM mission instances␣
↪with MOS observations for
        which emchain should be run. This function will fail if no XMM missions are␣
↪present in the archive.
    :param bool process_unscheduled: Whether this function should also process sub-
↪exposures marked 'U', for
```

(continues on next page)

(continued from previous page)

```
        unscheduled. Default is True, in which case they will be processed.
    :param int num_cores: The number of cores to use, default is set to 90% of available.
    :param bool disable_progress: Setting this to true will turn off the SAS generation␣
↪progress bar.
    :param Quantity timeout: The amount of time each individual process is allowed to␣
↪run for, the default is None.
        Please note that this is not a timeout for the entire emchain process, but a␣
↪timeout for individual
        ObsID-subexposure processes.
    :return: Information required by the SAS decorator that will run commands. Top level␣
↪keys of any dictionaries are
        internal DAXA mission names, next level keys are ObsIDs. The return is a tuple␣
↪containing a) a dictionary of
        bash commands, b) a dictionary of final output paths to check, c) a dictionary␣
↪of extra info (in this case
        obs and analysis dates), d) a generation message for the progress bar, e) the␣
↪number of cores allowed, and
        f) whether the progress bar should be hidden or not.
    :rtype: Tuple[dict, dict, dict, str, int, bool, Quantity]
```

Assembling whole-camera PN initial event lists (epchain)

The epchain function serves much the same purpose as emchain, but for PN camera data. There is only one PN camera, but the presence of multiple sub-exposures is still possible. This function generates raw, whole-camera, event lists just like emchain, but also creates whole-camera out-of-time (OOT) event lists, which attempts to identify event lists that were detected during a CCD readout (this can leave a characteristic streak in the direction of readout for bright sources in PN observations).

[36]:
```
help(epchain)
```

```
Help on function epchain in module daxa.process.xmm.assemble:

epchain(obs_archive: daxa.archive.base.Archive, process_unscheduled: bool = True, num_
↪cores: int = 9, disable_progress: bool = False, timeout: astropy.units.quantity.
↪Quantity = None)
    This function runs the epchain SAS process on XMM missions in the passed archive,␣
↪which assembles the
    PN-specific ODFs into combined photon event lists - rather than the per CCD files␣
↪that existed before. A run of
    epchain for out of time (OOT) events is also performed as part of this function call.
↪ The epchain manual can be
    found here (https://xmm-tools.cosmos.esa.int/external/sas/current/doc/epchain.pdf)␣
↪and gives detailed
    explanations of the process.

    Per the advice of the SAS epchain manual, the OOT event list epchain call is␣
↪performed first, and its intermediate
    files are saved and then used for the normal call to epchain.

    :param Archive obs_archive: An Archive instance containing XMM mission instances␣
↪with PN observations for
```

(continues on next page)

(continued from previous page)

```
        which epchain should be run. This function will fail if no XMM missions are␣
↪present in the archive.
    :param bool process_unscheduled: Whether this function should also process sub-
↪exposures marked 'U', for
        unscheduled. Default is True, in which case they will be processed.
    :param int num_cores: The number of cores to use, default is set to 90% of available.
    :param bool disable_progress: Setting this to true will turn off the SAS generation␣
↪progress bar.
    :param Quantity timeout: The amount of time each individual process is allowed to␣
↪run for, the default is None.
        Please note that this is not a timeout for the entire epchain process, but a␣
↪timeout for individual
        ObsID-subexposure processes.
    :return: Information required by the SAS decorator that will run commands. Top level␣
↪keys of any dictionaries are
        internal DAXA mission names, next level keys are ObsIDs. The return is a tuple␣
↪containing a) a dictionary of
        bash commands, b) a dictionary of final output paths to check, c) a dictionary␣
↪of extra info (in this case
        obs and analysis dates), d) a generation message for the progress bar, e) the␣
↪number of cores allowed, and
        f) whether the progress bar should be hidden or not.
    :rtype: Tuple[dict, dict, dict, str, int, bool, Quantity]
```

Searching for anomalous MOS CCD states (emanom)

The emanom function wraps the SAS function of the same name, and is an **optional** step that finds MOS CCDs which have operated in an 'anomalous' state, where the background at E < 1 keV is strongly enhanced. However, it should be noted that the "anonymous" anomalous state of MOS1 CCD#4 is not always detectable from the unexposed corner data.

We note that this is not enabled by default in the ``full_process_xmm`` function, and your results may vary.

[37]:
```
help(emanom)
```

```
Help on function emanom in module daxa.process.xmm.check:

emanom(obs_archive: daxa.archive.base.Archive, num_cores: int = 9, disable_progress:␣
↪bool = False, timeout: astropy.units.quantity.Quantity = None)
    This function runs the SAS emanom function, which attempts to identify when MOS CCDs␣
↪are have operated in an
    'anomalous' state, where the  background at E < 1 keV is strongly enhanced. Data␣
↪above 2 keV are unaffected, so
    CCDs in anomalous states used for science where the soft X-rays are unnecessary do␣
↪not need to be excluded.

    The emanom task calculates the (2.5-5.0 keV)/(0.4-0.8 keV) hardness ratio from the␣
↪corner data to determine
    whether a chip is in an anomalous state. However, it should be noted that the
↪"anonymous" anomalous state of
    MOS1 CCD#4 is not always detectable from the unexposed corner data.
```

(continues on next page)

(continued from previous page)

```
    This functionality is only usable if you have SAS v19.0.0 or higher - a version␣
↪check will be performed and
    a warning raised (though no error will be raised) if you use this function with an␣
↪earlier SAS version.

    :param Archive obs_archive: An Archive instance containing XMM mission instances␣
↪with MOS observations for
        which emchain should be run. This function will fail if no XMM missions are␣
↪present in the archive.
    :param int num_cores: The number of cores to use, default is set to 90% of available.
    :param bool disable_progress: Setting this to true will turn off the SAS generation␣
↪progress bar.
    :param Quantity timeout: The amount of time each individual process is allowed to␣
↪run for, the default is None.
        Please note that this is not a timeout for the entire emanom process, but a␣
↪timeout for individual
        ObsID-subexposure processes.
    :return: Information required by the SAS decorator that will run commands. Top level␣
↪keys of any dictionaries are
        internal DAXA mission names, next level keys are ObsIDs. The return is a tuple␣
↪containing a) a dictionary of
        bash commands, b) a dictionary of final output paths to check, c) a dictionary␣
↪of extra info (in this case
        obs and analysis dates), d) a generation message for the progress bar, e) the␣
↪number of cores allowed, and
        f) whether the progress bar should be hidden or not.
    :rtype: Tuple[dict, dict, dict, str, int, bool, Quantity]
```

Searching for flared periods of the observations (espfilt)

This particularly important function (espfilt) wraps the SAS tool of the same name, and attempts to automatically detect periods of observations which have been overwhelmed by soft-proton flaring. The ultimate result of this process is a set of 'good time intervals', which are periods of a particular observation that have been deemed safe to use; **this check if performed on all instruments and sub-exposures of an observation independently**.

Several configuration options are provided, mirroring those that can be passed into the espfilt SAS tool:

- **method** - There are two methods of identifying flaring available, 'histogram' (the default) and 'ratio'. The histogram method fits a gaussian to the whole-FoV count-rate distribution (in order to find a mean and a standard deviation), and then defines good-time intervals by finding the time periods where the whole-FoV lightcurve count-rate is within `allowed_sigma` of the mean. The 'ratio' method compares the count-rate from the FoV with the count-rate in the unexposed corners of XMM observations (i.e. there is an active CCD but the telescope is not focusing photons or soft-protons onto it); an acceptable ratio is defined by `ratio`, and points where the lightcurve ratio is lower than or equal to this are considered good-time intervals.
- **with_smoothing** - Should smoothing be applied to the light curve data. The default is True, in which case a smoothing factor of 51 seconds is used - an astropy quantity with time units may also be passed to set a custom value.
- **with_binning** - Should binning be applied to the light curve data. The default is True, in which case a binning size of 60 seconds is used - an astropy quantity with time units may also be passed to set a custom value.

- **ratio** - The acceptable ratio for the 'ratio' method.
- **filter_lo_en** - The lower energy bound used for soft-proton flaring identification; the default is 2.5 keV.
- **filter_hi_en** - The upper energy bound used for soft-proton flaring identification; the default is 8.5 keV.
- **range_scale** - Histogram fit range scale factor. The default is a dictionary with an entry for 'pn' (15.0) and an entry for 'mos' (6.0).
- **allowed_sigma** - Number of stdev from the mean count-rate allowed before a period is considered to **not** be a good time interval. Default is 3.
- **gauss_fit_lims** - The parameter limts for gaussian fits for the histogram method, the default is (0.1, 6.5).

[38]:
```
help(espfilt)
```

```
Help on function espfilt in module daxa.process.xmm.clean:

espfilt(obs_archive: daxa.archive.base.Archive, method: str = 'histogram', with_
↪smoothing: Union[bool, astropy.units.quantity.Quantity] = True, with_binning:␣
↪Union[bool, astropy.units.quantity.Quantity] = True, ratio: float = 1.2, filter_lo_en:␣
↪astropy.units.quantity.Quantity = <Quantity 2500. eV>, filter_hi_en: astropy.units.
↪quantity.Quantity = <Quantity 8500. eV>, range_scale: dict = None, allowed_sigma:␣
↪float = 3.0, gauss_fit_lims: Tuple[float, float] = (0.1, 6.5), num_cores: int = 9,␣
↪disable_progress: bool = False, timeout: astropy.units.quantity.Quantity = None)
    The DAXA wrapper for the XMM SAS task espfilt, which attempts to identify good time␣
↪intervals with minimal
    soft-proton flaring for individual sub-exposures (if multiple have been taken) of␣
↪XMM ObsID-Instrument
    combinations. Both EPIC-PN and EPIC-MOS observations will be processed by this␣
↪function.

    This function does not generate final event lists, but instead is used to create␣
↪good-time-interval files
    which are then applied to the creation of final event lists, along with other user-
↪specified filters, in the
    'cleaned_evt_lists' function.

    :param Archive obs_archive: An Archive instance containing XMM mission instances␣
↪with PN/MOS observations for
        which espfilt should be run. This function will fail if no XMM missions are␣
↪present in the archive.
    :param str method: The method that espfilt should use to find soft proton flaring.␣
↪Either 'ratio' or 'histogram'
        can be selected. The default is 'histogram'.
    :param bool/Quantity with_smoothing: Should smoothing be applied to the light curve␣
↪data. If set to True (the
        default) a smoothing factor of 51 seconds is used, if set to False smoothing␣
↪will be turned off, if an astropy
        Quantity is passed (with units convertible to seconds) then that value will be␣
↪used for the smoothing factor.
    :param bool/Quantity with_binning: Should binning be applied to the light curve data.
↪ If set to True (the
        default) a bin size of 60 seconds is used, if set to False binning will be␣
↪turned off, if an astropy
        Quantity is passed (with units convertible to seconds) then that value will be␣
↪used for the bin size.
```

(continues on next page)

(continued from previous page)

```
    :param float ratio: Flaring ratio of annulus counts.
    :param Quantity filter_lo_en: The lower energy bound for the event lists used for␣
↪soft proton flaring
        identification.
    :param Quantity filter_hi_en: The upper energy bound for the event lists used for␣
↪soft proton flaring
        identification.
    :param dict range_scale: Histogram fit range scale factor. The default is a␣
↪dictionary with an entry for 'pn'
        (15.0) and an entry for 'mos' (6.0).
    :param float allowed_sigma: Limit in sigma for unflared rates.
    :param Tuple[float, float] gauss_fit_lims: The parameter limits for gaussian fits.
    :param int num_cores: The number of cores to use, default is set to 90% of available.
    :param bool disable_progress: Setting this to true will turn off the SAS generation␣
↪progress bar.
    :param Quantity timeout: The amount of time each individual process is allowed to␣
↪run for, the default is None.
        Please note that this is not a timeout for the entire espfilt process, but a␣
↪timeout for individual
        ObsID-Inst-subexposure processes.
    :return: Information required by the SAS decorator that will run commands. Top level␣
↪keys of any dictionaries are
        internal DAXA mission names, next level keys are ObsIDs. The return is a tuple␣
↪containing a) a dictionary of
        bash commands, b) a dictionary of final output paths to check, c) a dictionary␣
↪of extra info (in this case
        obs and analysis dates), d) a generation message for the progress bar, e) the␣
↪number of cores allowed, and
        f) whether the progress bar should be hidden or not.
    :rtype: Tuple[dict, dict, dict, str, int, bool, Quantity]
```

Applying event filtering and good-time-intervals to the raw event lists (cleaned_evt_lists)

The cleaned_evt_lists function uses SAS' evselect tool to both apply the GTIs generated by the espfilt function, and to apply further filtering of events as directed by the user.

We allow for filtering based on flag, pattern, energy, and MOS anomolous states (if the emanon task was run) - the user can also define custom filtering expressions. The MOS and PN filtering statements are defined separately, as it is standard practise to use different filtering for the different instruments. The exact behaviour is controlled by passing the following arguments:

- **lo_en** - The lowest energy of event that should be left in the output cleaned event list. The default is None, in which case no energy filter is applied.
- **hi_en** - The highest energy of event that should be left in the output cleaned event list. The default is None, in which case no energy filter is applied.
- **pn_filt_expr** - This essentially lets the user set the evselect expression keyword, and determines what filters should be applied to the PN event lists (other than energy range and good time interval application) - the default is ("#XMMEA_EP", "(PATTERN <= 4)", "(FLAG .eq. 0)"), where the elements of the tuple are combined into the base filtering expression.

- **mos_filt_expr** - This essentially lets the user set the evselect expression keyword, and determines what filters should be applied to the MOS event lists (other than energy range and good time interval application) - the default is ("#XMMEA_EM", "(PATTERN <= 12)", "(FLAG .eq. 0)"), where the elements of the tuple are combined into the base filtering expression.
- **filt_mos_anom_state** - Whether the function should use the results of an 'emanom' run to identify and remove MOS CCDs that are in anomolous states. Default is 'False', meaning no such filtering would be applied.
- **acc_mos_anom_states** - A list/tuple of acceptable MOS CCD status codes found by emanom (status- G is good at all energies, I is intermediate for E<1 keV, B is bad for E<1 keV, O is off, chip not in use, U is undetermined (low band counts <= 0)).

[39]:
```
help(cleaned_evt_lists)
```

```
Help on function cleaned_evt_lists in module daxa.process.xmm.assemble:

cleaned_evt_lists(obs_archive: daxa.archive.base.Archive, lo_en: astropy.units.quantity.
↪Quantity = None, hi_en: astropy.units.quantity.Quantity = None, pn_filt_expr:␣
↪Union[str, List[str]] = ('#XMMEA_EP', '(PATTERN <= 4)', '(FLAG .eq. 0)'), mos_filt_
↪expr: Union[str, List[str]] = ('#XMMEA_EM', '(PATTERN <= 12)', '(FLAG .eq. 0)'), filt_
↪mos_anom_state: bool = False, acc_mos_anom_states: Union[List[str], str] = ('G', 'I',
↪'U'), num_cores: int = 9, disable_progress: bool = False, timeout: astropy.units.
↪quantity.Quantity = None)
    This function is used to apply the soft-proton filtering (along with any other␣
↪filtering you may desire, including
    the setting of energy limits) to XMM-Newton event lists, resulting in the creation␣
↪of sets of cleaned event lists
    which are ready to be analysed (or merged together, if there are multiple exposures␣
↪for a particular
    observation-instrument combination).

    :param Archive obs_archive: An Archive instance containing XMM mission instances for␣
↪which cleaned event lists
        should be created. This function will fail if no XMM missions are present in the␣
↪archive.
    :param Quantity lo_en: The lower bound of an energy filter to be applied to the␣
↪cleaned, filtered, event lists. If
        'lo_en' is set to an Astropy Quantity, then 'hi_en' must be as well. Default is␣
↪None, in which case no
        energy filter is applied.
    :param Quantity hi_en: The upper bound of an energy filter to be applied to the␣
↪cleaned, filtered, event lists. If
        'hi_en' is set to an Astropy Quantity, then 'lo_en' must be as well. Default is␣
↪None, in which case no
        energy filter is applied.
    :param str/List[str]/Tuple[str] pn_filt_expr: The filter expressions to be applied␣
↪to EPIC-PN event lists. Either
        a single string expression can be passed, or a list/tuple of separate␣
↪expressions, which will be combined
        using '&&' logic before being used as the expression for evselect. Other␣
↪expression components can be
        added during the process of the function, such as GTI filtering and energy␣
↪filtering.
    :param str/List[str]/Tuple[str] mos_filt_expr: The filter expressions to be applied␣
↪to EPIC-MOS event lists. Either
```

(continues on next page)

(continued from previous page)

```
        a single string expression can be passed, or a list/tuple of separate␣
↪expressions, which will be combined
        using '&&' logic before being used as the expression for evselect. Other␣
↪expression components can be
        added during the process of the function, such as GTI filtering, energy␣
↪filtering, and anomalous state CCD
        filtering..
    :param bool filt_mos_anom_state: Whether this function should use the results of an
↪'emanom' run
        to identify and remove MOS CCDs that are in anomolous states. If 'False' is␣
↪passed then no such filtering
        will be applied.
    :param List[str]/str acc_mos_anom_states: A list/tuple of acceptable MOS CCD status␣
↪codes found by emanom
        (status- G is good at all energies, I is intermediate for E<1 keV, B is bad for E
↪<1 keV, O is off, chip
        not in use, U is undetermined (low band counts <= 0)).
    :param int num_cores: The number of cores to use, default is set to 90% of available.
    :param bool disable_progress: Setting this to true will turn off the SAS generation␣
↪progress bar.
    :param Quantity timeout: The amount of time each individual process is allowed to␣
↪run for, the default is None.
        Please note that this is not a timeout for the entire cleaned_evt_lists process,␣
↪but a timeout for individual
        ObsID-Inst-subexposure processes.
    :return: Information required by the SAS decorator that will run commands. Top level␣
↪keys of any dictionaries are
        internal DAXA mission names, next level keys are ObsIDs. The return is a tuple␣
↪containing a) a dictionary of
        bash commands, b) a dictionary of final output paths to check, c) a dictionary␣
↪of extra info (in this case
        obs and analysis dates), d) a generation message for the progress bar, e) the␣
↪number of cores allowed, and
        f) whether the progress bar should be hidden or not.
    :rtype: Tuple[dict, dict, dict, str, int, bool, Quantity]
```

Finalising the event lists by combining sub-exposures (merge_subexposures)

The merge_subexposures function will automatically check for cases where a particular instrument of a particular ObsID has multiple sub-exposures, and then it will automatically combine them. There are no user-provided arguments which will affect the output of this function, only the standard arguments to set number of cores and timeout.

[40]:
```
help(merge_subexposures)
```

```
Help on function merge_subexposures in module daxa.process.xmm.assemble:

merge_subexposures(obs_archive: daxa.archive.base.Archive, num_cores: int = 9, disable_
↪progress: bool = False, timeout: astropy.units.quantity.Quantity = None)
    A function to identify cases where an instrument for a particular XMM observation␣
↪has multiple
```

(continues on next page)

(continued from previous page)

```
    sub-exposures, for which the event lists can be merged. This produces a final event␣
↪list, which is a
    combination of the sub-exposures.

    For those observation-instrument combinations with only a single
    exposure, this function will rename the cleaned event list so that the naming␣
↪convention is comparable
    to the merged event list naming convention (i.e. sub-exposure identifier will be␣
↪removed).

    :param Archive obs_archive: An Archive instance containing XMM mission instances for␣
↪which cleaned event lists
        should be created. This function will fail if no XMM missions are present in the␣
↪archive.
    :param int num_cores: The number of cores to use, default is set to 90% of available.
    :param bool disable_progress: Setting this to true will turn off the SAS generation␣
↪progress bar.
    :param Quantity timeout: The amount of time each individual process is allowed to␣
↪run for, the default is None.
        Please note that this is not a timeout for the entire merge_subexposures process,
↪ but a timeout for individual
        ObsID-Inst processes.
    :return: Information required by the SAS decorator that will run commands. Top level␣
↪keys of any dictionaries are
        internal DAXA mission names, next level keys are ObsIDs. The return is a tuple␣
↪containing a) a dictionary of
        bash commands, b) a dictionary of final output paths to check, c) a dictionary␣
↪of extra info (in this case
        obs and analysis dates), d) a generation message for the progress bar, e) the␣
↪number of cores allowed, and
        f) whether the progress bar should be hidden or not.
    :rtype: Tuple[dict, dict, dict, str, int, bool, Quantity]
```

Generating images and exposure maps (generate_images_expmaps)

This function (generate_images_expmaps) uses XGA (another module in our open source X-ray astrophysics ecosystem) to generate images and exposure maps for the archive that has been created, from the final, cleaned and merged, event lists produced by `merge_subexposures`. The user can control the upper and lower energy bounds for the images and exposure maps by passing astropy quantities in keV - the default setup creates images and exposure maps in the 0.5-2.0 keV and 2.0-10.0 keV bands.

[41]:
```
help(generate_images_expmaps)
```

```
Help on function generate_images_expmaps in module daxa.process.xmm.generate:

generate_images_expmaps(obs_archive: daxa.archive.base.Archive, lo_en: astropy.units.
↪quantity.Quantity = <Quantity [0.5, 2. ] keV>, hi_en: astropy.units.quantity.Quantity␣
↪= <Quantity [ 2., 10.] keV>, num_cores: int = 9)
    A function to generate images and exposure maps for a processed XMM mission dataset␣
↪contained within an
```

(continues on next page)

(continued from previous page)

```
    archive. Users can select the energy band(s) that they wish to generate images and␣
↪exposure maps within.

    :param Archive obs_archive:
    :param lo_en: The lower energy bound(s) for the product being generated. This can␣
↪either be passed as a
        scalar Astropy Quantity or, if sets of the same product in different energy␣
↪bands are to be generated, as a
        non-scalar Astropy Quantity. If multiple lower bounds are passed, they must each␣
↪have an entry in the
        hi_en argument. The default is 'Quantity([0.5, 2.0], 'keV')', which will␣
↪generate two sets of products, one
        with lower bound 0.5 keV, the other with lower bound 2 keV.
    :param hi_en: The upper energy bound(s) for the product being generated. This can␣
↪either be passed as a
        scalar Astropy Quantity or, if sets of the same product in different energy␣
↪bands are to be generated, as a
        non-scalar Astropy Quantity. If multiple upper bounds are passed, they must each␣
↪have an entry in the
        lo_en argument. The default is 'Quantity([2.0, 10.0], 'keV')', which will␣
↪generate two sets of products, one
        with upper bound 2.0 keV, the other with upper bound 10 keV.
    :param int num_cores:
```

Reflection Grating Spectrometer (RGS)

These functions prepare data for the two RGS instruments on XMM.

Preparing the initial raw RGS event lists (rgs_events)

The rgs_events function creates the initial raw event lists by calibrating and combining the separate CCD outputs in RGS event lists. As with the analogous EPIC camera functions, epchain and emchain, this happens separately for the two RGS instruments for each observation, and separately for each sub-exposure of the RGS instruments. There are no user-passed arguments that will control the output of this step.

[42]:
```
help(rgs_events)
```

```
Help on function rgs_events in module daxa.process.xmm.assemble:

rgs_events(obs_archive: daxa.archive.base.Archive, process_unscheduled: bool = True, num_
↪cores: int = 9, disable_progress: bool = False, timeout: astropy.units.quantity.
↪Quantity = None)
    This function runs the first step of the SAS RGS processing pipeline, rgsproc. This␣
↪should prepare the RGS event
    lists by calibrating and combining the separate CCD event lists into RGS events.␣
↪This happens separately for RGS1
    and RGS2, and for each sub-exposure of the two instruments.

    None of the calculations performed in this step should be affected by the choice of␣
↪source, the first step where
```

(continues on next page)

(continued from previous page)

```
    the choice of primary source should be taken into consideration is the next step,␣
↪rgs_angles; though as DAXA
    processes data to be generally useful we will not define a primary source, that is␣
↪for the user in the future as
    the aspect drift calculations can be re-run.

    :param Archive obs_archive: An Archive instance containing XMM mission instances␣
↪with RGS observations for
        which RGS processing should be run. This function will fail if no XMM missions␣
↪are present in the archive.
    :param bool process_unscheduled: Whether this function should also process sub-
↪exposures marked 'U', for
        unscheduled. Default is True, in which case they will be processed.
    :param int num_cores: The number of cores to use, default is set to 90% of available.
    :param bool disable_progress: Setting this to true will turn off the SAS generation␣
↪progress bar.
    :param Quantity timeout: The amount of time each individual process is allowed to␣
↪run for, the default is None.
        Please note that this is not a timeout for the entire rgs_events process, but a␣
↪timeout for individual
        ObsID-subexposure processes.
    :return: Information required by the SAS decorator that will run commands. Top level␣
↪keys of any dictionaries are
        internal DAXA mission names, next level keys are ObsIDs. The return is a tuple␣
↪containing a) a dictionary of
        bash commands, b) a dictionary of final output paths to check, c) a dictionary␣
↪of extra info (in this case
        obs and analysis dates), d) a generation message for the progress bar, e) the␣
↪number of cores allowed, and
        f) whether the progress bar should be hidden or not.
    :rtype: Tuple[dict, dict, dict, str, int, bool, Quantity]
```

Calculating the aspect drift for the event lists (rgs_angles)

The rgs_angles function calculates aspect drift for the RGS event lists - i.e. corrections for shifts in measured properties of events caused by the changing pointing position of the telescope (necessary because RGS instruments are grating spectrometers).

This step is source dependant because it depends on the central coordinate - however we run it with an 'uninformative' central coordinate, and for full accuracy this process should be repeated when you come to analyse the data.

[43]:
```
help(rgs_angles)
```

```
Help on function rgs_angles in module daxa.process.xmm.assemble:

rgs_angles(obs_archive: daxa.archive.base.Archive, num_cores: int = 9, disable_progress:␣
↪bool = False, timeout: astropy.units.quantity.Quantity = None)
    This function runs the second step of the SAS RGS processing pipeline, rgsproc. This␣
↪should calculate aspect drift
    corrections for some 'uninformative' source, and should likely be refined later when␣
↪these data are used to analyse
```

(continues on next page)

(continued from previous page)

```
    a specific source. This happens separately for RGS1 and RGS2, and for each sub-
↪exposure of the two instruments.

    :param Archive obs_archive: An Archive instance containing XMM mission instances␣
↪with RGS observations for
        which RGS processing should be run. This function will fail if no XMM missions␣
↪are present in the archive.
    :param int num_cores: The number of cores to use, default is set to 90% of available.
    :param bool disable_progress: Setting this to true will turn off the SAS generation␣
↪progress bar.
    :param Quantity timeout: The amount of time each individual process is allowed to␣
↪run for, the default is None.
        Please note that this is not a timeout for the entire rgs_events process, but a␣
↪timeout for individual
        ObsID-subexposure processes.
    :return: Information required by the SAS decorator that will run commands. Top level␣
↪keys of any dictionaries are
        internal DAXA mission names, next level keys are ObsIDs. The return is a tuple␣
↪containing a) a dictionary of
        bash commands, b) a dictionary of final output paths to check, c) a dictionary␣
↪of extra info (in this case
        obs and analysis dates), d) a generation message for the progress bar, e) the␣
↪number of cores allowed, and
        f) whether the progress bar should be hidden or not.
    :rtype: Tuple[dict, dict, dict, str, int, bool, Quantity]
```

Filtering RGS event lists (cleaned_rgs_event_lists)

This function (cleaned_rgs_event_lists) is similar in purpose to the `cleaned_evt_lists` function applied to EPIC data, in that it produces filtered and cleaned event lists. **No soft proton filtering is applied however**

[44]:
```
help(cleaned_rgs_event_lists)
```

```
Help on function cleaned_rgs_event_lists in module daxa.process.xmm.assemble:

cleaned_rgs_event_lists(obs_archive: daxa.archive.base.Archive, num_cores: int = 9,␣
↪disable_progress: bool = False, timeout: astropy.units.quantity.Quantity = None)
    This function runs the third step of the SAS RGS processing pipeline, rgsproc. Here␣
↪we filter the events to only
    those which should be useful for scientific analysis. The attitude and house-keeping␣
↪GTIs are also applied. This
    happens separately for RGS1 and RGS2, and for each sub-exposure of the two␣
↪instruments.

    Unfortunately it seems that combining sub-exposure event lists for a given ObsID-
↪Instrument combo is not
    supported/recommended, combinations of data are generally done after spectrum␣
↪generation, and even then they
    don't exactly recommend it - of course spectrum generation doesn't get done in DAXA.␣
↪As such this function
```

(continues on next page)

(continued from previous page)

```
    will produce individual event lists for RGS sub-exposures.

    :param Archive obs_archive: An Archive instance containing XMM mission instances␣
↪with RGS observations for
        which RGS processing should be run. This function will fail if no XMM missions␣
↪are present in the archive.
    :param int num_cores: The number of cores to use, default is set to 90% of available.
    :param bool disable_progress: Setting this to true will turn off the SAS generation␣
↪progress bar.
    :param Quantity timeout: The amount of time each individual process is allowed to␣
↪run for, the default is None.
        Please note that this is not a timeout for the entire rgs_events process, but a␣
↪timeout for individual
        ObsID-subexposure processes.
    :return: Information required by the SAS decorator that will run commands. Top level␣
↪keys of any dictionaries are
        internal DAXA mission names, next level keys are ObsIDs. The return is a tuple␣
↪containing a) a dictionary of
        bash commands, b) a dictionary of final output paths to check, c) a dictionary␣
↪of extra info (in this case
        obs and analysis dates), d) a generation message for the progress bar, e) the␣
↪number of cores allowed, and
        f) whether the progress bar should be hidden or not.
    :rtype: Tuple[dict, dict, dict, str, int, bool, Quantity]
```

3.4.2 Processing eROSITA data

This tutorial will teach you how to use DAXA to process raw eROSITA data into a science ready state using one line of Python code (or several lines, if you wish to have more control over the settings for each step). **This relies on there being an initialised (either manually before launching Python, or in your bash profile/rc) backend installation of the eROSITA Science Analysis Software System (eSASS), including accessible calibration files** - DAXA will check for such an installation, and will not allow processing to start without it.

All DAXA processing steps will parallelise as much as possible - processes running on different ObsIDs/instruments/sub-exposures will be run simultaneously (if cores are available)

Import Statements

[1]:
```
from daxa.mission import eRASS1DE, eROSITACalPV
from daxa.archive import Archive
from daxa.process.simple import full_process_erosita
from daxa.process.erosita.clean import flaregti
from daxa.process.erosita.assemble import cleaned_evt_lists
```

An Archive to be processed

Every processing function implemented in DAXA takes an Archive instance as its first argument; if you don't already know what that is then you should go back and read the following tutorials:

- Creating a DAXA archive - This explains how to create an archive, load an existing archive, and the various properties and features of DAXA archives.
- Using DAXA missions - Here we explain what DAXA mission classes are and how to use them to select only the data you need.

Here we create an archive of eRASS DR1 and eFEDS observations of the eFEDS cluster with identifier 339:

[2]:
```
ec = eROSITACalPV()
ec.filter_on_positions([[133.071, -1.025]])
er = eRASS1DE()
er.filter_on_positions([[133.071, -1.025]])

arch = Archive('eFEDSXCS-339', [ec, er], clobber=True)
```

```
/mnt/pact/dt237/code/PycharmProjects/DAXA/daxa/mission/base.py:1075: UserWarning: A␣
↪field-of-view cannot be easily defined for eROSITACalPV and this number is the␣
↪approximate half-length of an eFEDS section, the worst case separation - this is␣
↪unnecessarily large for pointed observations, and you should make your own judgement␣
↪on a search distance.
  fov = self.fov
Downloading eROSITACalPV data: 100%|| 1/1 [00:24<00:00, 24.33s/it]
Downloading eRASS DE:1 data: 100%|| 2/2 [00:01<00:00,  1.10it/s]
```

One-line solution

Though we provide individual functions that wrap the various steps required to reduce and prepare eROSITA data, and they can be used separately for greater control over the configuration parameters, we also include a one-line solution which executes the processing steps with default configuration.

We believe that the default parameters are adequate for most use cases, and this allows for users unfamiliar with the intricacies of eROSITA data to easily start working with it. Executing the following will automatically generate cleaned combined event lists for all telescope modules selected upon the original declaration of the mission.

fm00_300007_020_EventList_c001.fits

[3]:
```
full_process_erosita(arch)
```

```
eROSITACalPV - Finding flares in observations: 100%|| 2/2 [00:11<00:00,  5.91s/it]
eRASS DE:1 - Finding flares in observations: 100%|| 2/2 [00:02<00:00,  1.20s/it]
eROSITACalPV - Generating final event lists: 100%|| 2/2 [01:01<00:00, 30.75s/it]
eRASS DE:1 - Generating final event lists: 100%|| 2/2 [00:02<00:00,  1.07s/it]
```

General arguments for all processing functions

There are some arguments that all processing functions will take - they don't control the behaviour of the processing step itself, but instead how the commands are executed:

- **num_cores** - The number of cores that the function is allowed to use. This is set globally by the daxa.NUM_CORES constant (if set before any other DAXA function is imported), and by default is 90% of the cores available on the system.
- **timeout** - This controls how long an individual instance of the process is allowed to run before it is cancelled; the whole function may run for longer depending how many pieces of data need processing and how many cores are allocated. The default is generally null, but it can be set using an astropy quantity with time units.

Breaking down the eROSITA processing steps

There are fewer individual steps for eROSITA compared to a telescope like XMM - this reflects its simpler design, with only a single camera type, as well as the differences in how data are served to the community and the backend software design. This section deals describes the different processing steps that DAXA can apply to eROSITA data (both all-sky and calibration).

Identifying periods of high flaring (flaregti)

The flaregti function searches the event lists for periods where there is high soft-proton flaring - any periods where there are not are defined as good-time-intervals (GTIs) and will be used to clean the event lists later. This DAXA functions acts as an interface to the eSASS tool of the same name, which determines periods of flaring when the generated light-curve exceeds a set threshold - it also attempts to create a mask to remove sources prior to the final generation and assessment of the lightcurves.

The following arguments can be passed:

- **pimin** - Controls the lower energy bound for creating lightcurves used to determine badly flared times. Default is 0.2 keV.
- **pimax** - Controls the upper energy bound for creating lightcurves used to determine badly flared times. Default is 10.0 keV.
- **mask_pimin** - Controls the lower energy bound for data to perform source detection of emission on - in order to mask sources to account for their variability when determining flared time periods.. Default is 0.2 keV.
- **mask_pimax** - Controls the upper energy bound for data to perform source detection of emission on - in order to mask sources to account for their variability when determining flared time periods. Default is 10.0 keV.
- **binsize** - The X and Y binning size for the image on which source detection is performed to create a mask (in eROSITA sky pixels).
- **detml** - The detection likelihood threshold for sources that will be included in the mask creation.
- **timebin** - The time binning applied to the lightcurve prior to the count-rate threshold checks.
- **source_size** - The size of source for which a source likelihood is computed when creating source lists to generate a mask.
- **source_like** - The likelihood used to 'detect' a source which is then used to minimise the detected source rate to decide on the threshold for flaring events.
- **threshold** - The count-rate threshold above which the light curve is considered flared. If a positive value is set it acts as an absolute threshold for the entire observation under consideration, whereas if a negative threshold is set here the threshold is computed dynamically on a spatial grid. We set the default value to be negative.

- **max_threshold** - If positive, this limits the threshold values that are dynamically computed (if `threshold` is negative) so that they can only be less than `max_threshold`. By default, the value of this argument is negative, in which case no maximum is applied
- **mask_iter** - The number of iterations of masking, flare determination, and redection used in the creation of the final good-time intervals. The default is 3.

[4]:
```
help(flaregti)
```

```
Help on function flaregti in module daxa.process.erosita.clean:

flaregti(obs_archive: daxa.archive.base.Archive, pimin: astropy.units.quantity.Quantity␣
↪= <Quantity 200. eV>, pimax: astropy.units.quantity.Quantity = <Quantity 10000. eV>,␣
↪mask_pimin: astropy.units.quantity.Quantity = <Quantity 200. eV>, mask_pimax: astropy.
↪units.quantity.Quantity = <Quantity 10000. eV>, binsize: int = 1200, detml:␣
↪Union[float, int] = 10, timebin: astropy.units.quantity.Quantity = <Quantity 20. s>,␣
↪source_size: astropy.units.quantity.Quantity = <Quantity 25. arcsec>, source_like:␣
↪Union[float, int] = 10, threshold: astropy.units.quantity.Quantity = <Quantity -1. ct /
↪ (deg2 s)>, max_threshold: astropy.units.quantity.Quantity = <Quantity -1. ct / (deg2␣
↪s)>, mask_iter: int = 3, num_cores: int = 115, disable_progress: bool = False, timeout:
↪ astropy.units.quantity.Quantity = None)
    The DAXA wrapper for the eROSITA eSASS task flaregti, which attempts to identify␣
↪good time intervals with
    minimal flaring. This has been tested up to flaregti v1.20.

    This function does not generate final event lists, but instead is used to create␣
↪good-time-interval files
    which are then applied to the creation of final event lists, along with other user-
↪specified filters, in the
    'cleaned_evt_lists' function.

    :param Archive obs_archive: An Archive instance containing eROSITA mission instances␣
↪with observations for
        which flaregti should be run. This function will fail if no eROSITA missions are␣
↪present in the archive.
    :param float pimin:  Lower PI bound of energy range for lightcurve creation.
    :param float pimax:  Upper PI bound of energy range for lightcurve creation.
    :param float mask_pimin: Lower PI bound of energy range for finding sources to mask.
    :param float mask_pimax: Upper PI bound of energy range for finding sources to mask.
    :param int binsize: Bin size of mask image (unit: sky pixels).
    :param int detml: Likelihood threshold for mask creation.
    :param int timebin: Bin size for lightcurve (unit: seconds).
    :param int source_size: Diameter of source extracton area for dynamic threshold␣
↪calculation (unit: arcsec);
        this is the most important parameter if optimizing for extended sources.
    :param int source_like: Source likelihood for automatic threshold calculation.
    :param float threshold: Flare threshold; dynamic if negative (unit: counts/deg^2/
↪sec).
    :param float max_threshold: Maximum threshold rate, if positive (unit: counts/deg^2/
↪sec),
        if set this forces the threshold to be this rate or less.
    :param int mask_iter: Number of repetitions of source masking and GTI creation.
    :param int num_cores: The number of cores to use, default is set to 90% of available.
    :param bool disable_progress: Setting this to true will turn off the SAS generation␣
↪progress bar.
```

(continues on next page)

(continued from previous page)

```
    :param Quantity timeout: The amount of time each individual process is allowed to␣
↪run for, the default is None.
        Please note that this is not a timeout for the entire flaregti process, but a␣
↪timeout for individual
        ObsID-Inst-subexposure processes.
    :return: Information required by the eSASS decorator that will run commands. Top␣
↪level keys of any dictionaries are
        internal DAXA mission names, next level keys are ObsIDs. The return is a tuple␣
↪containing a) a dictionary of
        bash commands, b) a dictionary of final output paths to check, c) a dictionary␣
↪of extra info (in this case
        obs and analysis dates), d) a generation message for the progress bar, e) the␣
↪number of cores allowed, and
        f) whether the progress bar should be hidden or not.
    :rtype: Tuple[dict, dict, dict, str, int, bool, Quantity]
```

Applying event filtering and good-time-intervals to the raw event lists (cleaned_evt_lists)

This function (cleaned_evt_lists) creates the final, filtered and cleaned, event lists for eROSITA data. We make use of the evtool eSASS task for this. Our function will apply the good-time intervals generated by `flaregti`, as well as allowing the filtering of events based on pattern, flag, and energy. This is achieved through the passage of the following arguments:

- **lo_en** - This controls the lowest energy of event allowed into the cleaned event lists - the default is 0.2 keV, the lowest allowed by the eSASS tool.
- **hi_en** - This controls the highest energy of event allowed into the cleaned event lists - the default is 10.0 keV, the highest allowed by the eSASS tool.
- **flag** - Events are flagged during their initial processing (prior to download) - the flags represent combinations of circumstances, and include information on the owner (MPE or IKE), rejection decision, quality, and whether they are corrupted or not. We use a default value that will select all events flagged as either singly corrupt or as part of a corrupt frame.
- **flag_invert** - This controls whether the flag is used to define which events to *select* or which to *exclude*. It is often easier to define the bad events with a flag and then invert it, which is the default behaviour here - any event selected by `flag` will be excluded, unless `flag_invert` is set to False.
- **pattern** - Defines which event patterns are acceptable (where a pattern describes how an event was registered by the detector (this discusses eROSITA pattern fractions. - the default value is 15, which represent 1111 in binary, which in turn means that single, double, triple, and quadruple events are all selected by default. If the absolute highest quality is required, and you have sufficient events, then it may make sense to limit this more, in which case you could pass 1000 (for singles only), or 1010 (for singles and triples), etc.

[5]:
```
help(cleaned_evt_lists)
```

```
Help on function cleaned_evt_lists in module daxa.process.erosita.assemble:

cleaned_evt_lists(obs_archive: daxa.archive.base.Archive, lo_en: astropy.units.quantity.
↪Quantity = <Quantity 0.2 keV>, hi_en: astropy.units.quantity.Quantity = <Quantity 10.␣
↪keV>, flag: int = 3221225472, flag_invert: bool = True, pattern: int = 15, num_cores:␣
↪int = 115, disable_progress: bool = False, timeout: astropy.units.quantity.Quantity =␣
↪None)
```

(continues on next page)

(continued from previous page)

```
    The function wraps the eROSITA eSASS task evtool, which is used for selecting events.
    This has been tested up to evtool v2.10.1

    This function is used to apply the soft-proton filtering (along with any other␣
↪filtering you may desire, including
    the setting of energy limits) to eROSITA event lists, resulting in the creation of␣
↪sets of cleaned event lists
    which are ready to be analysed.

    :param Archive obs_archive: An Archive instance containing eROSITA mission instances␣
↪with observations for
        which cleaned event lists should be created. This function will fail if no␣
↪eROSITA missions are present in
        the archive.
    :param Quantity lo_en: The lower bound of an energy filter to be applied to the␣
↪cleaned, filtered, event lists. If
        'lo_en' is set to an Astropy Quantity, then 'hi_en' must be as well. Default is␣
↪0.2 keV, which is the
        minimum allowed by the eROSITA toolset. Passing None will result in the default␣
↪value being used.
    :param Quantity hi_en: The upper bound of an energy filter to be applied to the␣
↪cleaned, filtered, event lists. If
        'hi_en' is set to an Astropy Quantity, then 'lo_en' must be as well. Default is␣
↪10 keV, which is the
        maximum allowed by the eROSITA toolset. Passing None will result in the default␣
↪value being used.
    :param int flag: FLAG parameter to select events based on owner, information,␣
↪rejection, quality, and corrupted
        data. The eROSITA website contains the full description of event flags in␣
↪section 1.1.2 of the following link:
        https://erosita.mpe.mpg.de/edr/DataAnalysis/prod_descript/EventFiles_edr.html.␣
↪The default parameter will
        select all events flagged as either singly corrupt or as part of a corrupt frame.
    :param bool flag_invert: If set to True, this function will discard all events␣
↪selected by the flag parameter.
        This is the default behaviour.
    :param int pattern: Selects events of a certain pattern chosen by the integer key.␣
↪The default of 15 selects
        all four of the recognized legal patterns.
    :param int num_cores: The number of cores to use, default is set to 90% of available.
    :param bool disable_progress: Setting this to true will turn off the eSASS␣
↪generation progress bar.
    :param Quantity timeout: The amount of time each individual process is allowed to␣
↪run for, the default is None.
        Please note that this is not a timeout for the entire cleaned_evt_lists process,␣
↪but a timeout for individual
        ObsID-Inst-subexposure processes.
```

CHAPTER

FOUR

INSPECTING CLEANED DATA

4.1 Verifying the Cleaning of Count Rates of XMM Observations

4.1.1 Motivation

When it comes to observing and analyzing data from distant celestial objects, we often encounter various challenges. One such challenge is the presence of flares, which are sudden and intense bursts of radiation that can distort and obscure astronomical data. Flares can come from a variety of sources, but primarily solar flares cause problems. These events can be extremely difficult to detect and analyze, as they can occur unpredictably and with varying intensities. Therefore, removing or mitigating the effects of flares is crucial for accurate and reliable astronomical data analysis.

When observing celestial objects, astronomers often use instruments that can detect individual photons or particles of radiation. These instruments produce data in the form of events lists, which record the properties of each detected particle, such as its energy, arrival time, and position. However, as mentioned earlier, flares can produce sudden and intense bursts of radiation that can distort these events lists, making it difficult to accurately identify and analyze individual particles. In some cases, the data may even become unusable, as the flares can overwhelm the detectors and saturate the data acquisition system.

Count rate plots are a useful tool for visualizing the impact of flares on astronomy data. They show the rate at which particles are detected over time, allowing astronomers to identify periods of increased or decreased activity. Flares can often be seen as sudden spikes in the count rate, indicating a large number of particles detected in a short period of time.

By cleaning flared astronomy data, we can remove these spikes and other distortions, allowing us to more accurately identify and analyze individual particles in the events list. This can lead to more precise measurements of the properties of celestial objects, such as their distance, mass, and composition. Additionally, cleaning flared data can help us identify and study rare or unusual events, such as supernovae or gamma-ray bursts, that may be hidden in the noise of the flares.

4.1.2 Loading QDP Files

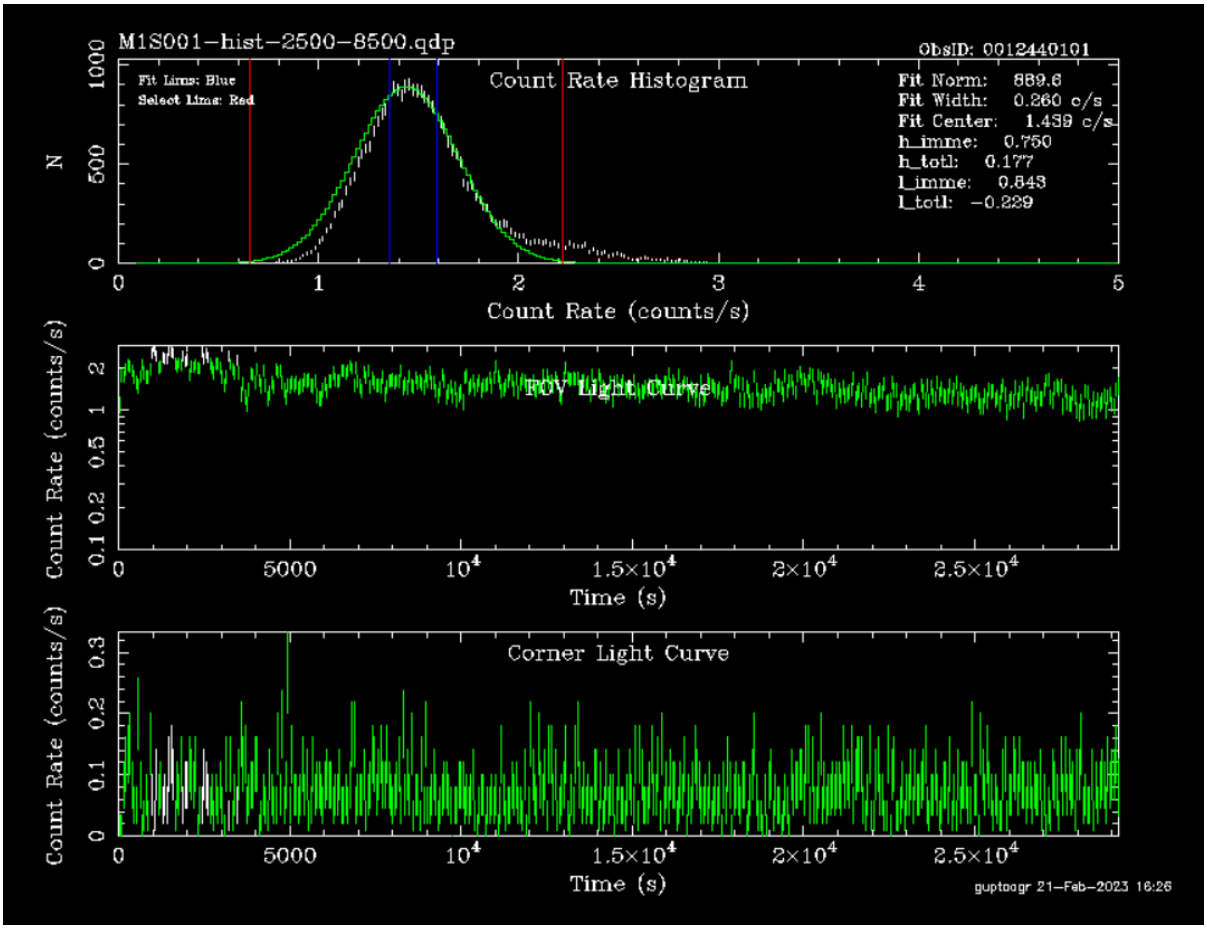
To load QDP files through the terminal, you need to have HEASOFT and SAS setup in the bash script. This allows easier access to plotting the QDP files when required. To open a QDP file, type the command “qdp myfile.qdp” and then it prompts you to enter a PGPLOT file/type. Type ‘/xs’ to open a separate window, which contains the plots corresponding to the QDP file.

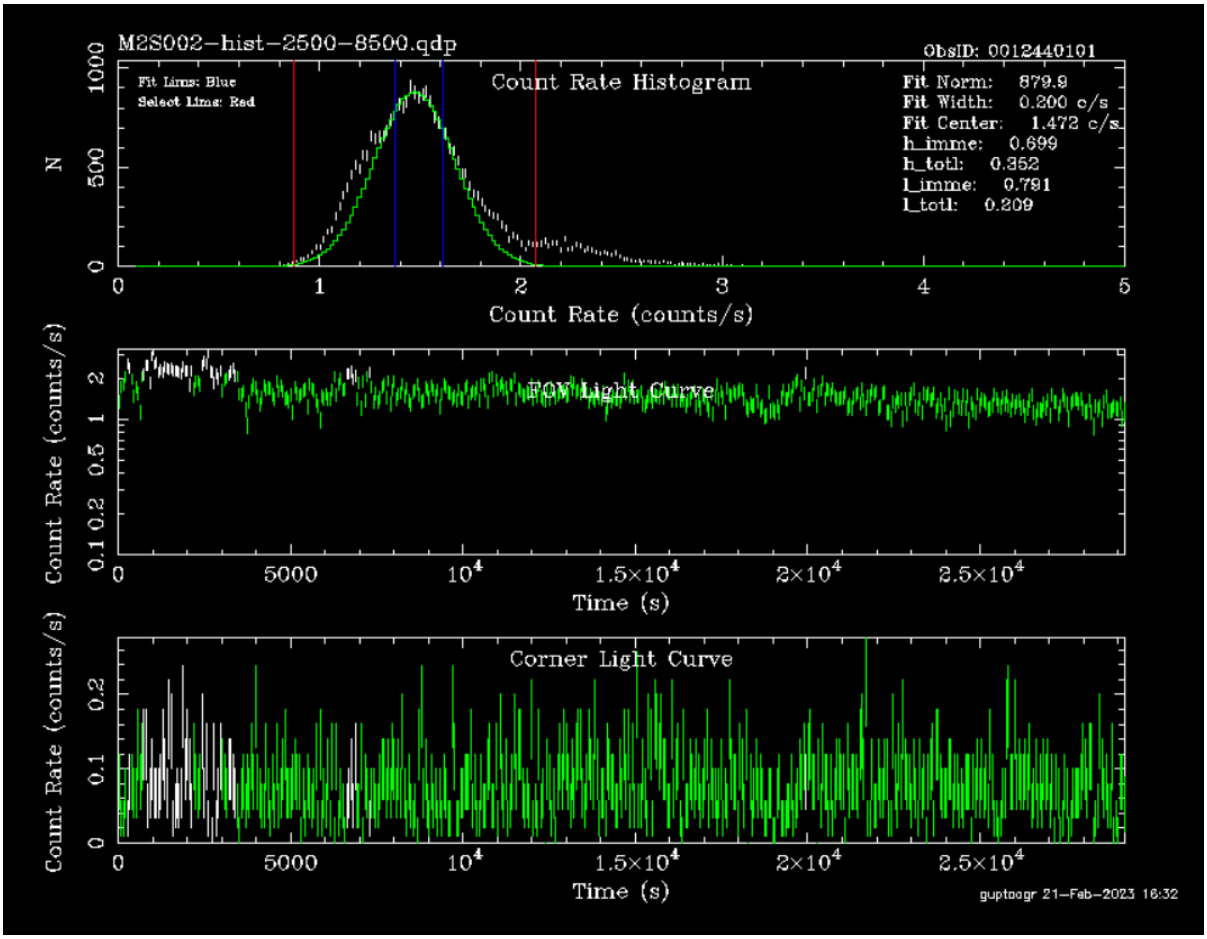
4.1.3 Cleaning Algorithm

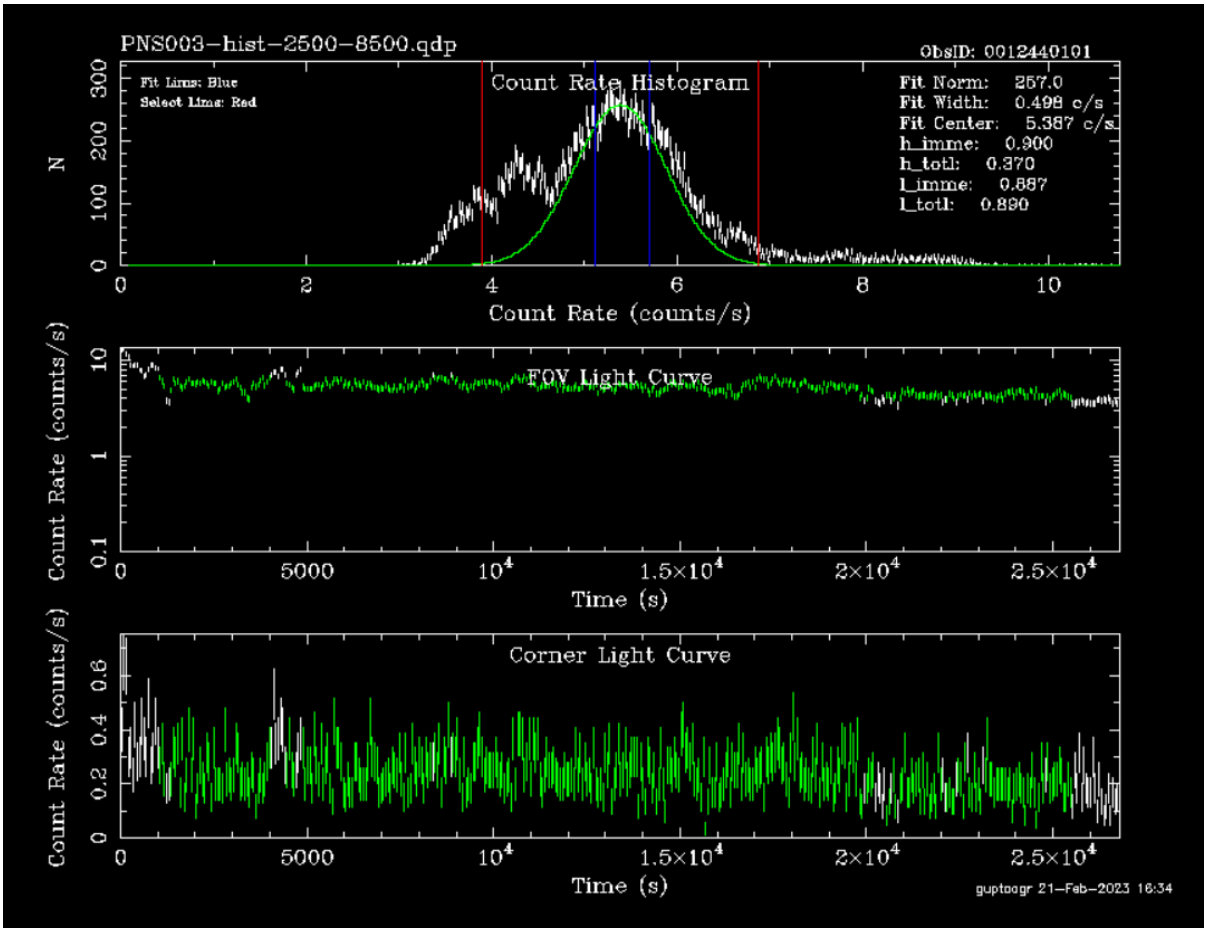
After an observation is taken, we get an events list which contains the count rates per second. Some of the data might be 'flared', which represents a higher number of count rates than usual. To get a good image from the events list of the observation, we need to remove this flaring as much as we can. A cleaning algorithm has been used to do this, which plots a histogram of the count rates and removes the count rates that are not in the interval of the selected limits (shown as red vertical bars in the histogram plots).

4.1.4 'Good' Plot (Properly Cleaned)

Loading the count rate plots for Observation ID 0012440101 and the corresponding M1, M2 and PN cameras.




M2S002-hist-2500-8500.qdp
ObsID: 0012440101
Fit Limz: Blue
Select Limz: Red
Count Rate Histogram
Fit Norm: 879.9
Fit Width: 0.200 c/s
Fit Center: 1.472 c/s
h_imme: 0.699
h_totl: 0.352
l_imme: 0.791
l_totl: 0.209
N
Count Rate (counts/s)
FOV Light Curve
Count Rate (counts/s)
Time (s)
Corner Light Curve
Count Rate (counts/s)
Time (s)
guptaogr 21-Feb-2023 16:32



In the plots above, we can see that the cleaning algorithm has been correctly applied. The count rates above and below those determined by the Gaussian (in the histogram) have been removed/cleaned from the data. These removed count rates are displayed in white. All the remaining count rates are displayed in green, which are to be used in the cleaned events lists to obtain an image of the observation.

[1]:
```
%matplotlib inline
from astropy.io import fits
from astropy.table import Table
from astropy.visualization import ImageNormalize, LogStretch, MinMaxInterval
from astropy.wcs import WCS
from astropy.visualization import astropy_mpl_style
from astropy.utils.data import get_pkg_data_filename
import os
import matplotlib.pyplot as plt
```

[2]:
```
def xrayimg(obs_id,cam,energy,typ):
    image_file = get_pkg_data_filename(obs_id+'/images/'+obs_id+'_'+cam+'_'+energy+'keV
↪'+typ+'.fits')
    image_data = fits.getdata(image_file, ext=0)
    plt.figure(figsize=(10, 8))
    if typ == 'img':
        norm = ImageNormalize(image_data, MinMaxInterval(), stretch=LogStretch())
        plt.imshow(image_data, origin='lower', cmap='plasma', norm=norm)
```

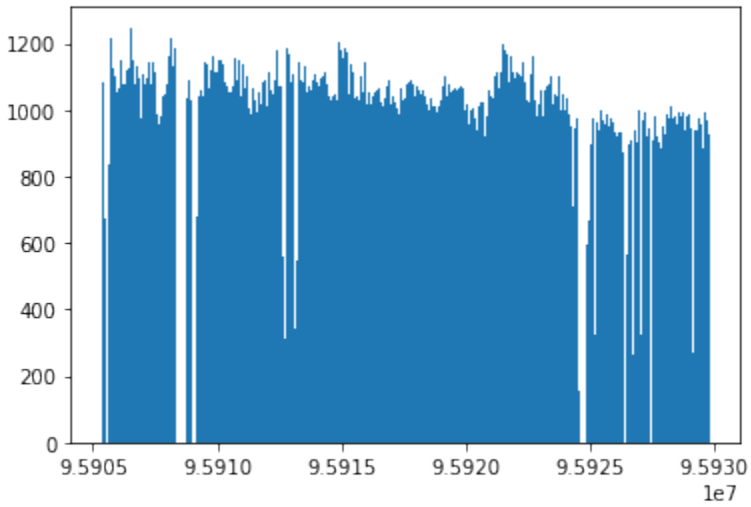
(continues on next page)

(continued from previous page)

```
        plt.colorbar()
    else:
        plt.imshow(image_data, origin='lower', cmap='plasma')
        plt.colorbar()
```

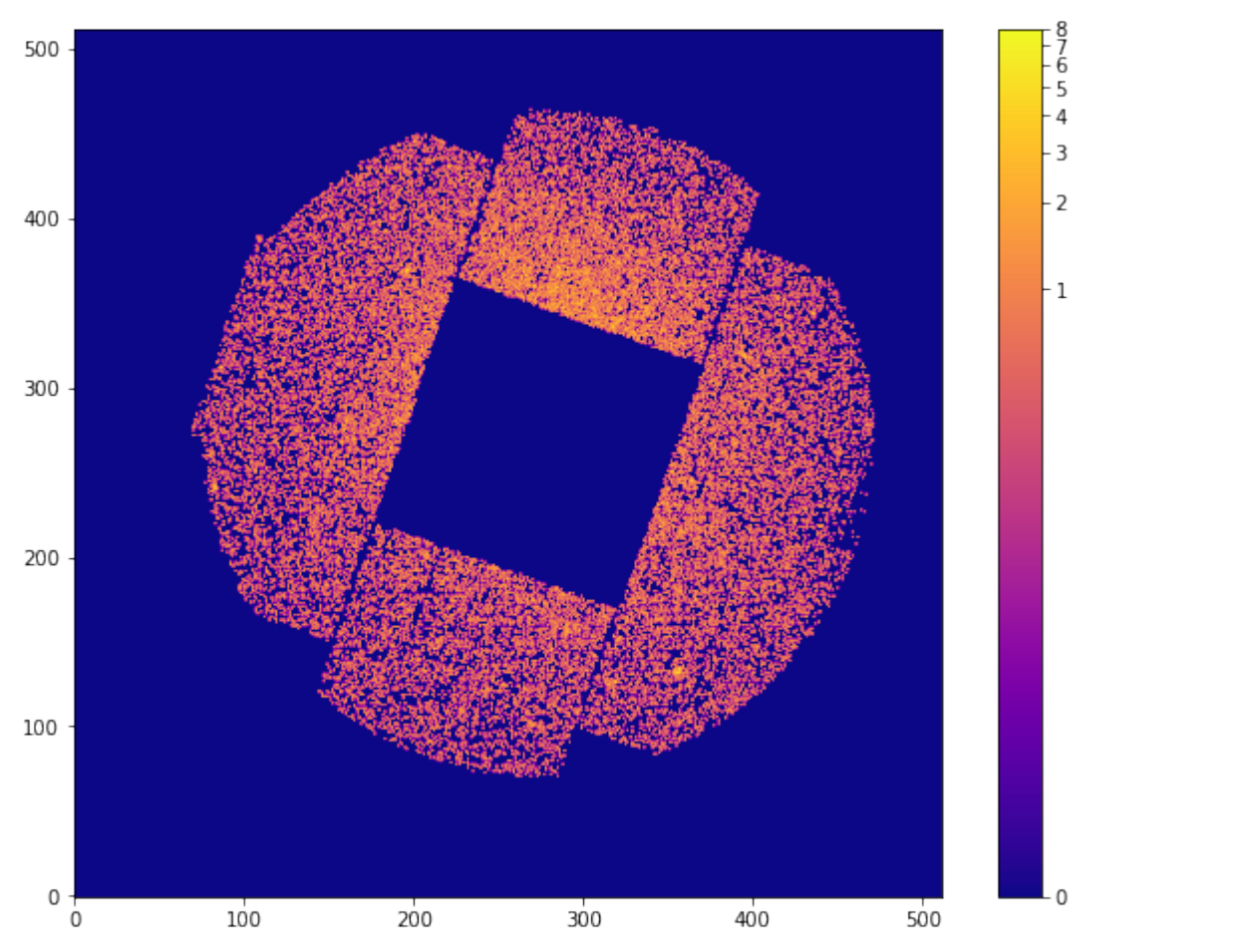
[3]:
```
def hist(obs_id, cam, bins):
    with fits.open(obs_id+'/'+cam+'_clean.fits') as hdu:
        asn_table = Table(hdu[1].data)
    plt.hist(asn_table[0][:], bins=bins)
```

[4]:
```
hist('0012440101','PN',500)
```



The histogram above is obtained from the cleaned events list of the PN camera of ObsID 0012440101. This can also be used to verify the cleaning of the data by looking at the gaps in the histogram, which indicate the removed (white) count rates in the QDP plot.

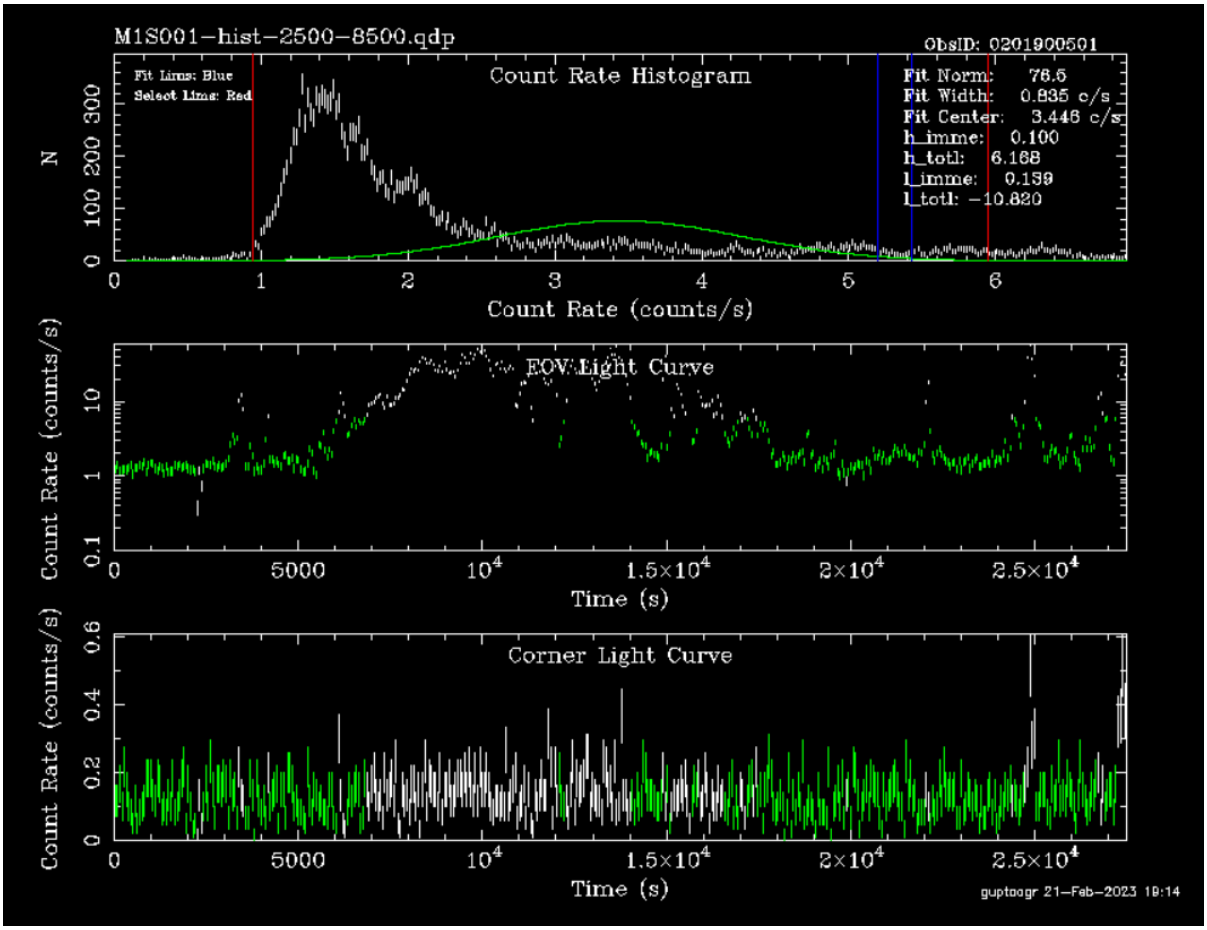
[5]:
```
xrayimg('0012440101','mos1','2.0-10.0','img')
```



This is an image of ObsID 0012440101, corresponding to the M1 camera and in the energy range 2.0-10.0 keV, which is obtained from the cleaned events list.

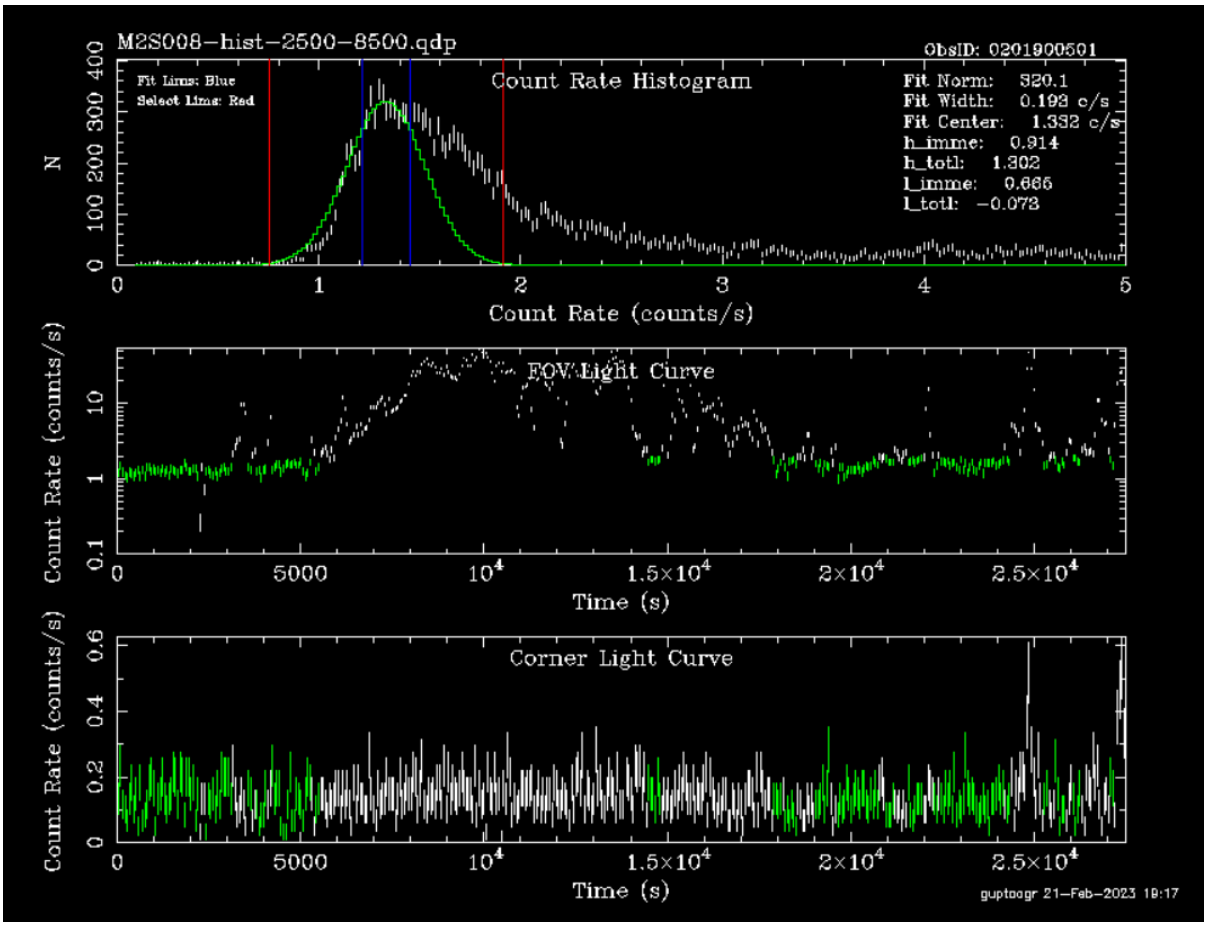
4.1.5 'Bad' Plot (Improperly Cleaned)

Loading the count rate plots for Observation ID 0201900501 and the corresponding M1 camera.



In the plot above, a proper Gaussian could have been easily fit in the Count Rate Histogram. However, a glitch in the cleaning algorithm might have resulted in improper cleaning and fitting of data. Thus, it can be observed that relatively higher count rates are green, which should instead have been white.

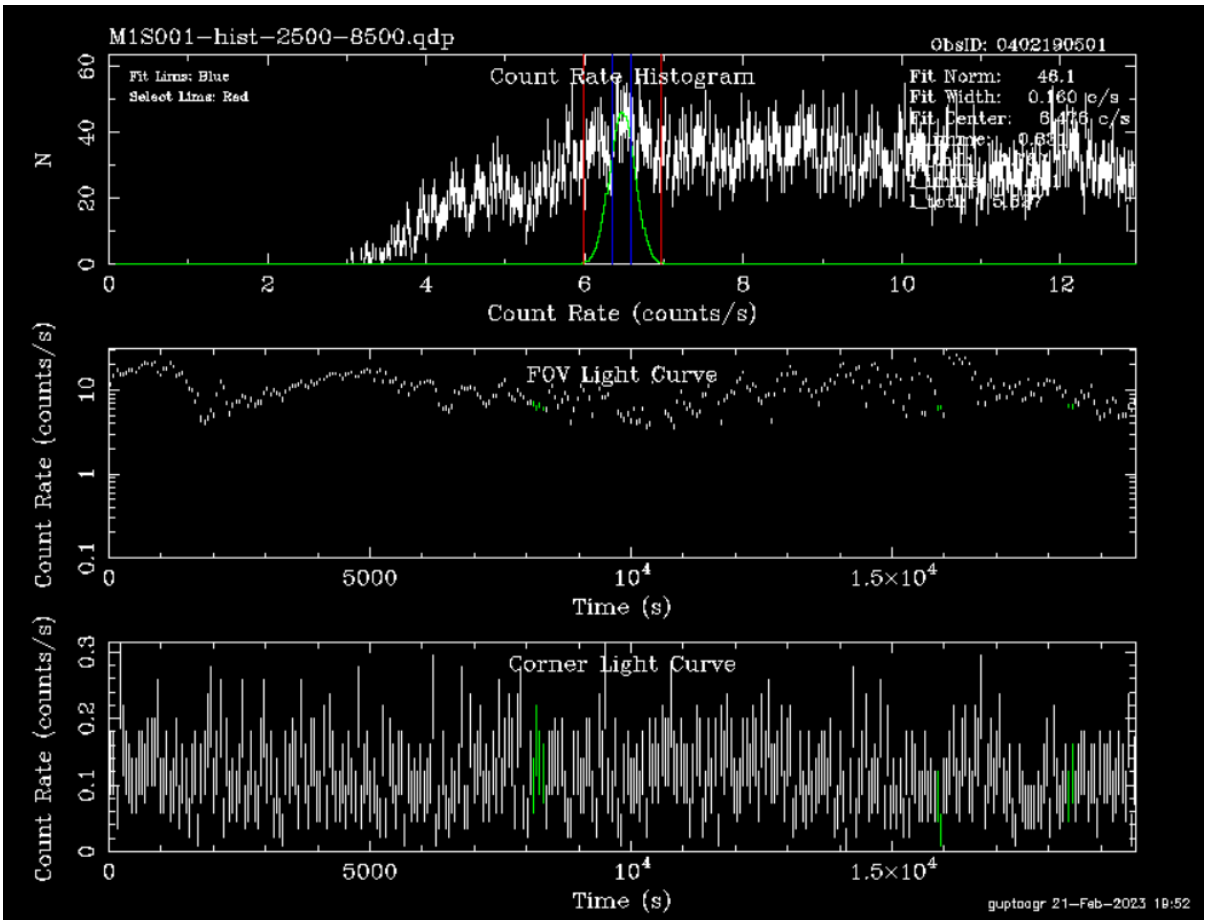
The corresponding M2 camera plot can also be seen for this observation, where the cleaning algorithm was correctly applied.

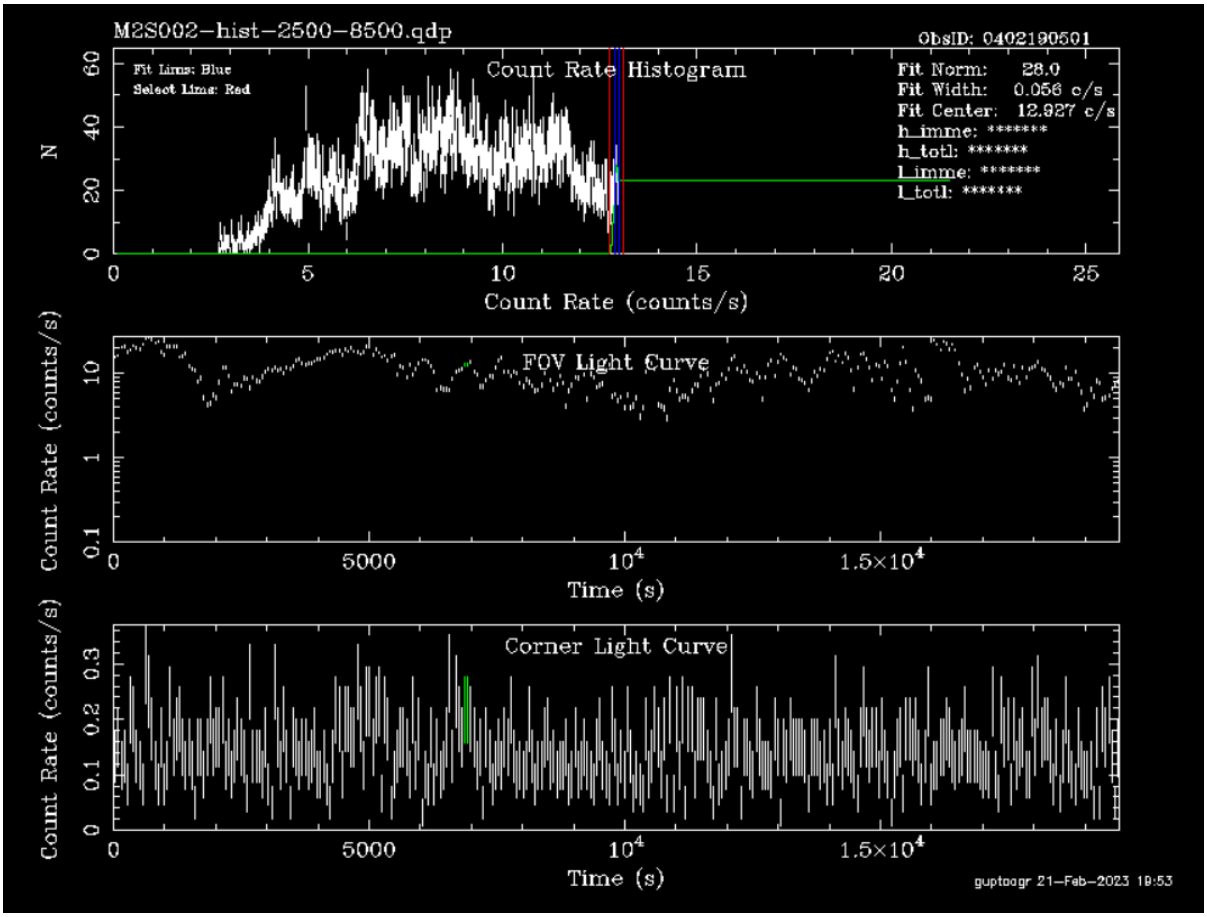


In the plot above, the cleaning algorithm was applied correctly and it can be seen that more count rates have been removed, as it should have been the case for M1 as well.

4.1.6 'Bad' Plot (Flared Data)

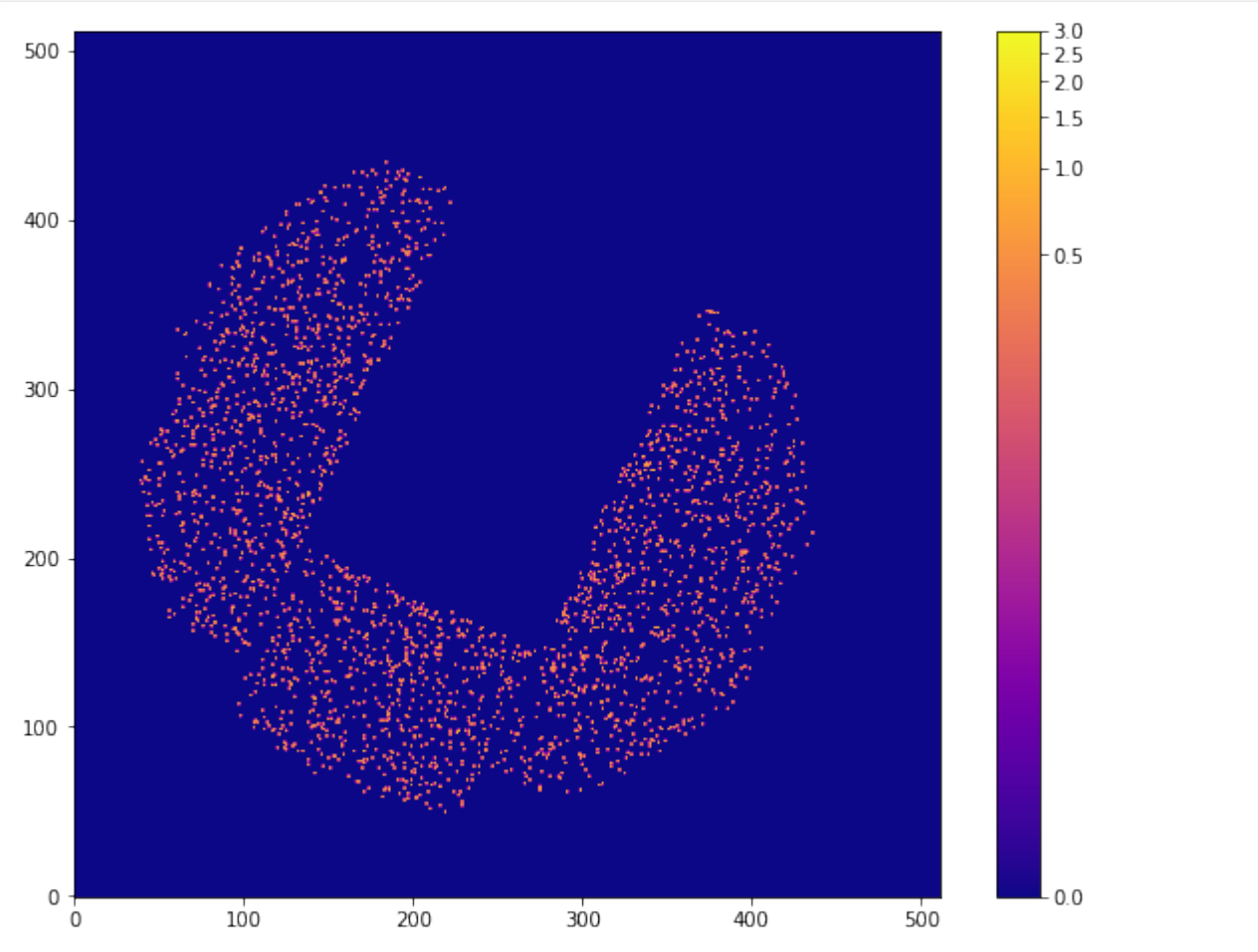
Loading the count rate plots for Observation ID 0402190501 and the corresponding M1 and M2 camera.


M1S001-hist-2500-8500.qdp
ObsID: 0402190501
Fit Limz: Blue
Select Limz: Red
Count Rate Histogram
Fit Norm: 46.1
Fit Width: 0.160 c/s
Fit Center: 6.476 c/s
N
Count Rate (counts/s)
FOV Light Curve
Count Rate (counts/s)
Time (s)
Corner Light Curve
Count Rate (counts/s)
Time (s)
guptaagr 21-Feb-2023 19:52



In the plots above, it can be seen that either the entire data or most of the data is flared. Thus, the cleaned events list would have a very low number of count rates, which can be verified by creating an image of the observation.

[6]:
```
xrayimg('0402190501','mos1','2.0-10.0','img')
```



Here, it can be seen that there's hardly an image and only a few pixels, which indicate the low number of count rates in the cleaned events list.

Note - There are also observations in which the entire data is flared, which means that there is no cleaned events list, thus there is no corresponding image as none of the data could be used to create an image.

CHAPTER

FIVE

DAXA PACKAGE

5.1 archive

5.1.1 archive.base module

class daxa.archive.base.**Archive**(*archive_name*, *missions=None*, *clobber=False*, *download_products=True*, *use_preprocessed=False*)

Bases: `object`

The Archive class, which is to be used to consolidate and provide some interface with a set of mission's data. Archives can be passed to processing and cleaning functions in DAXA, and also contain convenience functions for accessing summaries of the available data.

Parameters

- **archive_name** (*str*) – The name to be given to this archive - it will be used for storage and identification. If an existing archive with this name exists it will be read in, unless clobber=True.
- **missions** (*List[BaseMission]/BaseMission*) – The mission, or missions, which are to be included in this archive - any setup processes (i.e. the filtering of data to be acquired) should be performed prior to creating an archive. The default value is None, but this should be set for any new archives, it can only be left as None if an existing archive is being read back in.
- **clobber** (*bool*) – If an archive named 'archive_name' already exists, then setting clobber to True will cause it to be deleted and overwritten.
- **download_products** (*bool/dict*) – Controls whether pre-processed products should be downloaded for missions that offer it (assuming downloading was not triggered when the missions were declared). Default is True, but False may also be passed, as may a dictionary of DAXA mission names with True/False values.
- **use_preprocessed** (*bool/dict*) – Whether pre-processed data products should be used rather than re-processing locally with DAXA. If True then what pre-processed data products are available will be automatically re-organised into the DAXA processed data structure during the setup of this archive. If False (the default) then this will not automatically be applied. Just as with 'download_products', a dictionary may be passed for more nuanced control, with mission names as keys and True/False as values.

property archive_name

Property getter for the name assigned to this archive by the user. :return: The archive name. :rtype: str

property top_level_path

The property getter for the absolute path to the top-level DAXA storage directory.

Returns
Absolute top-level storage path.

Return type
str

property archive_path

The property getter for the absolute path to the output archive directory.

Returns
Absolute path to the archive.

Return type
str

property mission_names

Property getter for the names of the missions associated with this Archive.

Returns

Return type
List[str]

property missions

Property getter that returns a list of missions associated with this Archive.

Returns
Missions associated with this archive.

Return type
List[*BaseMission*]

property preprocessed_missions

Gets a list of missions that have pre-processed data downloaded, if there are none an error will be raised.

Returns
A list of the mission instances in this archive which have pre-processed data downloaded.

Return type
List[*BaseMission*]

property process_success

Property getter for a nested dictionary containing boolean flags describing whether different processing steps applied to observations from various missions are considered to have completed successfully.

Returns
A nested dictionary where top level keys are mission names, next level keys are processing function names, and lowest level keys are either ObsID or ObsID+instrument names. The values attributed with the lowest level keys are boolean, with True indicating that the processing function was successful

Return type
dict

property process_errors

Property getter for a nested dictionary containing error information from processing applied to mission data.

Returns
A nested dictionary where top level keys are mission names, next level keys are processing function names, and lowest level keys are either ObsID or ObsID+instrument names. The values attributed with the lowest level keys are error outputs (e.g. parsed from stderr from command line tools).

Return type
dict

`property process_warnings`

Property getter for a nested dictionary containing warning information from processing applied to mission data.

Returns
A nested dictionary where top level keys are mission names, next level keys are processing function names, and lowest level keys are either ObsID or ObsID+instrument names. The values attributed with the lowest level keys are warning outputs (e.g. parsed from stderr from command line tools).

Return type
dict

`property raw_process_errors`

Property getter for a nested dictionary containing unparsed error information (e.g. the entire stderr output from an XMM SAS process) from processing applied to mission data.

Returns
A nested dictionary where top level keys are mission names, next level keys are processing function names, and lowest level keys are either ObsID or ObsID+instrument names. The values attributed with the lowest level keys are error outputs (e.g. stderr from command line tools).

Return type
dict

`property process_logs`

Property getter for a nested dictionary containing log information from processing applied to mission data.

Returns
A nested dictionary where top level keys are mission names, next level keys are processing function names, and lowest level keys are either ObsID or ObsID+instrument names. The values attributed with the lowest level keys are logs (e.g. stdout from command line tools).

Return type
dict

`property process_extra_info`

Property getter for a nested dictionary containing extra information from processing applied to mission data. This can be things like paths to event lists, or configuration information. It is unlikely to be necessary for users to directly access this property.

Returns
A nested dictionary where top level keys are mission names, next level keys are processing function names, and lowest level keys are either ObsID or ObsID+instrument names. The values attributed with the lowest level keys are dictionaries of extra information (e.g. config info).

Return type
dict

property process_names

Property that returns a dictionary containing the names of all processing steps that have been run on this archive. Top-level keys are mission names, and the values are lists of process names.

Returns
The dictionary containing mission name and process name information. Top-level keys are mission names, and the values are lists of process names.

Return type
dict

property observation_summaries

This property returns information on the different observations available to each mission. This information will vary from mission to mission, and is primarily intended for use by DAXA processing methods, but could include things such as whether an instrument was active for a particular observation, what subexposures there were (relevant for XMM for instance), what filter was active, etc.

Returns
A dictionary of information with missions as the top level keys, then ObsIDs, then instruments. Keys on levels below that will be determined by the information available for specific instruments of specific missions.

Return type
bool

property process_observation

This property returns the dictionary of mission-ObsID-Instrument(-subexposure) boolean flags that indicate whether the data for that observation-instrument(-subexposure) should be processed for science. There is a companion get method that returns only the data identifiers that should be processed.

Returns
The dictionary containing information on whether particular data should be processed.

Return type
dict

property final_process_success

This property returns the dictionary which stores the final judgement (at the ObsID level) of whether there are any useful data (True) or whether no aspect of that observation reached the end of the final processing step successfully. The ObsIDs marked as False will be moved from the archive processed data directory to a separate failed data directory.

The flags are only added once the final processing step for a particular mission has been run.

Returns
The dictionary of final processing success flags.

Return type
dict

property source_regions

This property returns all source regions which have been associated with missions in this archive. The top level keys of the dictionary are mission names, the bottom level keys are observation identifiers, and the values are lists of region objects.

If an observation in this archive has had regions added for it, then those regions will also have been written to permanent storage in the archive directory structure. The path can be identified using the get_region_file_path method of this archive.

Returns
Dictionary containing regions on a mission-observation basis.

Return type
dict

`get_current_data_path`(*mission*, *obs_id*)

A method which returns the current location of the archive data for a particular ObsID of a particular mission. The two location options are in the 'processed' directory, which is the default and will be the home of all ObsIDs that haven't made it to the final process for a particular mission, or the 'failed' directory, where any ObsID that has no use (per the final checks) will be stored.

Parameters

- **mission** (*BaseMission/str*) – The mission for which to retrieve the current data path.
- **obs_id** (*str*) – The ObsID for which to retrieve the current data path.

Returns
The current path to the requested ObsID of the specified mission.

Return type
str

`construct_processed_data_path`(*mission=None*, *obs_id=None*)

This method is to construct paths to directories where processed data for a particular mission + observation ID combination will be stored. That functionality is added here so that any change to how those directories are named will take place in only one part of DAXA, and will propagate to other parts of the module. It is unlikely that a user will need to directly use this method.

If no mission is passed, then no observation ID may be passed. In the case of 'mission' and 'obs_id' being None, the returned string will be constructed ready to format; {mn} should be replaced by the DAXA mission name, and {oi} by the relevant ObsID.

Retrieving a data path from this method DOES NOT guarantee that it has been created.

Parameters

- **mission** (*BaseMission/str*) – The mission for which to retrieve the processed data path. Default is None in which case a path ready to be formatted with a mission name will be provided.
- **obs_id** (*str*) – The ObsID for which to retrieve the processed data path, cannot be set if 'mission' is set to None. Default is None, in which case a path ready to be formatted with an observation ID will be provided.

Returns
The requested path.

Return type
str

`construct_failed_data_path`(*mission=None*, *obs_id=None*)

This method is to construct paths to directories where data for a particular mission + observation ID combination which failed to process will be stored. That functionality is added here so that any change to how those directories are named will take place in only one part of DAXA, and will propagate to other parts of the module. It is unlikely that a user will need to directly use this method.

If no mission is passed, then no observation ID may be passed. In the case of 'mission' and 'obs_id' being None, the returned string will be constructed ready to format; {mn} should be replaced by the DAXA mission name, and {oi} by the relevant ObsID.

Retrieving a data path from this method DOES NOT guarantee that it has been created.

Parameters

- **mission** (*BaseMission/str*) – The mission for which to retrieve the failed data path. Default is None in which case a path ready to be formatted with a mission name will be provided.
- **obs_id** (*str*) – The ObsID for which to retrieve the failed data path, cannot be set if 'mission' is set to None. Default is None, in which case a path ready to be formatted with an observation ID will be provided.

Returns
The requested path.

Return type
str

get_region_file_path(*mission=None*, *obs_id=None*)

This method is to construct paths to files where the regions associated with a particular observation of a particular mission are stored after being added to the archive. If a mission and ObsID are specified then this method will check whether region information for that particular ObsID of that particular mission exists in this archive, and raise an error if it does not.

If no mission is passed, then no observation ID may be passed. In the case of 'mission' and 'obs_id' being None, the returned string will be constructed ready to format; {mn} should be replaced by the DAXA mission name, and {oi} by the relevant ObsID.

Retrieving a region file path from this method without passing mission and ObsID DOES NOT guarantee that one has been created for whatever mission and ObsID are added to the string later.

Parameters

- **mission** (*BaseMission/str*) – The mission for which to retrieve the region file path. Default is None in which case a path ready to be formatted with a mission name will be provided.
- **obs_id** (*str*) – The ObsID for which to retrieve the region file path, cannot be set if 'mission' is set to None. Default is None, in which case a path ready to be formatted with an observation ID will be provided.

Returns
The requested path.

Return type
str

get_obs_to_process(*mission_name*, *search_ident=None*)

This method will provide a list of lists of [ObsID, Instrument, SubExposure (depending on mission)] that should be processed for scientific use for a specific mission. The idea is that this method can be called, and just by iterating through the result you will get the identifiers of all valid data that match your input.

It shouldn't really need to be used directly by users, but instead will be very useful for the processing functions - it will tell them which data need to be processed.

Parameters

- **mission_name** (*str*) – The internal DAXA name of the mission to retrieve information for.
- **search_ident** (*str*) – Either an ObsID or an instrument name to retrieve matching information for. An ObsID will search through all the instruments/subexposures, an instrument

will search all ObsIDs and sub-exposures. The default is None, in which case all ObsIDs, instruments, and sub-exposures will be searched.

Returns
List of lists of [ObsID, Instrument, SubExposure (depending on mission)].

Return type
List[List]

`check_dependence_success`(*mission_name*, *obs_ident*, *dep_proc*, *no_success_error=True*)

This method should be used by processing functions, rather than the user, to determine whether previous processing steps (specified in the input to this function) ran successfully for the specified data.

Each processing function should be setup to call this method with appropriate previous steps and identifiers, and will know from its boolean array return which data can be processed safely. If no data has successfully run through a previous step, or no attempt to run a previous step occurred, then an error will be thrown.

Parameters

- **mission_name** (*str*) – The name of the mission for which we wish to check the success of previous processing steps.
- **obs_ident** (*str/List[str], List[List[str]]*) – A set (or individual) set of observation identifiers. This should be in the style output by get_obs_to_process (i.e. [ObsID, Inst, SubExp (depending on mission)], though does also support just an ObsID.
- **dep_proc** (*str/List[str]*) – The name(s) of the process(es) that have to have been run for further processing steps to be successful.
- **no_success_error** (*bool*) – If none of the specified previous processing steps have been run successfully, should a NoDependencyProcessError be raised. Default is True, but if set to False the error will not be raised and the return will be an all-False array. This will NOT override the error raised if a previous process hasn't been run at all.

Returns
A boolean array that defines whether the process(es) specified in the input were successful. Each set of identifying information provided in obs_ident has a corresponding entry in the return.

Return type
np.ndarray

`get_process_logs`(*process_name*, *mission_name=None*, *obs_id=None*, *inst=None*, *full_ident=None*)

This method allows for targeted retrieval of processing logs (stdout), for a specific processing step. The particular logs retrieved can be narrows down by mission, ObsID, or instrument. Multiple missions, ObsIDs, and instruments may be specified, but only one process at a time. The names of processes that have been run can be found in the 'process_names' property of an Archive.

Parameters

- **process_name** (*str*) – The process for which logs are to be retrieved (see 'process_names' property for the names of processes run on this archive).
- **mission_name** (*str/List[str]*) – The mission name(s) for which logs are to be retrieved. Default is None, in which case all missions will be searched, and either a single name or a list of names can be passed. See 'mission_names' for a list of associated mission names.
- **obs_id** (*str/List[str]*) – The ObsID(s) for which logs are to be retrieved. Default is None, in which case all ObsIDs will be searched. Either a single or a set of ObsIDs can be passed.

- **inst** (*str/List[str]*) – The instrument(s) for which logs are to be retrieved. Default is None, in which case all instruments will be searched. Either a single or a set of instruments can be passed.
- **full_ident** (*str/List[str]*) – A full unique identifier (or a set of them) to make matches too. This will override any ObsID or insts that are specified - for instance one could pass 0201903501PNS003. Default is None.

Returns
A dictionary containing the requested logs - top level keys are mission names, lower level keys are unique identifiers, and the values are string logs which match the provided information.

Return type
dict

get_process_raw_error_logs(*process_name*, *mission_name=None*, *obs_id=None*, *inst=None*, *full_ident=None*)

This method allows for targeted retrieval of processing raw-error logs (stderr), for a specific processing step. The particular logs retrieved can be narrows down by mission, ObsID, or instrument. Multiple missions, ObsIDs, and instruments may be specified, but only one process at a time. The names of processes that have been run can be found in the 'process_names' property of an Archive.

Parameters

- **process_name** (*str*) – The process for which logs are to be retrieved (see 'process_names' property for the names of processes run on this archive).
- **mission_name** (*str/List[str]*) – The mission name(s) for which logs are to be retrieved. Default is None, in which case all missions will be searched, and either a single name or a list of names can be passed. See 'mission_names' for a list of associated mission names.
- **obs_id** (*str/List[str]*) – The ObsID(s) for which logs are to be retrieved. Default is None, in which case all ObsIDs will be searched. Either a single or a set of ObsIDs can be passed.
- **inst** (*str/List[str]*) – The instrument(s) for which logs are to be retrieved. Default is None, in which case all instruments will be searched. Either a single or a set of instruments can be passed.
- **full_ident** (*str/List[str]*) – A full unique identifier (or a set of them) to make matches too. This will override any ObsID or insts that are specified - for instance one could pass 0201903501PNS003. Default is None.

Returns
A dictionary containing the requested logs - top level keys are mission names, lower level keys are unique identifiers, and the values are string logs which match the provided information.

Return type
dict

get_failed_processes(*process_name*)

A simple method to retrieve all unique identifiers of data that failed a particular processing step. The names of processes that have been run can be found in the 'process_names' property of an Archive.

Parameters
process_name (*str*) – The process for which unique identifiers of data that failed the processing step are to be retrieved (see 'process_names' property for the names of processes run on this archive).

Returns
A dictionary, with mission names as top level keys, and values being lists of failed unique identifiers.

Return type
dict

`get_failed_logs`(*process_name*)

A convenience method that retrieves the logs (stdout and stderr) for processing of particular data (be it a whole ObsID, a particular instrument of an ObsID, or a particular sub-exposure of a particular instrument of an ObsID) which FAILED.

Parameters
process_name (*str*) – The process for which logs (stdout and stderr) are to be retrieved if the data of a particular unique identifier failed.

Returns
A tuple of two dictionaries, the first containing stdout logs, and the second containing stderr logs - the structure of the dictionaries has mission names as top level keys, unique identifiers as lower level keys, and string logs as values.

Return type
Tuple[dict, dict]

`delete_raw_data`(*force_del=False*, *all_raw_data=False*)

This method will delete raw data downloaded for the missions in this archive; by default only directories corresponding to ObsIDs currently accepted through a mission's filter will be deleted, but if all_raw_data is set to True then the WHOLE raw data directory corresponding to a particular mission will be removed.

Confirmation from the user will be sought that they wish to delete the data, unless force_del is set to True - in which case the removal will be performed straight away.

Parameters

- **force_del** (*bool*) – This argument can be used to ensure that the delete option can be performed entirely programmatically, without requiring a user input. Default is False, but if set to True then the delete operation will be performed immediately.
- **all_raw_data** (*bool*) – This controls whether only the data selected by the current instance of each mission are deleted (when False, the default behaviour) or if the whole directory associated with each mission is removed.

`save`()

A simple method that saves the information necessary to reload this archive from disk at a later time. This largely consists of the various pieces of information regarding the success (or not) of various processing steps.

NOTE that the mission states are not saved here, as they could be triggered repeatedly, which can be slow for the ones with many possible ObsIDs (i.e. Swift and Integral). Instead, saves are triggered when the archive is created, in the init, and if the data in the archive are updated (as this necessitates a change in the mission states).

`info`()

A simple method to present summary information about this archive.

5.1.2 archive.assemble module

5.2 mission

5.2.1 mission.base module

class daxa.mission.base.**BaseMission**

Bases: object

The superclass for all missions defined in this module. Mission classes will be for storing and interacting with information about the available data for particular missions; including filtering the observations to be prepared and reduced in various ways. The mission classes will also be responsible for providing a consistent user experience of downloading data and generating processed archives.

abstract property name

Abstract property getter for the name of this mission. Must be overwritten in any subclass. This is to ensure that any subclasses that people might add will definitely set a proper name, which is not guaranteed by having it done in the init.

Returns
The mission name

Return type
str

property pretty_name

The property getter for the 'pretty name' of this mission. This version of the name will NOT be used to identify a mission internally in DAXA, or to name any directories, but will be used when the user sees a name (e.g. when a progress bar is running for a mission download).

Returns
The 'pretty' name.

Return type
str

abstract property coord_frame

Abstract property getter for the coordinate frame of the RA-Decs of the observations of this mission. Must be overwritten in any subclass. This is to ensure that any subclasses that people might add will definitely set a coordinate frame, which is not guaranteed by having it done in the init.

Returns
The coordinate frame of the RA-Dec

Return type
BaseRADecFrame

abstract property id_regex

Abstract property getter for the regular expression (regex) pattern for observation IDs of this mission. Must be overwritten in any subclass. This is to ensure that any subclasses that people might add will definitely set an ID pattern, which is not guaranteed by having it done in the init.

Returns
The regex pattern for observation IDs.

Return type
str

`abstract property fov`

Abstract property getter for the approximate field-of-view of this mission's instrument(s). In cases where different instruments have different field-of-views this may be a dictionary (see ROSATPointed for an example). Must be overwritten in any subclass. This is to ensure that any subclasses that people might add will definitely set a FoV, which is not guaranteed by having it done in the init.

The convention will be that the value supplied is the radius/half-side-length of the field of view. In cases where the field of view is not square/circular, it should be the half-side-length of the longest side.

A dictionary should ONLY be defined if the instruments have different field of views, and have their own observations in the all_obs_info table (e.g. ROSAT's instruments are mutually exclusive and cannot have multiple per observation).

Returns
The approximate field of view(s) for the mission's instrument(s). In cases with multiple instruments then this may be a dictionary, with keys being instrument names.

Return type
Union[Quantity, dict]

`property all_mission_instruments`

Property getter for the names of all possible instruments associated with this mission.

Returns
A list of instrument names.

Return type
List[str]

`property chosen_instruments`

Property getter for the names of the currently selected instruments associated with this mission which will be processed into an archive by DAXA functions.

Returns
A list of instrument names

Return type
List[str]

`property top_level_path`

The property getter for the absolute path to the top-level directory where raw data storage directories are created.

Returns
Absolute top-level storage path.

Return type
str

`property raw_data_path`

Property getter for the directory in which raw data for the current mission is stored.

Returns
Storage path for raw data for this mission.

Return type
str

`property filter_array`

A property getter for the 'filter' array, which is set by the filtering methods built-in to this class (or can be

set externally using the filter_array property setter) and controls which observations will be downloaded and processed.

Returns
An array of boolean values; True means that an observation is used, False means that it is not.

Return type
np.ndarray

`property filtering_operations`

A property getter for the filtering operations that have been applied to this mission, in the order they were applied. This is mainly stored so that missions that have been reinstated from a save file can be updated by running the exact same filtering operations again.

Returns
A list of dictionaries which have two keys, 'name', and 'arguments'; the 'name' key corresponds to the name of the filtering method, and the 'arguments' key corresponds to a dictionary of arguments that were passed to the method. 0th element was applied first, Nth element was applied last.

Return type
List[dict]

`abstract property all_obs_info`

A property getter that returns the base dataframe containing information about all the observations available for an instance of a mission class. This is an abstract method purely because its property setter is an abstract method, one cannot be without the other.

Returns
A pandas dataframe with (at minimum) the following columns; 'ra', 'dec', 'ObsID', 'science_usable', 'start', 'duration'

Return type
pd.DataFrame

`property filtered_obs_info`

A property getter that applies the current filter array to the dataframe of observation information, and returns filtered dataframe containing all columns available for this mission.

Returns
A filtered dataframe of observation information.

Return type
pd.DataFrame

`property science_usable`

Property getter for the 'science_usable' column of the all observation information dataframe. This 'science_usable' column describes whether a particular observation is usable by this module; i.e. that the data are suitable for scientific use (so far as can be identified by querying the storage service). This science_usable property is the basis for the filter array, resetting the filter array will return it to the values of this column.

Data that are marked as scientifically useful but are still in a proprietary period will return True here, as the user may have been the one to take those data. If suitable credentials cannot be produced at download time however, those proprietary data will be marked as unusable.

Returns
A boolean array detailing whether an observation is scientifically useful or not.

Return type
np.ndarray

property ra_decs

Property getter for the RA-Dec coordinates of ALL the observations associated with this mission - for the coordinates of filtered observations (i.e. the observations that will actually be used for downloading/processing), see the filtered_ra_decs property.

Returns
The full set of RA-Dec coordinates of all observations associated with this mission.

Return type
SkyCoord

property filtered_ra_decs

Property getter for the RA-Dec coordinates of the filtered set of observations associated with this mission - for coordinates of ALL observations see the ra_decs property.

Returns
The RA-Dec coordinates of filtered observations associated with this mission.

Return type
SkyCoord

property obs_ids

Property getter for the ObsIDs of ALL the observations associated with this mission - for the ObsIDs of filtered observations (i.e. the observations that will actually be used for downloading/processing), see the filtered_obs_ids property.

Returns
The full set of ObsIDs of all observations associated with this mission.

Return type
np.ndarray

property filtered_obs_ids

Property getter for the ObsIDs of the filtered set of observations associated with this mission - for ObsIDs of ALL observations see the obs_ids property.

Returns
The ObsIDs of filtered observations associated with this mission.

Return type
np.ndarray

property download_completed

Property getter that describes whether the specified data for this mission have been downloaded.

Returns
Boolean flag describing if data have been downloaded.

Return type
bool

property downloaded_type

Property getter that describes what type of data was downloaded for this mission (or raises an exception if no download has been performed yet). The value will be either 'raw', 'preprocessed', or 'raw+preprocessed'.

Returns
A string identifier for the type of data downloaded; the value will be either 'raw', 'preprocessed', or 'raw+preprocessed'

Return type
str

property locked

Property getter for the locked attribute of this mission instance - if a mission is locked then no further changes can be made to the observations selected.

Returns
The locked boolean.

Return type
bool

property processed

A property getter that returns whether the observations associated with this mission have been fully processed or not.

Returns
The processed boolean flag.

Return type
bool

property preprocessed_energy_bands

Property getter for a non-scalar astropy Quantity containing the energy bands of the pre-processed products supplied by this mission. The return will be in the form of a dictionary with instrument names as keys and an array of pairs of energies, in keV, as values.

Returns
A dictionary with mission instrument names as keys, and non-scalar astropy Quantities as values, with the first column being lower energy bounds and the second column being upper energy bounds.

Return type
Quantity

property one_inst_per_obs

This property returns a boolean flag that describes whether this mission has one instrument per ObsID or not. Most DAXA missions have multiple instruments per observation (or can do, if the user has selected them).

Returns
Flag showing whether there are multiple instruments per observation.

Return type
bool

reset_filter()

Very simple method which simply resets the filter array, meaning that all observations THAT HAVE BEEN MARKED AS USABLE will now be downloaded and processed, and any filters applied to the current mission have been undone.

check_obsid_pattern(*obs_id_to_check*)

A simple method that will check an input ObsID against the ObsID regular expression pattern defined for the current mission class. If the input ObsID is compliant with the regular expression then True will be returned, if not then False will be returned.

Parameters
obs_id_to_check (*str*) – The ObsID that we wish to check against the ID pattern.

Returns
A boolean flag indicating whether the input ObsID is compliant with the ID regular expression. True means that it is, False means it is not.

Return type
bool

check_inst_names(*insts*, *error_on_bad_inst=True*, *show_warn=True*)

A method to perform some checks on the validity of chosen instrument names for a given mission.

Parameters

- **insts** (*List[str]/str*) – Instrument names that are to be checked for the current mission, either a single name or a list of names.
- **error_on_bad_inst** (*bool*) – Controls whether an exception is raised if the instrument(s) aren't actually associated with this mission - intended for DAXA checking operations (see 'get_process_logs' of Archive for an example). Default is True.
- **show_warn** (*bool*) – Should warnings produced by this method be shown? Default is True

Returns
The list of instruments (possibly altered to match formats expected by this module).

Return type
List

filter_on_obs_ids(*allowed_obs_ids*)

This filtering method will select only observations with IDs specified by the allowed_obs_ids argument.

Please be aware that filtering methods are cumulative, so running another method will not remove the filtering that has already been applied, you can use the reset_filter method for that.

Parameters
allowed_obs_ids (*str/List[str]*) – The ObsID (or list of ObsIDs) that you wish to be let through the filter.

filter_on_rect_region(*lower_left*, *upper_right*)

A method that filters observations based on whether their CENTRAL COORDINATE falls within a rectangular region defined using coordinates of the bottom left and top right corners. Observations are kept if they fall within the region.

Please be aware that filtering methods are cumulative, so running another method will not remove the filtering that has already been applied, you can use the reset_filter method for that.

Parameters

- **lower_left** (*SkyCoord/np.ndarray/list*) – The RA-Dec coordinates of the lower left corner of the rectangular region. This can be passed as a SkyCoord, or a list/array with two entries - this will then be used to create a SkyCoord which assumes the default frame of the current mission and that the inputs are in degrees. NOTE that we wish the coordinates to be passed with RA increasing from left to right, but we will attempt to interpret coordinates passed with RA increasing from right to left, and will show a warning.
- **upper_right** (*SkyCoord/np.ndarray/list*) – The RA-Dec coordinates of the upper right corner of the rectangular region. This can be passed as a SkyCoord, or a list/array with two entries - this will then be used to create a SkyCoord which assumes the default frame of the current mission and that the inputs are in degrees. NOTE that we wish the coordinates to be passed with RA increasing from left to right, but we will attempt to interpret coordinates passed with RA increasing from right to left, and will show a warning.

filter_on_positions(*positions*, *search_distance=None*, *return_pos_obs_info=False*)

This method allows you to filter the observations available for a mission based on a set of coordinates for which you wish to locate observations. The method searches for observations by the current mission that have central coordinates within the distance set by the search_distance argument.

Please be aware that filtering methods are cumulative, so running another method will not remove the filtering that has already been applied, you can use the reset_filter method for that.

Parameters

- **positions** (*list/np.ndarray/SkyCoord*) – The positions for which you wish to search for observations. They can be passed either as a list or nested list (i.e. [r, d] OR [[r1, d1], [r2, d2]]), a numpy array, or an already defined SkyCoord. If a list or array is passed then the coordinates are assumed to be in degrees, and the default mission frame will be used.
- **search_distance** (*Quantity/float/int/list/np.ndarray/dict*) – The distance within which to search for observations by this mission. Distance may be specified either as an Astropy Quantity that can be converted to degrees (a float/integer will be assumed to be in units of degrees), as a dictionary of quantities/floats/ints where the keys are names of different instruments (possibly with different field of views), or as a non-scalar Quantity, list, or numpy array with one entry per set of coordinates (for when you wish to use different search distances for each object). The default is None, in which case a value of 1.2 times the approximate field of view defined for each instrument will be used; where different instruments have different FoVs, observation searches will be undertaken on an instrument-by-instrument basis using the different field of views.
- **return_pos_obs_info** (*bool*) – Allows this method to return information (in the form of a Pandas dataframe) which identifies the positions which have been associated with observations, and the observations they have been associated with. Default is False.

Returns

If return_pos_obs_info is True, then a dataframe containing information on which ObsIDs are relevant to which positions will be returned. If return_pos_obs_info is False, then None will be returned.

Return type

Union[None,pd.DataFrame]

filter_on_name(*object_name*, *search_distance=None*, *parse_name=False*)

This method wraps the 'filter_on_positions' method, and allows you to filter the mission's observations so that it contains data on a single (or a list of) specific objects. The names are passed by the user, and then parsed into coordinates using the Sesame resolver. Those coordinates and the search distance are then used to find observations that might be relevant.

Parameters

- **object_name** (*str/List[str]*) – The name(s) of objects you would like to search for.
- **search_distance** (*Quantity/float/int/list/np.ndarray/dict*) – The distance within which to search for observations by this mission. Distance may be specified either as an Astropy Quantity that can be converted to degrees (a float/integer will be assumed to be in units of degrees), as a dictionary of quantities/floats/ints where the keys are names of different instruments (possibly with different field of views), or as a non-scalar Quantity, list, or numpy array with one entry per set of coordinates (for when you wish to use different search distances for each object). The default is None, in which case a value of 1.2 times the approximate field of view defined for each instrument will be used; where different instruments have different FoVs, observation searches will be undertaken on an instrument-by-instrument basis using the different field of views.
- **parse_name** (*bool*) – Whether to attempt extracting the coordinates from the name by parsing with a regex. For objects catalog names that have J-coordinates embedded in their names, e.g., 'CRTS SSS100805 J194428-420209', this may be much faster than a Sesame query for the same object name.

filter_on_time(*start_datetime*, *end_datetime*, *over_run=True*)

This method allows you to filter observations for this mission based on when they were taken. A start and end time are passed by the user, and observations that fall within that window are allowed through the filter. The exact behaviour of this filtering method is controlled by the over_run argument, if set to True then observations with a start or end within the search window will be selected, but if False then only observations with a start AND end within the window are selected.

Please be aware that filtering methods are cumulative, so running another method will not remove the filtering that has already been applied, you can use the reset_filter method for that.

Parameters

- **start_datetime** (*datetime*) – The beginning of the time window in which to search for observations.
- **end_datetime** (*datetime*) – The end of the time window in which to search for observations.
- **over_run** (*bool*) – This controls whether selected observations have to be entirely within the passed time window or whether either a start or end time can be within the search window. If set to True then observations with a start or end within the search window will be selected, but if False then only observations with a start AND end within the window are selected. Default is True.

filter_on_target_type(*target_type*)

This method allows the filtering of observations based on what type of object their target source was. It is only supported for missions that have that data available, and will raise an exception for those missions that don't support this filtering.

WARNING: You should not trust these target types without question, they are the result of crude mappings, and some may be incorrect. They also don't take into account sources that might serendipitously appear in a particular observation.

Parameters

target_type (*str/List[str]*) – The types of target source you would like to find observations of. For allowed types, please use the 'show_allowed_target_types' method. Can either be a single type, or a list of types.

filter_on_positions_at_time(*positions*, *start_datetimes*, *end_datetimes*, *search_distance=None*, *return_obs_info=False*, *over_run=True*)

This method allows you to filter the observations available for a mission based on a set of coordinates for which you wish to locate observations that were taken within a certain time frame. The method spatially searches for observations that have central coordinates within the distance set by the search_distance argument, and temporally by start and end times passed by the user; and observations that fall within that window are allowed through the filter.

The exact behaviour of the temporal filtering method is controlled by the over_run argument, if set to True then observations with a start or end within the search window will be selected, but if False then only observations with a start AND end within the window are selected.

Please be aware that filtering methods are cumulative, so running another method will not remove the filtering that has already been applied, you can use the reset_filter method for that.

Parameters

- **positions** (*list/np.ndarray/SkyCoord*) – The positions for which you wish to search for observations. They can be passed either as a list or nested list (i.e. [r, d] OR [[r1, d1], [r2, d2]]), a numpy array, or an already defined SkyCoord. If a list or array is passed then the coordinates are assumed to be in degrees, and the default mission frame will be used.

- **start_datetimes** (*np.array(datetime)/datetime*) – The beginnings of time windows in which to search for observations. There should be one entry per position passed.
- **end_datetimes** (*np.array(datetime)/datetime*) – The endings of time windows in which to search for observations. There should be one entry per position passed.
- **search_distance** (*Quantity/float/int/list/np.ndarray/dict*) – The distance within which to search for observations by this mission. Distance may be specified either as an Astropy Quantity that can be converted to degrees (a float/integer will be assumed to be in units of degrees), as a dictionary of quantities/floats/ints where the keys are names of different instruments (possibly with different field of views), or as a non-scalar Quantity, list, or numpy array with one entry per set of coordinates (for when you wish to use different search distances for each object). The default is None, in which case a value of 1.2 times the approximate field of view defined for each instrument will be used; where different instruments have different FoVs, observation searches will be undertaken on an instrument-by-instrument basis using the different field of views.
- **return_obs_info** (*bool*) – Allows this method to return information (in the form of a Pandas dataframe) which identifies the positions which have been associated with observations, in the specified time frame, and the observations they have been associated with. Default is False.
- **over_run** (*bool*) – This controls whether selected observations have to be entirely within the passed time window or whether either a start or end time can be within the search window. If set to True then observations with a start or end within the search window will be selected, but if False then only observations with a start AND end within the window are selected. Default is True.

abstract download(*download_products=False*)

An abstract method to actually acquire and download the mission data that have not been filtered out (if a filter has been applied, otherwise all data will be downloaded). This must be overwritten by every subclass as each mission might need a different method of downloading the data, the same reason fetch_obs_info must be overwritten in each subclass.

abstract assess_process_obs(*obs_info*)

A slightly unusual abstract method which will allow each mission to assess the information on a particular observation that has been put together by an Archive (the archive assembles it because sometimes this detailed information only becomes available at the first stages of processing), and make a decision on whether that particular observation-instrument-subexposure (for missions like XMM) should be processed further for scientific use.

Implemented as an abstract method because the information and decision-making process will likely be different for every mission.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters

obs_info (*dict*) – The multi-level dictionary containing available observation information for an observation.

abstract ident_to_obsid(*ident*)

A slightly unusual abstract method which will allow each mission convert a unique identifier being used in the processing steps to the ObsID (as these unique identifiers will contain the ObsID). This is necessary because XMM, for instance, has processing steps that act on whole ObsIDs (e.g. cifbuild), and processing steps that act on individual sub-exposures of instruments of ObsIDs, so the ID could be '0201903501M1S001'.

Implemented as an abstract method because the unique identifier style may well be different for different missions - many will just always be the ObsID, but we want to be able to have low level control.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
ident (*str*) – The unique identifier used in a particular processing step.

static show_allowed_target_types(*table_format='fancy_grid'*)

This simple method just displays the DAXA source type taxonomy (the target source types you can filter by) in a nice table, with descriptions of what each source type means. Filtering on target source type is not guaranteed to work with every mission, as target type information is not necessarily available, but this filtering is used through the filter_on_target_type method.

Parameters
table_format (*str*) – The style format for the table to be displayed (should be one of the 'tabulate' module formats). The default is 'fancy_grid'.

get_evt_list_path(*obs_id*, *inst=None*)

A get method that provides the path to a downloaded pre-generated event list for the current mission (if available). This method will not work if pre-processed data have not been downloaded.

Parameters

- **obs_id** (*str*) – The ObsID of the event list.
- **inst** (*str*) – The instrument of the event list (if applicable).

Returns
The requested event list path.

Return type
str

get_image_path(*obs_id*, *lo_en=None*, *hi_en=None*, *inst=None*)

A get method that provides the path to a downloaded pre-generated image for the current mission (if available). This method will not work if pre-processed data have not been downloaded.

Parameters

- **obs_id** (*str*) – The ObsID of the image.
- **lo_en** (*Quantity*) – The lower energy bound of the image.
- **hi_en** (*Quantity*) – The upper energy bound of the image.
- **inst** (*str*) – The instrument of the image (if applicable).

Returns
The requested image file path.

Return type
str

get_expmap_path(*obs_id*, *lo_en=None*, *hi_en=None*, *inst=None*)

A get method that provides the path to a downloaded pre-generated exposure map for the current mission (if available). This method will not work if pre-processed data have not been downloaded.

Parameters

- **obs_id** (*str*) – The ObsID of the exposure map.
- **lo_en** (*Quantity*) – The lower energy bound of the exposure map.

- **hi_en** (*Quantity*) – The upper energy bound of the exposure map.
- **inst** (*str*) – The instrument of the exposure map (if applicable).

Returns
The requested exposure map file path.

Return type
str

get_background_path(*obs_id*, *lo_en=None*, *hi_en=None*, *inst=None*)

A get method that provides the path to a downloaded pre-generated background map for the current mission (if available). This method will not work if pre-processed data have not been downloaded.

Parameters

- **obs_id** (*str*) – The ObsID of the background map.
- **lo_en** (*Quantity*) – The lower energy bound of the background map.
- **hi_en** (*Quantity*) – The upper energy bound of the background map.
- **inst** (*str*) – The instrument of the background map (if applicable).

Returns
The requested background map file path.

Return type
str

delete_raw_data(*force_del=False*, *all_raw_data=False*)

This method will delete raw data downloaded for this mission; by default only directories corresponding to ObsIDs currently accepted through the filter will be deleted, but if all_raw_data is set to True then the WHOLE raw data directory corresponding to this mission will be removed.

Confirmation from the user will be sought that they wish to delete the data, unless force_del is set to True - in which case the removal will be performed straight away.

Parameters

- **force_del** (*bool*) – This argument can be used to ensure that the delete option can be performed entirely programmatically, without requiring a user input. Default is False, but if set to True then the delete operation will be performed immediately.
- **all_raw_data** (*bool*) – This controls whether only the data selected by the current instance of the mission are deleted (when False, the default behaviour) or if the whole directory associated with the mission is removed.

save(*save_root_path*, *state_file_name=None*)

A method to save a file representation of the current state of a DAXA mission object. This may be used by the user, and can be safely sent to another user or system to recreate a mission. It is also used by the archive saving mechanic, so that mission objects can be re-set up - it is worth noting that the archive save files ARE NOT how to make a portable archive.

Parameters

- **save_root_path** (*str*) – The DIRECTORY where you wish a save file to be stored.
- **state_file_name** (*str*) – Optionally, the name of the file to be stored in the root save directory. If this is not supplied (the default is None) then the output file will be called {mission name}_state.json. Any filename passed to this argument must end in '.json'.

info()

5.2.2 mission.xmm module

class daxa.mission.xmm.**XMMPointed**(*insts=None*, *save_file_path=None*)

Bases: *BaseMission*

The mission class for pointed XMM observations (i.e. slewing observations are NOT included in the data accessed and collected by instances of this class). The available observation information is fetched from the XMM Science Archive using AstroQuery, and data are downloaded with the same module.

Parameters

- **insts** (*List[str]/str*) – The instruments that the user is choosing to download/process data from. The EPIC PN, MOS1, and MOS2 instruments are selected by default. You may also select RGS1 (R1) and RGS2 (R2), though as they less widely used they are not selected by default. It is also possible to select the Optical Monitor (OM), though it is an optical/UV telescope, and as such it is not selected by default.
- **save_file_path** (*str*) – An optional argument that can use a DAXA mission class save file to recreate the state of a previously defined mission (the same filters having been applied etc.)

property name

Property getter for the name of this mission.

Returns
The mission name

Return type
str

property coord_frame

Property getter for the coordinate frame of the RA-Decs of the observations of this mission.

Returns
The coordinate frame of the RA-Dec.

Return type
BaseRADecFrame

property id_regex

Property getter for the regular expression (regex) pattern for observation IDs of this mission.

Returns
The regex pattern for observation IDs.

Return type
str

property fov

Property getter for the approximate field of view set for this mission. This is the radius/half-side-length of the field of view. In cases where the field of view is not square/circular, it is the half-side-length of the longest side.

Returns
The approximate field of view(s) for the mission's instrument(s). In cases with multiple instruments then this may be a dictionary, with keys being instrument names.

Return type
Union[Quantity, dict]

property all_obs_info

A property getter that returns the base dataframe containing information about all the observations available for an instance of a mission class.

Returns
A pandas dataframe with (at minimum) the following columns; 'ra', 'dec', 'ObsID', 'science_usable', 'start', 'duration'

Return type
pd.DataFrame

download(*num_cores=1*, *credentials=None*, *download_products=False*)

A method to acquire and download the pointed XMM data that have not been filtered out (if a filter has been applied, otherwise all data will be downloaded). Instruments specified by the chosen_instruments property will be downloaded, which is set either on declaration of the class instance or by passing a new value to the chosen_instruments property.

Parameters

- **num_cores** (*int*) – The number of cores that can be used to parallelise downloading the data. Default is the value of NUM_CORES, specified in the configuration file, or if that hasn't been set then 90% of the cores available on the current machine. The number of cores will be CAPPED AT 10 FOR XMM - we have experienced reliably dropped connections when more than 10 download processes are created.
- **credentials** (*dict/str*) – The path to an ini file containing credentials, a dictionary containing 'user' and 'password' entries, or a dictionary of ObsID top level keys, with 'user' and 'password' entries for providing different credentials for different observations.
- **download_products** (*bool*) – CURRENTLY NON-FUNCTIONAL.

assess_process_obs(*obs_info*)

A slightly unusual method which will allow the XMMPointed mission to assess the information on a particular observation that has been put together by an Archive (the archive assembles it because sometimes this detailed information only becomes available at the first stages of processing), and make a decision on whether that particular observation-instrument-subexposure should be processed further for scientific use.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
obs_info (*dict*) – The multi-level dictionary containing available observation information for an observation.

Returns
A two-level dictionary with instruments as the top level keys, and sub-exposure IDs as the low level keys. The values associated with the sub-exposure keys are boolean, True for usable, False for not.

Return type
dict

ident_to_obsid(*ident*)

A slightly unusual abstract method which will allow each mission convert a unique identifier being used in the processing steps to the ObsID (as these unique identifiers will contain the ObsID). This is necessary because XMM, for instance, has processing steps that act on whole ObsIDs (e.g. cifbuild), and processing steps that act on individual sub-exposures of instruments of ObsIDs, so the ID could be '0201903501M1S001'.

Implemented as an abstract method because the unique identifier style may well be different for different missions - many will just always be the ObsID, but we want to be able to have low level control.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
ident (*str*) – The unique identifier used in a particular processing step.

5.2.3 mission.erosita module

class daxa.mission.erosita.**eROSITACalPV**(*insts=None*, *fields=None*, *save_file_path=None*)

Bases: *BaseMission*

The mission class for the eROSITA early data release observations made during the Calibration and Performance Verification program.

Parameters

- **fields** (*List[str]/str*) – The eROSITA calibration field name(s) or type to download/process data from.
- **insts** (*List[str]/str*) – The instruments that the user is choosing to download/process data from.
- **save_file_path** (*str*) – An optional argument that can use a DAXA mission class save file to recreate the state of a previously defined mission (the same filters having been applied etc.)

property name

Property getter for the name of this mission.

Returns
The mission name

Return type
str

property chosen_instruments

Property getter for the names of the currently selected instruments associated with this mission which will be processed into an archive by DAXA functions. Overwritten here because there are custom behaviours for eROSITA.

Returns
A list of instrument names

Return type
List[str]

property coord_frame

Property getter for the coordinate frame of the RA-Decs of the observations of this mission.

Returns
The coordinate frame of the RA-Dec.

Return type
BaseRADecFrame

property id_regex

Property getter for the regular expression (regex) pattern for observation IDs of this mission.

Returns
The regex pattern for observation IDs.

Return type
str

property fov

Property getter for the approximate field of view set for this mission. This is the radius/half-side-length of the field of view. In cases where the field of view is not square/circular, it is the half-side-length of the longest side.

NOTE - THIS FIELD OF VIEW IS SORT OF NONSENSE BECAUSE OF HOW SOME OF THE eROSITACalPV DATA WERE TAKEN IN POINTING AND SOME IN SLEWING MODE.

Returns
The approximate field of view(s) for the mission's instrument(s). In cases with multiple instruments then this may be a dictionary, with keys being instrument names.

Return type
Union[Quantity, dict]

property all_obs_info

A property getter that returns the base dataframe containing information about all the observations available for an instance of a mission class.

Returns
A pandas dataframe with (at minimum) the following columns; 'ra', 'dec', 'ObsID', 'science_usable', 'start', 'duration'

Return type
pd.DataFrame

property all_mission_fields

Property getter for the names of all possible fields associated with this mission.

Returns
A list of field names.

Return type
List[str]

property all_mission_field_types

Property getter for the names of all possible field types associated with this mission.

Returns
A list of field types.

Return type
List[str]

property chosen_fields

Property getter for the names of the currently selected fields associated with this mission which will be processed into an archive by DAXA functions.

Returns
A list of field names

Return type
List[str]

filter_on_fields(*fields*)

This filtering method will select only observations included in the fields specified.

Parameters
allowed_fields (*str/List[str]*) – The fields or field types (or list of fields or field types) that you wish to be let through the filter.

filter_on_obs_ids(*allowed_obs_ids*)

This filtering method will select only observations with IDs specified by the allowed_obs_ids argument.

Please be aware that filtering methods are cumulative, so running another method will not remove the filtering that has already been applied, you can use the reset_filter method for that.

Parameters
allowed_obs_ids (*str/List[str]*) – The ObsID (or list of ObsIDs) that you wish to be let through the filter.

download(*num_cores=1*, *download_products=True*)

A method to acquire and download the eROSITA Calibration and Performance Validation data that have not been filtered out (if a filter has been applied, otherwise all data will be downloaded). Fields (or field types) specified by the chosen_fields property will be downloaded, which is set either on declaration of the class instance or by passing a new value to the chosen_fields property. Downloaded data is then filtered according to Instruments specified by the chosen_instruments property (set in the same manner as chosen_fields).

Parameters

- **num_cores** (*int*) – The number of cores that can be used to parallelise downloading the data. Default is the value of NUM_CORES, specified in the configuration file, or if that hasn't been set then 90% of the cores available on the current machine.
- **download_products** (*bool*) – UNLIKE MOST MISSIONS, this does not actually change what is downloaded, but rather changes the DAXA classification of the downloaded event lists from raw to raw+preprocessed. This means they would be included in the processed data storage structure of an archive.

get_evt_list_path(*obs_id*, *inst=None*)

A get method that provides the path to a downloaded pre-generated event list for the current mission (if available). This method will not work if pre-processed data have not been downloaded.

Parameters

- **obs_id** (*str*) – The ObsID of the event list.
- **inst** (*str*) – The instrument of the event list (if applicable).

Returns
The requested event list path.

Return type
str

assess_process_obs(*obs_info*)

A slightly unusual method which will allow the eROSITACalPV mission to assess the information on a particular observation that has been put together by an Archive (the archive assembles it because sometimes this detailed information only becomes available at the first stages of processing), and make a decision on whether that particular observation-instrument should be processed further for scientific use.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
obs_info (*dict*) – The multi-level dictionary containing available observation information for an observation.

`ident_to_obsid`(*ident*)

A slightly unusual abstract method which will allow each mission convert a unique identifier being used in the processing steps to the ObsID (as these unique identifiers will contain the ObsID). This is necessary because XMM, for instance, has processing steps that act on whole ObsIDs (e.g. cifbuild), and processing steps that act on individual sub-exposures of instruments of ObsIDs, so the ID could be '0201903501M1S001'.

Implemented as an abstract method because the unique identifier style may well be different for different missions - many will just always be the ObsID, but we want to be able to have low level control.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
ident (*str*) – The unique identifier used in a particular processing step.

`class daxa.mission.erosita.eRASS1DE`(*insts=None*, *save_file_path=None*)

Bases: *BaseMission*

The mission class for the first data release of the German half of the eROSITA All-Sky Survey

Parameters

- **insts** (*List[str]/str*) – The instruments that the user is choosing to download/process data from.
- **save_file_path** (*str*) – An optional argument that can use a DAXA mission class save file to recreate the state of a previously defined mission (the same filters having been applied etc.)

`property name`

Property getter for the name of this mission.

Returns
The mission name

Return type
str

`property chosen_instruments`

Property getter for the names of the currently selected instruments associated with this mission which will be processed into an archive by DAXA functions. Overwritten here because there are custom behaviours for eROSITA.

Returns
A list of instrument names

Return type
List[str]

`property coord_frame`

Property getter for the coordinate frame of the RA-Decs of the observations of this mission.

Returns
The coordinate frame of the RA-Dec.

Return type
BaseRADecFrame

property id_regex

Property getter for the regular expression (regex) pattern for observation IDs of this mission.

Returns
The regex pattern for observation IDs.

Return type
str

property fov

Property getter for the approximate field of view set for this mission. This is the radius/half-side-length of the field of view. In cases where the field of view is not square/circular, it is the half-side-length of the longest side.

Returns
The approximate field of view(s) for the mission's instrument(s). In cases with multiple instruments then this may be a dictionary, with keys being instrument names.

Return type
Union[Quantity, dict]

property all_obs_info

A property getter that returns the base dataframe containing information about all the observations available for an instance of a mission class.

Returns
A pandas dataframe with (at minimum) the following columns; 'ra', 'dec', 'ObsID', 'science_usable', 'start', 'duration'

Return type
pd.DataFrame

download(*num_cores=1*, *download_products=True*, *pipeline_version=None*)

A method to acquire and download the German eROSITA All-Sky Survey DR1 data that have not been filtered out (if a filter has been applied, otherwise all data will be downloaded). Downloaded data is then filtered according to Instruments specified by the chosen_instruments property (set in the same manner as chosen_fields).

Parameters

- **num_cores** (*int*) – The number of cores that can be used to parallelise downloading the data. Default is the value of NUM_CORES, specified in the configuration file, or if that hasn't been set then 90% of the cores available on the current machine.
- **download_products** (*bool*) – This controls whether the data downloaded include the images and exposure maps generated by the eROSITA team and included in the first data release. The default is True.
- **pipeline_version** (*int*) – The processing pipeline version used to generate the data that is to be downloaded. The default is None, in which case the latest available will be used.

get_evt_list_path(*obs_id*, *inst=None*)

A get method that provides the path to a downloaded pre-generated event list for the current mission (if available). This method will not work if pre-processed data have not been downloaded.

Parameters

- **obs_id** (*str*) – The ObsID of the event list.
- **inst** (*str*) – The instrument of the event list (if applicable).

Returns
The requested event list path.

Return type
str

`get_image_path`(*obs_id*, *lo_en=None*, *hi_en=None*, *inst=None*)

A get method that provides the path to a downloaded pre-generated image for the current mission (if available). This method will not work if pre-processed data have not been downloaded.

Parameters

- **obs_id** (*str*) – The ObsID of the image.
- **lo_en** (*Quantity*) – The lower energy bound of the image.
- **hi_en** (*Quantity*) – The upper energy bound of the image.
- **inst** (*str*) – The instrument of the image (if applicable).

Returns
The requested image file path.

Return type
str

`get_expmap_path`(*obs_id*, *lo_en=None*, *hi_en=None*, *inst=None*)

A get method that provides the path to a downloaded pre-generated exposure map for the current mission (if available). This method will not work if pre-processed data have not been downloaded.

Parameters

- **obs_id** (*str*) – The ObsID of the exposure map.
- **lo_en** (*Quantity*) – The lower energy bound of the exposure map.
- **hi_en** (*Quantity*) – The upper energy bound of the exposure map.
- **inst** (*str*) – The instrument of the exposure map (if applicable).

Returns
The requested exposure map file path.

Return type
str

`get_background_path`(*obs_id*, *lo_en=None*, *hi_en=None*, *inst=None*)

A get method that provides the path to a downloaded pre-generated background map for the current mission (if available). This method will not work if pre-processed data have not been downloaded.

Parameters

- **obs_id** (*str*) – The ObsID of the background map.
- **lo_en** (*Quantity*) – The lower energy bound of the background map.
- **hi_en** (*Quantity*) – The upper energy bound of the background map.
- **inst** (*str*) – The instrument of the background map (if applicable).

Returns
The requested background map file path.

assess_process_obs(*obs_info*)

A slightly unusual method which will allow the eRASS1DE mission to assess the information on a particular observation that has been put together by an Archive (the archive assembles it because sometimes this detailed information only becomes available at the first stages of processing), and make a decision on whether that particular observation-instrument should be processed further for scientific use.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
obs_info (*dict*) – The multi-level dictionary containing available observation information for an observation.

ident_to_obsid(*ident*)

A slightly unusual abstract method which will allow each mission convert a unique identifier being used in the processing steps to the ObsID (as these unique identifiers will contain the ObsID). This is necessary because XMM, for instance, has processing steps that act on whole ObsIDs (e.g. cifbuild), and processing steps that act on individual sub-exposures of instruments of ObsIDs, so the ID could be '0201903501M1S001'.

Implemented as an abstract method because the unique identifier style may well be different for different missions - many will just always be the ObsID, but we want to be able to have low level control.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
ident (*str*) – The unique identifier used in a particular processing step.

5.2.4 mission.chandra module

class daxa.mission.chandra.**Chandra**(*insts=None*, *save_file_path=None*)

Bases: *BaseMission*

The mission class for Chandra observations. The available observation information is fetched from the HEASArc CHANMASTER table, and data are downloaded from the HEASArc https access to their FTP server. Proprietary data are not currently supported by this class.

This will be the only Chandra mission class, as they do not appear to take data during slewing (like XMM and NuSTAR).

Selecting particular instruments will effectively be putting a filter on the observations, as due to the design of Chandra ACIS and HRC instruments cannot observe simultaneously.

Functionally, this class is very similar to NuSTARPointed.

Parameters

- **insts** (*List[str]/str*) – The instruments that the user is choosing to download/process data from. You can pass either a single string value or a list of strings. They may include ACIS-I, ACIS-S, HRC-I, and HRC-S.
- **save_file_path** (*str*) – An optional argument that can use a DAXA mission class save file to recreate the state of a previously defined mission (the same filters having been applied etc.)

property name

Property getter for the name of this mission

Returns
The mission name.

Return type
str

`property chosen_instruments`

Property getter for the names of the currently selected instruments associated with this mission which will be processed into an archive by DAXA functions. Overwritten here because there are custom behaviours for Chandra, as it has one instrument per ObsID.

Returns
A list of instrument names

Return type
List[str]

`property coord_frame`

Property getter for the coordinate frame of the RA-Decs of the observations of this mission. Chandra is the only one so far to actually use ICRS! (or at least it does for its source lists and image WCS headers).

Returns
The coordinate frame of the RA-Dec.

Return type
BaseRADecFrame

`property fov`

Property getter for the approximate field of view set for this mission. This is the radius/half-side-length of the field of view. In cases where the field of view is not square/circular, it is the half-side-length of the longest side.

Returns
The approximate field of view(s) for the mission's instrument(s). In cases with multiple instruments then this may be a dictionary, with keys being instrument names.

Return type
Union[Quantity, dict]

`property id_regex`

Property getter for the regular expression (regex) pattern for observation IDs of this mission.

Returns
The regex pattern for observation IDs.

Return type
str

`property all_obs_info`

A property getter that returns the base dataframe containing information about all the observations available for an instance of a mission class.

Returns
A pandas dataframe with (at minimum) the following columns; 'ra', 'dec', 'ObsID', 'science_usable', 'start', 'duration'

Return type
pd.DataFrame

download(*num_cores=1*, *credentials=None*, *download_products=True*)

A method to acquire and download the pointed Chandra data that have not been filtered out (if a filter has been applied, otherwise all data will be downloaded). Instruments specified by the chosen_instruments property will be downloaded, which is set either on declaration of the class instance or by passing a new value to the chosen_instruments property.

If you're using DAXA only for data acquisition, and wish to use CIAO scripts for reprocessing (e.g. 'chandra_repro'), then set 'download_products=True'.

Parameters

- **num_cores** (*int*) – The number of cores that can be used to parallelise downloading the data. Default is the value of NUM_CORES, specified in the configuration file, or if that hasn't been set then 90% of the cores available on the current machine.
- **credentials** (*dict/str*) – The path to an ini file containing credentials, a dictionary containing 'user' and 'password' entries, or a dictionary of ObsID top level keys, with 'user' and 'password' entries for providing different credentials for different observations.
- **download_products** (*bool*) – Whether the 'standard' Chandra data structure should be downloaded (i.e. with 'primary' and 'secondary' directories, including pre-generated images. The default is True, if set to False only event lists will be downloaded.

assess_process_obs(*obs_info*)

A slightly unusual method which will allow the Chandra mission to assess the information on a particular observation that has been put together by an Archive (the archive assembles it because sometimes this detailed information only becomes available at the first stages of processing), and make a decision on whether that particular observation-instrument should be processed further for scientific use.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters

obs_info (*dict*) – The multi-level dictionary containing available observation information for an observation.

ident_to_obsid(*ident*)

A slightly unusual abstract method which will allow each mission convert a unique identifier being used in the processing steps to the ObsID (as these unique identifiers will contain the ObsID). This is necessary because XMM, for instance, has processing steps that act on whole ObsIDs (e.g. cifbuild), and processing steps that act on individual sub-exposures of instruments of ObsIDs, so the ID could be '0201903501M1S001'.

Implemented as an abstract method because the unique identifier style may well be different for different missions - many will just always be the ObsID, but we want to be able to have low level control.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters

ident (*str*) – The unique identifier used in a particular processing step.

5.2.5 mission.nustar module

class daxa.mission.nustar.**NuSTARPointed**(*insts=None*, *save_file_path=None*)

Bases: *BaseMission*

The mission class for pointed NuSTAR observations (i.e. slewing observations are NOT included in the data accessed and collected by instances of this class), nor are observations for which the spacecraft mode was 'STELLAR'. The available observation information is fetched from the HEASArc NuMASTER table, and data are downloaded from the HEASArc https access to their FTP server. Proprietary data are not currently supported by this class.

Parameters

- **insts** (*List[str]/str*) – The instruments that the user is choosing to download/process data from. You can pass either a single string value or a list of strings. They may include FPMA and FPMB (the default is both).
- **save_file_path** (*str*) – An optional argument that can use a DAXA mission class save file to recreate the state of a previously defined mission (the same filters having been applied etc.)

property name

Property getter for the name of this mission

Returns
The mission name.

Return type
str

property coord_frame

Property getter for the coordinate frame of the RA-Decs of the observations of this mission.

Returns
The coordinate frame of the RA-Dec.

Return type
BaseRADecFrame

property id_regex

Property getter for the regular expression (regex) pattern for observation IDs of this mission.

Returns
The regex pattern for observation IDs.

Return type
str

property fov

Property getter for the approximate field of view set for this mission. This is the radius/half-side-length of the field of view. In cases where the field of view is not square/circular, it is the half-side-length of the longest side.

Returns
The approximate field of view(s) for the mission's instrument(s). In cases with multiple instruments then this may be a dictionary, with keys being instrument names.

Return type
Union[Quantity, dict]

property all_obs_info

A property getter that returns the base dataframe containing information about all the observations available for an instance of a mission class.

Returns
A pandas dataframe with (at minimum) the following columns; 'ra', 'dec', 'ObsID', 'science_usable', 'start', 'duration'

Return type
pd.DataFrame

download(*num_cores=1*, *credentials=None*, *download_products=True*)

A method to acquire and download the pointed NuSTAR data that have not been filtered out (if a filter has been applied, otherwise all data will be downloaded). Instruments specified by the chosen_instruments property will be downloaded, which is set either on declaration of the class instance or by passing a new value to the chosen_instruments property.

Parameters

- **num_cores** (*int*) – The number of cores that can be used to parallelise downloading the data. Default is the value of NUM_CORES, specified in the configuration file, or if that hasn't been set then 90% of the cores available on the current machine.
- **credentials** (*dict/str*) – The path to an ini file containing credentials, a dictionary containing 'user' and 'password' entries, or a dictionary of ObsID top level keys, with 'user' and 'password' entries for providing different credentials for different observations.
- **download_products** (*bool*) – This controls whether the data downloaded include the pre-processed event lists and images stored by HEASArc, or whether they are the original raw event lists. Default is True.

assess_process_obs(*obs_info*)

A slightly unusual method which will allow the NuSTARPointed mission to assess the information on a particular observation that has been put together by an Archive (the archive assembles it because sometimes this detailed information only becomes available at the first stages of processing), and make a decision on whether that particular observation-instrument should be processed further for scientific use.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
obs_info (*dict*) – The multi-level dictionary containing available observation information for an observation.

ident_to_obsid(*ident*)

A slightly unusual abstract method which will allow each mission convert a unique identifier being used in the processing steps to the ObsID (as these unique identifiers will contain the ObsID). This is necessary because XMM, for instance, has processing steps that act on whole ObsIDs (e.g. cifbuild), and processing steps that act on individual sub-exposures of instruments of ObsIDs, so the ID could be '0201903501M1S001'.

Implemented as an abstract method because the unique identifier style may well be different for different missions - many will just always be the ObsID, but we want to be able to have low level control.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
ident (*str*) – The unique identifier used in a particular processing step.

5.2.6 mission.rosat module

class daxa.mission.rosat.**ROSATPointed**(*insts=None*, *save_file_path=None*)

Bases: *BaseMission*

The mission class for ROSAT Pointed observations, taken after the initial all-sky survey. This mission includes the follow-up observations taken to complete the all-sky survey in pointed mode towards the end of the ROSAT lifetime. This mission class pulls observation information from the HEASArc ROSMASTER table, and downloads data from the HEASArc website.

NOTE: Follow-up All-Sky observations are marked as being taken with 'PSPC' (rather than a specific PSPC-C or B) in the ROSMASTER table, but they were actually taken with PSPC-B, so DAXA corrects the entries on acquisition of the table from HEASArc.

Another mission class is available for ROSAT All-Sky Survey observations, specifically the ones taken in slewing mode with PSPC-C at the beginning of the ROSAT mission.

Parameters

- **insts** (*List[str]/str*) – The instruments that the user is choosing to download/process data from. You can pass either a single string value or a list of strings. They may include PSPCB, PSPCC, and HRI.
- **save_file_path** (*str*) – An optional argument that can use a DAXA mission class save file to recreate the state of a previously defined mission (the same filters having been applied etc.)

property name

Property getter for the name of this mission

Returns
The mission name.

Return type
str

property chosen_instruments

Property getter for the names of the currently selected instruments associated with this mission which will be processed into an archive by DAXA functions. Overwritten here because there are custom behaviours for ROSATPointed, as it has one instrument per ObsID.

Returns
A list of instrument names.

Return type
List[str]

property coord_frame

Property getter for the coordinate frame of the RA-Decs of the observations of this mission. Not completely certain that FK5 is the correct frame for RASS, but a processed image downloaded from HEASArc used FK5 as the reference frame for its WCS.

Returns
The coordinate frame of the RA-Dec.

Return type
BaseRADecFrame

property id_regex

Property getter for the regular expression (regex) pattern for observation IDs of this mission.

Returns
The regex pattern for observation IDs.

Return type
str

`property fov`

Property getter for the approximate field of view set for this mission. This is the radius/half-side-length of the field of view. In cases where the field of view is not square/circular, it is the half-side-length of the longest side.

Returns
The approximate field of view(s) for the mission's instrument(s). In cases with multiple instruments then this may be a dictionary, with keys being instrument names.

Return type
Union[Quantity, dict]

`property all_obs_info`

A property getter that returns the base dataframe containing information about all the observations available for an instance of a mission class.

Returns
A pandas dataframe with (at minimum) the following columns; 'ra', 'dec', 'ObsID', 'science_usable', 'start', 'duration'

Return type
pd.DataFrame

`download`(*num_cores=1*, *download_products=True*)

A method to acquire and download the ROSAT pointed data that have not been filtered out (if a filter has been applied, otherwise all data will be downloaded).

Proprietary data is not a relevant concept for ROSAT at this point, so no option to provide credentials is provided here as it is in some other mission classes.

Parameters

- **num_cores** (*int*) – The number of cores that can be used to parallelise downloading the data. Default is the value of NUM_CORES, specified in the configuration file, or if that hasn't been set then 90% of the cores available on the current machine.
- **download_products** (*bool*) – This controls whether the HEASArc-published images and exposure maps are downloaded alongside the event lists and attitude files. Setting this to True will download the images/exposure maps. The default is True.

`assess_process_obs`(*obs_info*)

A slightly unusual abstract method which will allow each mission to assess the information on a particular observation that has been put together by an Archive (the archive assembles it because sometimes this detailed information only becomes available at the first stages of processing), and make a decision on whether that particular observation-instrument-subexposure (for missions like XMM) should be processed further for scientific use.

Implemented as an abstract method because the information and decision-making process will likely be different for every mission.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
obs_info (*dict*) – The multi-level dictionary containing available observation information for an observation.

`ident_to_obsid`(*ident*)

A slightly unusual abstract method which will allow each mission convert a unique identifier being used in the processing steps to the ObsID (as these unique identifiers will contain the ObsID). This is necessary because XMM, for instance, has processing steps that act on whole ObsIDs (e.g. cifbuild), and processing steps that act on individual sub-exposures of instruments of ObsIDs, so the ID could be '0201903501M1S001'.

Implemented as an abstract method because the unique identifier style may well be different for different missions - many will just always be the ObsID, but we want to be able to have low level control.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
ident (*str*) – The unique identifier used in a particular processing step.

`class daxa.mission.rosat.`**ROSATAllSky**(*save_file_path=None*)

Bases: *BaseMission*

The mission class for ROSAT All-Sky Survey (RASS) observations. The available observation information is fetched from the HEASArc RASSMASTER table, and data are downloaded from the HEASArc https access to their FTP server. Only data from the initial scanning phase of RASS will be fetched by this class, not the follow-up pointed mode observations used to complete the survey towards the end of the ROSAT mission.

Another mission class is available for pointed ROSAT observations.

No instrument choice is offered for this mission class because all RASS observations in the scanning portion of the survey were taken with PSPC-C.

Parameters
save_file_path (*str*) – An optional argument that can use a DAXA mission class save file to recreate the state of a previously defined mission (the same filters having been applied etc.)

`property name`

Property getter for the name of this mission

Returns
The mission name.

Return type
str

`property coord_frame`

Property getter for the coordinate frame of the RA-Decs of the observations of this mission. Not completely certain that FK5 is the correct frame for RASS, but a processed image downloaded from HEASArc used FK5 as the reference frame for its WCS.

Returns
The coordinate frame of the RA-Dec.

Return type
BaseRADecFrame

`property id_regex`

Property getter for the regular expression (regex) pattern for observation IDs of this mission.

Returns
The regex pattern for observation IDs.

Return type
str

`property fov`

Property getter for the approximate field of view set for this mission. This is the radius/half-side-length of the field of view. In cases where the field of view is not square/circular, it is the half-side-length of the longest side.

Returns
The approximate field of view(s) for the mission's instrument(s). In cases with multiple instruments then this may be a dictionary, with keys being instrument names.

Return type
Union[Quantity, dict]

`property all_obs_info`

A property getter that returns the base dataframe containing information about all the observations available for an instance of a mission class.

Returns
A pandas dataframe with (at minimum) the following columns; 'ra', 'dec', 'ObsID', 'science_usable', 'start', 'duration'

Return type
pd.DataFrame

`download`(*num_cores=1*, *download_products=True*)

A method to acquire and download the ROSAT All-Sky Survey data that have not been filtered out (if a filter has been applied, otherwise all data will be downloaded).

Proprietary data is not a relevant concept for RASS, so no option to provide credentials is provided here as it is in some other mission classes.

Parameters

- **num_cores** (*int*) – The number of cores that can be used to parallelise downloading the data. Default is the value of NUM_CORES, specified in the configuration file, or if that hasn't been set then 90% of the cores available on the current machine.
- **download_products** (*bool*) – This controls whether the HEASArc-published images and exposure maps are downloaded alongside the event lists and attitude files. Setting this to True will download the images/exposure maps. The default is True.

`assess_process_obs`(*obs_info*)

A slightly unusual abstract method which will allow each mission to assess the information on a particular observation that has been put together by an Archive (the archive assembles it because sometimes this detailed information only becomes available at the first stages of processing), and make a decision on whether that particular observation-instrument-subexposure (for missions like XMM) should be processed further for scientific use.

Implemented as an abstract method because the information and decision-making process will likely be different for every mission.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
obs_info (*dict*) – The multi-level dictionary containing available observation information for an observation.

ident_to_obsid(*ident*)

A slightly unusual abstract method which will allow each mission convert a unique identifier being used in the processing steps to the ObsID (as these unique identifiers will contain the ObsID). This is necessary because XMM, for instance, has processing steps that act on whole ObsIDs (e.g. cifbuild), and processing steps that act on individual sub-exposures of instruments of ObsIDs, so the ID could be '0201903501M1S001'.

Implemented as an abstract method because the unique identifier style may well be different for different missions - many will just always be the ObsID, but we want to be able to have low level control.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
ident (*str*) – The unique identifier used in a particular processing step.

5.2.7 mission.swift module

class daxa.mission.swift.**Swift**(*insts=None*, *save_file_path=None*)

Bases: *BaseMission*

The mission class for observations by the Neil Gehrels Swift Observatory. The available observation information is fetched from the HEASArc SWIFTMASTR table, and data are downloaded from the HEASArc https access to their FTP server. Proprietary data are not currently supported by this class.

Parameters

- **insts** (*List[str]/str*) – The instruments that the user is choosing to download/process data from. You can pass either a single string value or a list of strings. They may include XRT, BAT, and UVOT (the default is both XRT and BAT).
- **save_file_path** (*str*) – An optional argument that can use a DAXA mission class save file to recreate the state of a previously defined mission (the same filters having been applied etc.)

property name

Property getter for the name of this mission

Returns
The mission name.

Return type
str

property coord_frame

Property getter for the coordinate frame of the RA-Decs of the observations of this mission.

Returns
The coordinate frame of the RA-Dec.

Return type
BaseRADecFrame

property id_regex

Property getter for the regular expression (regex) pattern for observation IDs of this mission.

Returns
The regex pattern for observation IDs.

Return type
str

property fov

Property getter for the approximate field of view set for this mission. This is the radius/half-side-length of the field of view. In cases where the field of view is not square/circular, it is the half-side-length of the longest side.

Returns
The approximate field of view(s) for the mission's instrument(s). In cases with multiple instruments then this may be a dictionary, with keys being instrument names.

Return type
Union[Quantity, dict]

property all_obs_info

A property getter that returns the base dataframe containing information about all the observations available for an instance of a mission class.

Returns
A pandas dataframe with (at minimum) the following columns; 'ra', 'dec', 'ObsID', 'science_usable', 'start', 'duration'

Return type
pd.DataFrame

download(*num_cores=1*, *download_products=True*)

A method to acquire and download the Swift data that have not been filtered out (if a filter has been applied, otherwise all data will be downloaded). Instruments specified by the chosen_instruments property will be downloaded, which is set either on declaration of the class instance or by passing a new value to the chosen_instruments property.

Parameters

- **num_cores** (*int*) – The number of cores that can be used to parallelise downloading the data. Default is the value of NUM_CORES, specified in the configuration file, or if that hasn't been set then 90% of the cores available on the current machine.
- **download_products** (*bool*) – This controls whether the data downloaded include the pre-processed event lists and images stored by HEASArc, or whether they are the original raw event lists. Default is True.

assess_process_obs(*obs_info*)

A slightly unusual method which will allow the Swift mission to assess the information on a particular observation that has been put together by an Archive (the archive assembles it because sometimes this detailed information only becomes available at the first stages of processing), and make a decision on whether that particular observation-instrument should be processed further for scientific use.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
obs_info (*dict*) – The multi-level dictionary containing available observation information for an observation.

ident_to_obsid(*ident*)

A slightly unusual abstract method which will allow each mission convert a unique identifier being used

in the processing steps to the ObsID (as these unique identifiers will contain the ObsID). This is necessary because XMM, for instance, has processing steps that act on whole ObsIDs (e.g. cifbuild), and processing steps that act on individual sub-exposures of instruments of ObsIDs, so the ID could be '0201903501M1S001'.

Implemented as an abstract method because the unique identifier style may well be different for different missions - many will just always be the ObsID, but we want to be able to have low level control.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
ident (*str*) – The unique identifier used in a particular processing step.

5.2.8 mission.suzaku module

class daxa.mission.suzaku.**Suzaku**(*insts=None*, *save_file_path=None*)

Bases: *BaseMission*

The mission class for Suzaku observations, specifically those from the XIS instruments, as XRS' cooling system was damaged soon after launch, and HXD was not an imaging instrument. The available observation information is fetched from the HEASArc SUZAMASTER table, and data are downloaded from the HEASArc https access to their FTP server.

Parameters

- **insts** (*List[str]/str*) – The instruments that the user is choosing to download/process data from. You can pass either a single string value or a list of strings. They may include XIS0, XIS1, XIS2, and XIS3 (the default is all of them).
- **save_file_path** (*str*) – An optional argument that can use a DAXA mission class save file to recreate the state of a previously defined mission (the same filters having been applied etc.)

property name

Property getter for the name of this mission

Returns
The mission name.

Return type
str

property coord_frame

Property getter for the coordinate frame of the RA-Decs of the observations of this mission.

Returns
The coordinate frame of the RA-Dec.

Return type
BaseRADecFrame

property id_regex

Property getter for the regular expression (regex) pattern for observation IDs of this mission.

Returns
The regex pattern for observation IDs.

Return type
str

property fov

Property getter for the approximate field of view set for this mission. This is the radius/half-side-length of the field of view. In cases where the field of view is not square/circular, it is the half-side-length of the longest side.

Returns
The approximate field of view(s) for the mission's instrument(s). In cases with multiple instruments then this may be a dictionary, with keys being instrument names.

Return type
Union[Quantity, dict]

property all_obs_info

A property getter that returns the base dataframe containing information about all the observations available for an instance of a mission class.

Returns
A pandas dataframe with (at minimum) the following columns; 'ra', 'dec', 'ObsID', 'science_usable', 'start', 'duration'

Return type
pd.DataFrame

download(*num_cores=1*, *download_products=True*)

A method to acquire and download the Suzaku data that have not been filtered out (if a filter has been applied, otherwise all data will be downloaded). Instruments specified by the chosen_instruments property will be downloaded, which is set either on declaration of the class instance or by passing a new value to the chosen_instruments property.

Parameters

- **num_cores** (*int*) – The number of cores that can be used to parallelise downloading the data. Default is the value of NUM_CORES, specified in the configuration file, or if that hasn't been set then 90% of the cores available on the current machine.
- **download_products** (*bool*) – This controls whether the data downloaded include the pre-processed event lists and images stored by HEASArc, or whether they are the original raw event lists. Default is True.

assess_process_obs(*obs_info*)

A slightly unusual method which will allow the Suzaku mission to assess the information on a particular observation that has been put together by an Archive (the archive assembles it because sometimes this detailed information only becomes available at the first stages of processing), and make a decision on whether that particular observation-instrument should be processed further for scientific use.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
obs_info (*dict*) – The multi-level dictionary containing available observation information for an observation.

ident_to_obsid(*ident*)

A slightly unusual abstract method which will allow each mission convert a unique identifier being used in the processing steps to the ObsID (as these unique identifiers will contain the ObsID). This is necessary because XMM, for instance, has processing steps that act on whole ObsIDs (e.g. cifbuild), and processing steps that act on individual sub-exposures of instruments of ObsIDs, so the ID could be '0201903501M1S001'.

Implemented as an abstract method because the unique identifier style may well be different for different missions - many will just always be the ObsID, but we want to be able to have low level control.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
ident (*str*) – The unique identifier used in a particular processing step.

5.2.9 mission.asca module

class daxa.mission.asca.**ASCA**(*insts=None*, *save_file_path=None*)

Bases: *BaseMission*

The mission class for ASCA observations, both from the GIS AND SIS instruments. The available observation information is fetched from the HEASArc ASCAMASTER table, and data are downloaded from the HEASArc https access to their FTP server.

Parameters

- **insts** (*List[str]/str*) – The instruments that the user is choosing to download/process data from. You can pass either a single string value or a list of strings. They may include SIS0, SIS1, GIS2, and GIS3 (the default is all of them).
- **save_file_path** (*str*) – An optional argument that can use a DAXA mission class save file to recreate the state of a previously defined mission (the same filters having been applied etc.)

property name

Property getter for the name of this mission

Returns
The mission name.

Return type
str

property coord_frame

Property getter for the coordinate frame of the RA-Decs of the observations of this mission.

Returns
The coordinate frame of the RA-Dec.

Return type
BaseRADecFrame

property id_regex

Property getter for the regular expression (regex) pattern for observation IDs of this mission.

Returns
The regex pattern for observation IDs.

Return type
str

property fov

Property getter for the approximate field of view set for this mission. This is the radius/half-side-length of the field of view. In cases where the field of view is not square/circular, it is the half-side-length of the longest side.

Returns
The approximate field of view(s) for the mission's instrument(s). In cases with multiple instruments then this may be a dictionary, with keys being instrument names.

Return type
Union[Quantity, dict]

`property all_obs_info`

A property getter that returns the base dataframe containing information about all the observations available for an instance of a mission class.

Returns
A pandas dataframe with (at minimum) the following columns; 'ra', 'dec', 'ObsID', 'science_usable', 'start', 'duration'

Return type
pd.DataFrame

`download`(*num_cores=1*, *download_products=True*)

A method to acquire and download the ASCA data that have not been filtered out (if a filter has been applied, otherwise all data will be downloaded). Instruments specified by the chosen_instruments property will be downloaded, which is set either on declaration of the class instance or by passing a new value to the chosen_instruments property.

Parameters

- **num_cores** (*int*) – The number of cores that can be used to parallelise downloading the data. Default is the value of NUM_CORES, specified in the configuration file, or if that hasn't been set then 90% of the cores available on the current machine.
- **download_products** (*bool*) – This controls whether the data downloaded include the pre-processed event lists and images stored by HEASArc, or whether they are the original raw event lists. Default is True.

`get_evt_list_path`(*obs_id*, *inst=None*)

A get method that provides the path to a downloaded pre-generated event list for the current mission (if available). This method will not work if pre-processed data have not been downloaded.

Parameters

- **obs_id** (*str*) – The ObsID of the event list.
- **inst** (*str*) – The instrument of the event list (if applicable).

Returns
The requested event list path.

Return type
str

`assess_process_obs`(*obs_info*)

A slightly unusual method which will allow the ASCA mission to assess the information on a particular observation that has been put together by an Archive (the archive assembles it because sometimes this detailed information only becomes available at the first stages of processing), and make a decision on whether that particular observation-instrument should be processed further for scientific use.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
obs_info (*dict*) – The multi-level dictionary containing available observation information for an observation.

ident_to_obsid(*ident*)

A slightly unusual abstract method which will allow each mission convert a unique identifier being used in the processing steps to the ObsID (as these unique identifiers will contain the ObsID). This is necessary because XMM, for instance, has processing steps that act on whole ObsIDs (e.g. cifbuild), and processing steps that act on individual sub-exposures of instruments of ObsIDs, so the ID could be '0201903501M1S001'.

Implemented as an abstract method because the unique identifier style may well be different for different missions - many will just always be the ObsID, but we want to be able to have low level control.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
ident (*str*) – The unique identifier used in a particular processing step.

5.2.10 mission.integral module

class daxa.mission.integral.**INTEGRALPointed**(*insts=None*, *save_file_path=None*)

Bases: *BaseMission*

The mission class for pointed observations by the INTErnational Gamma-Ray Astrophysics Laboratory (INTEGRAL); i.e. observations taken when the spacecraft isn't slewing, and is not in an engineering mode (and is public). The available observation information is fetched from the HEASArc INTSCWPUB table, and data are downloaded from the HEASArc https access to their FTP server. Proprietary data are not currently supported by this class.

NOTE - This class treats Science Window IDs as 'obs ids', though observation IDs are a separate concept in the HEASArc table at least. The way DAXA is set up however, science window IDs are a closer analogy to the ObsIDs used by the rest of the missions.

Parameters

- **insts** (*List[str]/str*) – The instruments that the user is choosing to download/process data from. You can pass either a single string value or a list of strings. They may include JEMX1, JEMX2, ISGRI, PICsIT, and SPI (the default is JEMX1, JEMX2, and ISGRI). OMC is not supported by DAXA.
- **save_file_path** (*str*) – An optional argument that can use a DAXA mission class save file to recreate the state of a previously defined mission (the same filters having been applied etc.)

property name

Property getter for the name of this mission

Returns
The mission name.

Return type
str

property coord_frame

Property getter for the coordinate frame of the RA-Decs of the observations of this mission.

Returns
The coordinate frame of the RA-Dec.

Return type
BaseRADecFrame

property id_regex

Property getter for the regular expression (regex) pattern for observation IDs of this mission.

Returns
The regex pattern for observation IDs.

Return type
str

property fov

Property getter for the approximate field of view set for this mission. This is the radius/half-side-length of the field of view. In cases where the field of view is not square/circular, it is the half-side-length of the longest side.

Returns
The approximate field of view(s) for the mission's instrument(s). In cases with multiple instruments then this may be a dictionary, with keys being instrument names.

Return type
Union[Quantity, dict]

property all_obs_info

A property getter that returns the base dataframe containing information about all the observations available for an instance of a mission class.

Returns
A pandas dataframe with (at minimum) the following columns; 'ra', 'dec', 'ObsID', 'science_usable', 'start', 'duration'

Return type
pd.DataFrame

download(*num_cores=1*, *download_products=True*)

A method to acquire and download the pointed INTEGRAL data that have not been filtered out (if a filter has been applied, otherwise all data will be downloaded). Instruments specified by the chosen_instruments property will be downloaded, which is set either on declaration of the class instance or by passing a new value to the chosen_instruments property.

There is no download_processed option here because there are no pre-generated products (i.e. images/spectra) to download, though the event lists etc. have gone through some form of processing.

Parameters

- **num_cores** (*int*) – The number of cores that can be used to parallelise downloading the data. Default is the value of NUM_CORES, specified in the configuration file, or if that hasn't been set then 90% of the cores available on the current machine.
- **download_products** (*bool*) – PRESENT FOR COMPATIBILITY WITH OTHER DAXA TASKS - the INTEGRAL archive does not provide pre-processed products for download.

assess_process_obs(*obs_info*)

A slightly unusual method which will allow the INTEGRALPointed mission to assess the information on a particular observation that has been put together by an Archive (the archive assembles it because sometimes this detailed information only becomes available at the first stages of processing), and make a decision on whether that particular observation-instrument should be processed further for scientific use.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
obs_info (*dict*) – The multi-level dictionary containing available observation information for an observation.

ident_to_obsid(*ident*)

A slightly unusual abstract method which will allow each mission convert a unique identifier being used in the processing steps to the ObsID (as these unique identifiers will contain the ObsID). This is necessary because XMM, for instance, has processing steps that act on whole ObsIDs (e.g. cifbuild), and processing steps that act on individual sub-exposures of instruments of ObsIDs, so the ID could be '0201903501M1S001'.

Implemented as an abstract method because the unique identifier style may well be different for different missions - many will just always be the ObsID, but we want to be able to have low level control.

This method should never need to be triggered by the user, as it will be called automatically when detailed observation information becomes available to the Archive.

Parameters
ident (*str*) – The unique identifier used in a particular processing step.

5.3 process

5.3.1 process.simple

daxa.process.simple.**full_process_xmm**(*obs_archive*, *lo_en=None*, *hi_en=None*, *process_unscheduled=True*, *find_mos_anom_state=False*, *num_cores=1*, *timeout=None*)

This is a convenience function that will fully process and prepare XMM data in an archive using the default configuration settings of all the cleaning steps. If you wish to exercise finer grained control over the processing of your data then you can copy the steps of this function and alter the various parameter values.

Parameters

- **obs_archive** (Archive) – An archive object that contains at least one XMM mission to be processed.
- **lo_en** (*Quantity*) – If an energy filter should be applied to the final cleaned event lists, this is the lower energy bound. The default is None, in which case NO ENERGY FILTER is applied.
- **hi_en** (*Quantity*) – If an energy filter should be applied to the final cleaned event lists, this is the upper energy bound. The default is None, in which case NO ENERGY FILTER is applied.
- **process_unscheduled** (*bool*) – Should unscheduled sub-exposures be processed and included in the final event lists. The default is True.
- **find_mos_anom_state** (*bool*) – Whether the emanom task should be run to search for anomolous states of the MOS cameras. This is set to False by default, and you should be aware that I have not found it to be particularly reliable with the default settings, it tends to remove good chips.
- **num_cores** (*int*) – The number of cores that can be used by the processing functions. The default is set to the DAXA NUM_CORES parameter, which is configured to be 90% of the system's cores.
- **timeout** (*Quantity*) – The amount of time each individual process is allowed to run for, the default is None. Please note that this is not a timeout for the entire processing stack, but a

timeout for the individual processes of each stage, whether they are at the ObsID, ObsID-Inst, or ObsID-Inst-Subexposure level of granularity.

daxa.process.simple.**full_process_erosita**(*obs_archive*, *lo_en=None*, *hi_en=None*, *num_cores=1*, *timeout=None*)

This is a convenience function that will fully process and prepare eROSITA data in an archive using the default configuration settings of all the cleaning steps. If you wish to exercise finer grained control over the processing of your data then you can copy the steps of this function and alter the various parameter values.

Parameters

- **obs_archive** (Archive) – An archive object that contains at least one eROSITA mission to be processed.
- **lo_en** (*Quantity*) – If an energy filter should be applied to the final cleaned event lists, this is the lower energy bound. The default is None, in which case NO ENERGY FILTER is applied.
- **hi_en** (*Quantity*) – If an energy filter should be applied to the final cleaned event lists, this is the upper energy bound. The default is None, in which case NO ENERGY FILTER is applied.
- **num_cores** (*int*) – The number of cores that can be used by the processing functions. The default is set to the DAXA NUM_CORES parameter, which is configured to be 90% of the system's cores.
- **timeout** (*Quantity*) – The amount of time each individual process is allowed to run for, the default is None. Please note that this is not a timeout for the entire processing stack, but a timeout for the individual processes of each stage.

5.3.2 Mission Specific

process.xmm

process.xmm.assemble

daxa.process.xmm.assemble.**epchain**(*obs_archive*, *process_unscheduled=True*, *num_cores=1*, *disable_progress=False*, *timeout=None*)

This function runs the epchain SAS process on XMM missions in the passed archive, which assembles the PN-specific ODFs into combined photon event lists - rather than the per CCD files that existed before. A run of epchain for out of time (OOT) events is also performed as part of this function call. The epchain manual can be found here (https://xmm-tools.cosmos.esa.int/external/sas/current/doc/epchain.pdf) and gives detailed explanations of the process.

Per the advice of the SAS epchain manual, the OOT event list epchain call is performed first, and its intermediate files are saved and then used for the normal call to epchain.

Parameters

- **obs_archive** (Archive) – An Archive instance containing XMM mission instances with PN observations for which epchain should be run. This function will fail if no XMM missions are present in the archive.
- **process_unscheduled** (*bool*) – Whether this function should also process sub-exposures marked 'U', for unscheduled. Default is True, in which case they will be processed.
- **num_cores** (*int*) – The number of cores to use, default is set to 90% of available.

- **disable_progress** (*bool*) – Setting this to true will turn off the SAS generation progress bar.
- **timeout** (*Quantity*) – The amount of time each individual process is allowed to run for, the default is None. Please note that this is not a timeout for the entire epchain process, but a timeout for individual ObsID-subexposure processes.

Returns

Information required by the SAS decorator that will run commands. Top level keys of any dictionaries are internal DAXA mission names, next level keys are ObsIDs. The return is a tuple containing a) a dictionary of bash commands, b) a dictionary of final output paths to check, c) a dictionary of extra info (in this case obs and analysis dates), d) a generation message for the progress bar, e) the number of cores allowed, and f) whether the progress bar should be hidden or not.

Return type

Tuple[dict, dict, dict, str, int, bool, Quantity]

daxa.process.xmm.assemble.**emchain**(*obs_archive*, *process_unscheduled=True*, *num_cores=1*, *disable_progress=False*, *timeout=None*)

This function runs the emchain SAS process on XMM missions in the passed archive, which assembles the MOS-specific ODFs into combined photon event lists - rather than the per CCD files that existed before. The emchain manual can be found here (https://xmm-tools.cosmos.esa.int/external/sas/current/doc/emchain.pdf) and gives detailed explanations of the process.

The DAXA wrapper does not allow emchain to automatically loop through all the sub-exposures for a given ObsID-MOSX combo, but rather creates a separate process call for each of them. This allows for greater parallelisation (if on a system with a significant core count), but also allows the same level of granularity in the logging of processing of different sub-exposures as in DAXA's epchain implementation.

The particular CCDs to be processed are not specified in emchain, unlike in epchain, because it can sometimes have unintended consequences. For instance processing a MOS observation in FastUncompressed mode, with timing on CCD 1 and imaging everywhere else, can cause emchain to fail (even though no actual failure occurs) because the submode is set to Unknown, rather than FastUncompressed.

Parameters

- **obs_archive** (Archive) – An Archive instance containing XMM mission instances with MOS observations for which emchain should be run. This function will fail if no XMM missions are present in the archive.
- **process_unscheduled** (*bool*) – Whether this function should also process sub-exposures marked 'U', for unscheduled. Default is True, in which case they will be processed.
- **num_cores** (*int*) – The number of cores to use, default is set to 90% of available.
- **disable_progress** (*bool*) – Setting this to true will turn off the SAS generation progress bar.
- **timeout** (*Quantity*) – The amount of time each individual process is allowed to run for, the default is None. Please note that this is not a timeout for the entire emchain process, but a timeout for individual ObsID-subexposure processes.

Returns

Information required by the SAS decorator that will run commands. Top level keys of any dictionaries are internal DAXA mission names, next level keys are ObsIDs. The return is a tuple containing a) a dictionary of bash commands, b) a dictionary of final output paths to check, c) a dictionary of extra info (in this case obs and analysis dates), d) a generation message for the progress bar, e) the number of cores allowed, and f) whether the progress bar should be hidden or not.

Return type
Tuple[dict, dict, dict, str, int, bool, Quantity]

daxa.process.xmm.assemble.**rgs_events**(*obs_archive*, *process_unscheduled=True*, *num_cores=1*, *disable_progress=False*, *timeout=None*)

This function runs the first step of the SAS RGS processing pipeline, rgsproc. This should prepare the RGS event lists by calibrating and combining the separate CCD event lists into RGS events. This happens separately for RGS1 and RGS2, and for each sub-exposure of the two instruments.

None of the calculations performed in this step should be affected by the choice of source, the first step where the choice of primary source should be taken into consideration is the next step, rgs_angles; though as DAXA processes data to be generally useful we will not define a primary source, that is for the user in the future as the aspect drift calculations can be re-run.

Parameters

- **obs_archive** (Archive) – An Archive instance containing XMM mission instances with RGS observations for which RGS processing should be run. This function will fail if no XMM missions are present in the archive.
- **process_unscheduled** (*bool*) – Whether this function should also process sub-exposures marked 'U', for unscheduled. Default is True, in which case they will be processed.
- **num_cores** (*int*) – The number of cores to use, default is set to 90% of available.
- **disable_progress** (*bool*) – Setting this to true will turn off the SAS generation progress bar.
- **timeout** (*Quantity*) – The amount of time each individual process is allowed to run for, the default is None. Please note that this is not a timeout for the entire rgs_events process, but a timeout for individual ObsID-subexposure processes.

Returns
Information required by the SAS decorator that will run commands. Top level keys of any dictionaries are internal DAXA mission names, next level keys are ObsIDs. The return is a tuple containing a) a dictionary of bash commands, b) a dictionary of final output paths to check, c) a dictionary of extra info (in this case obs and analysis dates), d) a generation message for the progress bar, e) the number of cores allowed, and f) whether the progress bar should be hidden or not.

Return type
Tuple[dict, dict, dict, str, int, bool, Quantity]

daxa.process.xmm.assemble.**rgs_angles**(*obs_archive*, *num_cores=1*, *disable_progress=False*, *timeout=None*)

This function runs the second step of the SAS RGS processing pipeline, rgsproc. This should calculate aspect drift corrections for some 'uninformative' source, and should likely be refined later when these data are used to analyse a specific source. This happens separately for RGS1 and RGS2, and for each sub-exposure of the two instruments.

Parameters

- **obs_archive** (Archive) – An Archive instance containing XMM mission instances with RGS observations for which RGS processing should be run. This function will fail if no XMM missions are present in the archive.
- **num_cores** (*int*) – The number of cores to use, default is set to 90% of available.
- **disable_progress** (*bool*) – Setting this to true will turn off the SAS generation progress bar.

- **timeout** (*Quantity*) – The amount of time each individual process is allowed to run for, the default is None. Please note that this is not a timeout for the entire rgs_events process, but a timeout for individual ObsID-subexposure processes.

Returns

Information required by the SAS decorator that will run commands. Top level keys of any dictionaries are internal DAXA mission names, next level keys are ObsIDs. The return is a tuple containing a) a dictionary of bash commands, b) a dictionary of final output paths to check, c) a dictionary of extra info (in this case obs and analysis dates), d) a generation message for the progress bar, e) the number of cores allowed, and f) whether the progress bar should be hidden or not.

Return type

Tuple[dict, dict, dict, str, int, bool, Quantity]

daxa.process.xmm.assemble.**cleaned_rgs_event_lists**(*obs_archive*, *num_cores=1*, *disable_progress=False*, *timeout=None*)

This function runs the third step of the SAS RGS processing pipeline, rgsproc. Here we filter the events to only those which should be useful for scientific analysis. The attitude and house-keeping GTIs are also applied. This happens separately for RGS1 and RGS2, and for each sub-exposure of the two instruments.

Unfortunately it seems that combining sub-exposure event lists for a given ObsID-Instrument combo is not supported/recommended, combinations of data are generally done after spectrum generation, and even then they don't exactly recommend it - of course spectrum generation doesn't get done in DAXA. As such this function will produce individual event lists for RGS sub-exposures.

Parameters

- **obs_archive** (Archive) – An Archive instance containing XMM mission instances with RGS observations for which RGS processing should be run. This function will fail if no XMM missions are present in the archive.
- **num_cores** (*int*) – The number of cores to use, default is set to 90% of available.
- **disable_progress** (*bool*) – Setting this to true will turn off the SAS generation progress bar.
- **timeout** (*Quantity*) – The amount of time each individual process is allowed to run for, the default is None. Please note that this is not a timeout for the entire rgs_events process, but a timeout for individual ObsID-subexposure processes.

Returns

Information required by the SAS decorator that will run commands. Top level keys of any dictionaries are internal DAXA mission names, next level keys are ObsIDs. The return is a tuple containing a) a dictionary of bash commands, b) a dictionary of final output paths to check, c) a dictionary of extra info (in this case obs and analysis dates), d) a generation message for the progress bar, e) the number of cores allowed, and f) whether the progress bar should be hidden or not.

Return type

Tuple[dict, dict, dict, str, int, bool, Quantity]

daxa.process.xmm.assemble.**cleaned_evt_lists**(*obs_archive*, *lo_en=None*, *hi_en=None*, *pn_filt_expr=('#XMMEA_EP', '(PATTERN <= 4)', '(FLAG .eq. 0)')*, *mos_filt_expr=('#XMMEA_EM', '(PATTERN <= 12)', '(FLAG .eq. 0)')*, *filt_mos_anom_state=False*, *acc_mos_anom_states=('G', 'I', 'U')*, *num_cores=1*, *disable_progress=False*, *timeout=None*)

This function is used to apply the soft-proton filtering (along with any other filtering you may desire, including

the setting of energy limits) to XMM-Newton event lists, resulting in the creation of sets of cleaned event lists which are ready to be analysed (or merged together, if there are multiple exposures for a particular observation-instrument combination).

Parameters

- **obs_archive** (Archive) – An Archive instance containing XMM mission instances for which cleaned event lists should be created. This function will fail if no XMM missions are present in the archive.
- **lo_en** (*Quantity*) – The lower bound of an energy filter to be applied to the cleaned, filtered, event lists. If 'lo_en' is set to an Astropy Quantity, then 'hi_en' must be as well. Default is None, in which case no energy filter is applied.
- **hi_en** (*Quantity*) – The upper bound of an energy filter to be applied to the cleaned, filtered, event lists. If 'hi_en' is set to an Astropy Quantity, then 'lo_en' must be as well. Default is None, in which case no energy filter is applied.
- **pn_filt_expr** (*str/List[str]/Tuple[str]*) – The filter expressions to be applied to EPIC-PN event lists. Either a single string expression can be passed, or a list/tuple of separate expressions, which will be combined using '&&' logic before being used as the expression for evselect. Other expression components can be added during the process of the function, such as GTI filtering and energy filtering.
- **mos_filt_expr** (*str/List[str]/Tuple[str]*) – The filter expressions to be applied to EPIC-MOS event lists. Either a single string expression can be passed, or a list/tuple of separate expressions, which will be combined using '&&' logic before being used as the expression for evselect. Other expression components can be added during the process of the function, such as GTI filtering, energy filtering, and anomalous state CCD filtering.
- **filt_mos_anom_state** (*bool*) – Whether this function should use the results of an 'emanom' run to identify and remove MOS CCDs that are in anomolous states. If 'False' is passed then no such filtering will be applied.
- **acc_mos_anom_states** (*List[str]/str*) – A list/tuple of acceptable MOS CCD status codes found by emanom (status- G is good at all energies, I is intermediate for E<1 keV, B is bad for E<1 keV, O is off, chip not in use, U is undetermined (low band counts <= 0)).
- **num_cores** (*int*) – The number of cores to use, default is set to 90% of available.
- **disable_progress** (*bool*) – Setting this to true will turn off the SAS generation progress bar.
- **timeout** (*Quantity*) – The amount of time each individual process is allowed to run for, the default is None. Please note that this is not a timeout for the entire cleaned_evt_lists process, but a timeout for individual ObsID-Inst-subexposure processes.

Returns

Information required by the SAS decorator that will run commands. Top level keys of any dictionaries are internal DAXA mission names, next level keys are ObsIDs. The return is a tuple containing a) a dictionary of bash commands, b) a dictionary of final output paths to check, c) a dictionary of extra info (in this case obs and analysis dates), d) a generation message for the progress bar, e) the number of cores allowed, and f) whether the progress bar should be hidden or not.

Return type

Tuple[dict, dict, dict, str, int, bool, Quantity]

daxa.process.xmm.assemble.**merge_subexposures**(*obs_archive*, *num_cores=1*, *disable_progress=False*, *timeout=None*)

A function to identify cases where an instrument for a particular XMM observation has multiple sub-exposures,

for which the event lists can be merged. This produces a final event list, which is a combination of the sub-exposures.

For those observation-instrument combinations with only a single exposure, this function will rename the cleaned event list so that the naming convention is comparable to the merged event list naming convention (i.e. sub-exposure identifier will be removed).

Parameters

- **obs_archive** (Archive) – An Archive instance containing XMM mission instances for which cleaned event lists should be created. This function will fail if no XMM missions are present in the archive.
- **num_cores** (*int*) – The number of cores to use, default is set to 90% of available.
- **disable_progress** (*bool*) – Setting this to true will turn off the SAS generation progress bar.
- **timeout** (*Quantity*) – The amount of time each individual process is allowed to run for, the default is None. Please note that this is not a timeout for the entire merge_subexposures process, but a timeout for individual ObsID-Inst processes.

Returns

Information required by the SAS decorator that will run commands. Top level keys of any dictionaries are internal DAXA mission names, next level keys are ObsIDs. The return is a tuple containing a) a dictionary of bash commands, b) a dictionary of final output paths to check, c) a dictionary of extra info (in this case obs and analysis dates), d) a generation message for the progress bar, e) the number of cores allowed, and f) whether the progress bar should be hidden or not.

Return type

Tuple[dict, dict, dict, str, int, bool, Quantity]

process.xmm.check

daxa.process.xmm.check.**emanom**(*obs_archive*, *num_cores=1*, *disable_progress=False*, *timeout=None*)

This function runs the SAS emanom function, which attempts to identify when MOS CCDs are have operated in an 'anomalous' state, where the background at E < 1 keV is strongly enhanced. Data above 2 keV are unaffected, so CCDs in anomalous states used for science where the soft X-rays are unnecessary do not need to be excluded.

The emanom task calculates the (2.5-5.0 keV)/(0.4-0.8 keV) hardness ratio from the corner data to determine whether a chip is in an anomalous state. However, it should be noted that the "anonymous" anomalous state of MOS1 CCD#4 is not always detectable from the unexposed corner data.

This functionality is only usable if you have SAS v19.0.0 or higher - a version check will be performed and a warning raised (though no error will be raised) if you use this function with an earlier SAS version.

Parameters

- **obs_archive** (Archive) – An Archive instance containing XMM mission instances with MOS observations for which emchain should be run. This function will fail if no XMM missions are present in the archive.
- **num_cores** (*int*) – The number of cores to use, default is set to 90% of available.
- **disable_progress** (*bool*) – Setting this to true will turn off the SAS generation progress bar.

- **timeout** (*Quantity*) – The amount of time each individual process is allowed to run for, the default is None. Please note that this is not a timeout for the entire emanom process, but a timeout for individual ObsID-subexposure processes.

Returns
Information required by the SAS decorator that will run commands. Top level keys of any dictionaries are internal DAXA mission names, next level keys are ObsIDs. The return is a tuple containing a) a dictionary of bash commands, b) a dictionary of final output paths to check, c) a dictionary of extra info (in this case obs and analysis dates), d) a generation message for the progress bar, e) the number of cores allowed, and f) whether the progress bar should be hidden or not.

Return type
Tuple[dict, dict, dict, str, int, bool, Quantity]

daxa.process.xmm.check.**parse_emanom_out**(*log_file_path*, *acceptable_states=('G', 'I', 'U')*)

A simple function to parse the output file created by the emanom SAS task, which seeks to identify which MOS CCDs are in 'anomalous states'. The most relevant pieces of information contained in the output file are the CCD IDs, and the state code; the 'state' values have the following meanings:

Status- G is good at all energies

- I is intermediate for E<1 keV
- B is bad for E<1 keV
- O is off, chip not in use
- U is undetermined (low band counts <= 0)

Users can specify which states they would like to keep using the 'acceptable_states' parameter.

Parameters

- **log_file_path** (*str*) – The path to the output file created by emanom.
- **acceptable_states** (*List[str]/str*) – The CCD states which should be accepted. If a CCD is accepted then its ID will be returned by this function.

Returns
A list of CCD IDs which are in 'acceptable' states.

Return type
List[int]

process.xmm.clean

daxa.process.xmm.clean.**espfilt**(*obs_archive*, *method='histogram'*, *with_smoothing=True*, *with_binning=True*, *ratio=1.2*, *filter_lo_en=<Quantity 2500. eV>*, *filter_hi_en=<Quantity 8500. eV>*, *range_scale=None*, *allowed_sigma=3.0*, *gauss_fit_lims=(0.1, 6.5)*, *num_cores=1*, *disable_progress=False*, *timeout=None*)

The DAXA wrapper for the XMM SAS task espfilt, which attempts to identify good time intervals with minimal soft-proton flaring for individual sub-exposures (if multiple have been taken) of XMM ObsID-Instrument combinations. Both EPIC-PN and EPIC-MOS observations will be processed by this function.

This function does not generate final event lists, but instead is used to create good-time-interval files which are then applied to the creation of final event lists, along with other user-specified filters, in the 'cleaned_evt_lists' function.

Parameters

- **obs_archive** (Archive) – An Archive instance containing XMM mission instances with PN/MOS observations for which espfilt should be run. This function will fail if no XMM missions are present in the archive.
- **method** (*str*) – The method that espfilt should use to find soft proton flaring. Either 'ratio' or 'histogram' can be selected. The default is 'histogram'.
- **with_smoothing** (*bool/Quantity*) – Should smoothing be applied to the light curve data. If set to True (the default) a smoothing factor of 51 seconds is used, if set to False smoothing will be turned off, if an astropy Quantity is passed (with units convertible to seconds) then that value will be used for the smoothing factor.
- **with_binning** (*bool/Quantity*) – Should binning be applied to the light curve data. If set to True (the default) a bin size of 60 seconds is used, if set to False binning will be turned off, if an astropy Quantity is passed (with units convertible to seconds) then that value will be used for the bin size.
- **ratio** (*float*) – Flaring ratio of annulus counts.
- **filter_lo_en** (*Quantity*) – The lower energy bound for the event lists used for soft proton flaring identification.
- **filter_hi_en** (*Quantity*) – The upper energy bound for the event lists used for soft proton flaring identification.
- **range_scale** (*dict*) – Histogram fit range scale factor. The default is a dictionary with an entry for 'pn' (15.0) and an entry for 'mos' (6.0).
- **allowed_sigma** (*float*) – Limit in sigma for unflared rates.
- **gauss_fit_lims** (*Tuple[float, float]*) – The parameter limits for gaussian fits.
- **num_cores** (*int*) – The number of cores to use, default is set to 90% of available.
- **disable_progress** (*bool*) – Setting this to true will turn off the SAS generation progress bar.
- **timeout** (*Quantity*) – The amount of time each individual process is allowed to run for, the default is None. Please note that this is not a timeout for the entire espfilt process, but a timeout for individual ObsID-Inst-subexposure processes.

Returns

Information required by the SAS decorator that will run commands. Top level keys of any dictionaries are internal DAXA mission names, next level keys are ObsIDs. The return is a tuple containing a) a dictionary of bash commands, b) a dictionary of final output paths to check, c) a dictionary of extra info (in this case obs and analysis dates), d) a generation message for the progress bar, e) the number of cores allowed, and f) whether the progress bar should be hidden or not.

Return type

Tuple[dict, dict, dict, str, int, bool, Quantity]

process.xmm.generate

daxa.process.xmm.generate.**generate_images_expmaps**(*obs_archive*, *lo_en=<Quantity [0.5, 2.] keV>*, *hi_en=<Quantity [2., 10.] keV>*, *num_cores=1*)

A function to generate images and exposure maps for a processed XMM mission dataset contained within an archive. Users can select the energy band(s) that they wish to generate images and exposure maps within.

Parameters

- **obs_archive** (Archive) –
- **lo_en** – The lower energy bound(s) for the product being generated. This can either be passed as a scalar Astropy Quantity or, if sets of the same product in different energy bands are to be generated, as a non-scalar Astropy Quantity. If multiple lower bounds are passed, they must each have an entry in the hi_en argument. The default is 'Quantity([0.5, 2.0], 'keV')', which will generate two sets of products, one with lower bound 0.5 keV, the other with lower bound 2 keV.
- **hi_en** – The upper energy bound(s) for the product being generated. This can either be passed as a scalar Astropy Quantity or, if sets of the same product in different energy bands are to be generated, as a non-scalar Astropy Quantity. If multiple upper bounds are passed, they must each have an entry in the lo_en argument. The default is 'Quantity([2.0, 10.0], 'keV')', which will generate two sets of products, one with upper bound 2.0 keV, the other with upper bound 10 keV.
- **num_cores** (*int*) – The number of cores to use, default is set to 90% of available.

process.xmm.setup

daxa.process.xmm.setup.**cif_build**(*obs_archive*, *num_cores=1*, *disable_progress=False*, *analysis_date='now'*, *timeout=None*)

A DAXA Python interface for the SAS cifbuild command, used to generate calibration files for XMM observations prior to processing. The observation date is supplied by the XMM mission instance(s), and is the date when the observation was started (as acquired from the XSA).

Parameters

- **obs_archive** (Archive) – An Archive instance containing XMM mission instances for which observation calibration files should be generated. This function will fail if no XMM missions are present in the archive.
- **num_cores** (*int*) – The number of cores to use, default is set to 90% of available.
- **disable_progress** (*bool*) – Setting this to true will turn off the SAS generation progress bar.
- **analysis_date** (*str/datetime*) – The analysis date for which to generate calibration file. The default is 'now', but this parameter can be used to create calibration files as they would have been on a past date.
- **timeout** (*Quantity*) – The amount of time each individual process is allowed to run for, the default is None. Please note that this is not a timeout for the entire cif_build process, but a timeout for individual ObsID processes.

Returns

Information required by the SAS decorator that will run commands. Top level keys of any dictionaries are internal DAXA mission names, next level keys are ObsIDs. The return is a tuple containing a) a dictionary of bash commands, b) a dictionary of final output paths to check, c)

a dictionary of extra info (in this case obs and analysis dates), d) a generation message for the progress bar, e) the number of cores allowed, and f) whether the progress bar should be hidden or not.

Return type
Tuple[dict, dict, dict, str, int, bool, Quantity]

daxa.process.xmm.setup.**odf_ingest**(*obs_archive*, *num_cores=1*, *disable_progress=False*, *timeout=None*)

This function runs the SAS odfingest task, which creates a summary of the raw data available in the ODF directory, and is used by many SAS processing tasks.

Parameters

- **obs_archive** (Archive) – An Archive instance containing XMM mission instances for which observation summary files should be generated. This function will fail if no XMM missions are present in the archive.
- **num_cores** (*int*) – The number of cores to use, default is set to 90% of available.
- **disable_progress** (*bool*) – Setting this to true will turn off the SAS generation progress bar.
- **timeout** (*Quantity*) – The amount of time each individual process is allowed to run for, the default is None. Please note that this is not a timeout for the entire odf_ingest process, but a timeout for individual ObsID processes.

Returns
Information required by the SAS decorator that will run commands. Top level keys of any dictionaries are internal DAXA mission names, next level keys are ObsIDs. The return is a tuple containing a) a dictionary of bash commands, b) a dictionary of final output paths to check, c) a dictionary of extra info (in this case obs and analysis dates), d) a generation message for the progress bar, e) the number of cores allowed, and f) whether the progress bar should be hidden or not.

Return type
Tuple[dict, dict, dict, str, int, bool, Quantity]

daxa.process.xmm.setup.**parse_odf_sum**(*sum_path*, *obs_id=None*)

A function that takes a path to an XMM SAS summary file generated by the odf ingest command. The file will be filtered and parsed so that data relevant to DAXA processing valid scientific observations can be extracted. This includes things like whether a particular instrument was active, the number of sub-exposures, whether those sub-exposures were in a science mode that produces data useful for the study of astrophysical objects (i.e. not in a calibration or diagnosis mode). Data relevant to SAS procedures that calibrate and construct exposure lists is not included in the output of this function.

Parameters

- **sum_path** (*str*) – The path to the odf_ingest-generated summary file that is to be parsed into a dictionary of relevant information.
- **obs_id** (*str*) – Optionally, the observation ID that goes with this summary file can be passed, purely to make a possible error message more useful.

Returns
Multi-level dictionary of information, with top-level keys being instrument names. Next level contains information on whether the instrument was active and the number of exposures. This level has an 'exposures' key which is for a dictionary where the keys are all the exp_ids available for this instrument. Those keys are for dictionaries of exposure specific information, including mode, whether the exposure was scheduled, and the modes of the individual CCDs.

Return type
dict

process.erosita

process.erosita.assemble

daxa.process.erosita.assemble.**cleaned_evt_lists**(*obs_archive*, *lo_en=<Quantity 0.2 keV>*, *hi_en=<Quantity 10. keV>*, *flag=3221225472*, *flag_invert=True*, *pattern=15*, *num_cores=1*, *disable_progress=False*, *timeout=None*)

The function wraps the eROSITA eSASS task evtool, which is used for selecting events. This has been tested up to evtool v2.10.1

This function is used to apply the soft-proton filtering (along with any other filtering you may desire, including the setting of energy limits) to eROSITA event lists, resulting in the creation of sets of cleaned event lists which are ready to be analysed.

Parameters

- **obs_archive** (Archive) – An Archive instance containing eROSITA mission instances with observations for which cleaned event lists should be created. This function will fail if no eROSITA missions are present in the archive.
- **lo_en** (*Quantity*) – The lower bound of an energy filter to be applied to the cleaned, filtered, event lists. If 'lo_en' is set to an Astropy Quantity, then 'hi_en' must be as well. Default is 0.2 keV, which is the minimum allowed by the eROSITA toolset. Passing None will result in the default value being used.
- **hi_en** (*Quantity*) – The upper bound of an energy filter to be applied to the cleaned, filtered, event lists. If 'hi_en' is set to an Astropy Quantity, then 'lo_en' must be as well. Default is 10 keV, which is the maximum allowed by the eROSITA toolset. Passing None will result in the default value being used.
- **flag** (*int*) – FLAG parameter to select events based on owner, information, rejection, quality, and corrupted data. The eROSITA website contains the full description of event flags in section 1.1.2 of the following link: https://erosita.mpe.mpg.de/edr/DataAnalysis/prod_descript/EventFiles_edr.html. The default parameter will select all events flagged as either singly corrupt or as part of a corrupt frame.
- **flag_invert** (*bool*) – If set to True, this function will discard all events selected by the flag parameter. This is the default behaviour.
- **pattern** (*int*) – Selects events of a certain pattern chosen by the integer key. The default of 15 selects all four of the recognized legal patterns.
- **num_cores** (*int*) – The number of cores to use, default is set to 90% of available.
- **disable_progress** (*bool*) – Setting this to true will turn off the eSASS generation progress bar.
- **timeout** (*Quantity*) – The amount of time each individual process is allowed to run for, the default is None. Please note that this is not a timeout for the entire cleaned_evt_lists process, but a timeout for individual ObsID-Inst-subexposure processes.

process.erosita.clean

daxa.process.erosita.clean.**flaregti**(*obs_archive*, *pimin=<Quantity 200. eV>*, *pimax=<Quantity 10000. eV>*, *mask_pimin=<Quantity 200. eV>*, *mask_pimax=<Quantity 10000. eV>*, *binsize=1200*, *detml=10*, *timebin=<Quantity 20. s>*, *source_size=<Quantity 25. arcsec>*, *source_like=10*, *threshold=<Quantity -1. ct / (deg2 s)>*, *max_threshold=<Quantity -1. ct / (deg2 s)>*, *mask_iter=3*, *num_cores=1*, *disable_progress=False*, *timeout=None*)

The DAXA wrapper for the eROSITA eSASS task flaregti, which attempts to identify good time intervals with minimal flaring. This has been tested up to flaregti v1.20.

This function does not generate final event lists, but instead is used to create good-time-interval files which are then applied to the creation of final event lists, along with other user-specified filters, in the 'cleaned_evt_lists' function.

Parameters

- **obs_archive** (*Archive*) – An Archive instance containing eROSITA mission instances with observations for which flaregti should be run. This function will fail if no eROSITA missions are present in the archive.
- **pimin** (*float*) – Lower PI bound of energy range for lightcurve creation.
- **pimax** (*float*) – Upper PI bound of energy range for lightcurve creation.
- **mask_pimin** (*float*) – Lower PI bound of energy range for finding sources to mask.
- **mask_pimax** (*float*) – Upper PI bound of energy range for finding sources to mask.
- **binsize** (*int*) – Bin size of mask image (unit: sky pixels).
- **detml** (*int*) – Likelihood threshold for mask creation.
- **timebin** (*int*) – Bin size for lightcurve (unit: seconds).
- **source_size** (*int*) – Diameter of source extraction area for dynamic threshold calculation (unit: arcsec); this is the most important parameter if optimizing for extended sources.
- **source_like** (*int*) – Source likelihood for automatic threshold calculation.
- **threshold** (*float*) – Flare threshold; dynamic if negative (unit: counts/deg^2/sec).
- **max_threshold** (*float*) – Maximum threshold rate, if positive (unit: counts/deg^2/sec), if set this forces the threshold to be this rate or less.
- **mask_iter** (*int*) – Number of repetitions of source masking and GTI creation.
- **num_cores** (*int*) – The number of cores to use, default is set to 90% of available.
- **disable_progress** (*bool*) – Setting this to true will turn off the SAS generation progress bar.
- **timeout** (*Quantity*) – The amount of time each individual process is allowed to run for, the default is None. Please note that this is not a timeout for the entire flaregti process, but a timeout for individual ObsID-Inst-subexposure processes.

Returns

Information required by the eSASS decorator that will run commands. Top level keys of any dictionaries are internal DAXA mission names, next level keys are ObsIDs. The return is a tuple containing a) a dictionary of bash commands, b) a dictionary of final output paths to check, c) a dictionary of extra info (in this case obs and analysis dates), d) a generation message for the

progress bar, e) the number of cores allowed, and f) whether the progress bar should be hidden or not.

Return type
Tuple[dict, dict, dict, str, int, bool, Quantity]

process.erosita.setup

process.general

process.general.preprocessed

daxa.process.general.preprocessed.**preprocessed_in_archive**(*arch*, *missions=None*)

This function acts on an archive's missions which were created with pre-processed data (with things like pre-generated event lists, images, and exposure maps downloaded when the archive was set up). It will take the existing products and re-organise/re-name them into DAXA's processed archive structure, with the DAXA file naming scheme.

Parameters

- **arch** (Archive) – A DAXA archive that contains at least one mission with pre-processed data.
- **missions** (*List[BaseMission]*) – Optionally, a list of mission names that are to have their preprocessed data reorganised into the DAXA archive. Default is None, in which case all 'pre-processed' missions will be acted upon.

CHAPTER

SIX

PLANNED DAXA FEATURES

- **NuSTAR Data** - Data can be acquired and downloaded, but not processed yet.
- **Chandra Data** - Data can be acquired and downloaded, but not processed yet. Data can also be downloaded in a form that will work with the standard Chandra reprocessing scripts.
- **ROSAT Data** - Data can be acquired and downloaded, but not processed yet. Archived data products (images/exposure maps) can also be downloaded.
- **Swift Data** - Data can be acquired and downloaded, but not processed yet. Archived data products (images/exposure maps) can also be downloaded.
- **Suzaku Data** - Data can be acquired and downloaded, but not processed yet. Archived data products (images/exposure maps) can also be downloaded.
- **ASCA Data** - Data can be acquired and downloaded, but not processed yet. Archived data products (images/exposure maps) can also be downloaded.
- **INTEGRAL Data** - Data can be acquired and downloaded, but not processed yet. The downloaded data may not be in a structure compatible with OSA (work in progress).
- **Data Versioning** - A key feature of DAXA will be the ability to version archives, so that any updates to calibration or re-processing will be reflected in a version number and changelog.

CHAPTER

SEVEN

GETTING HELP AND SUPPORT

The best way to get help is to contact the development team, we would be happy to help with any questions you may have.

If you encounter a bug, or would like to make a feature request, please use the GitHub issues page, it really helps to keep track of everything. If you wish to contribute to DAXA, and opening an issue isn't sufficient to communicate your idea properly then please contact the development team to arrange a virtual meeting.

7.1 Contact

David Turner	turne540@msu.edu
Jessica Pilling	jp735@sussex.ac.uk

PYTHON MODULE INDEX

d

INDEX

D

E

F

G

I

L

M

N

O

P

R

S

T

X