ESOAsg Documentation

Release 0.1.1dev

Ema and Associates

May 11, 2021

CONTENTS

ESOAsg is a (work in progress) package created to help users to make data ESO phase3 compliant and to provide an easy access to the ESO archive. This code is free software and distributed in the hope that will be useful, but **without any warranty**. Data providers and data users are always the final resposible for validation and collection of the data.

CHAPTER

ONE

GET STARTED

1.1 Installing ESOAsg

This section describes how to install ESOAsg.

1.1.1 Installing Dependencies

There are a few packages that need to be installed before running the package.

We highly recommend that you use Anaconda for the majority of these installations.

Detailed installation instructions are presented below:

python and dependencies

ESOAsg runs with python 3.7 and with the following dependencies:

- python – version 3.7 or later
- astropy – version 4.2 or later
- astroquery – version 0.4 or later
- IPython – version 7.12 or later
- matplotlib – version 3.3 or later
- numpy – version 1.19 or later
- packaging – version 20.4 or later
- pytest version 6.2 or later
- pyvo – version 1.1 or later
- requests – version 2.25 or later

The following packages are required to work on Ligo gravitational wave events, see for instance this notebook:

- healpy – version 1.14 or later
- ligo.skymap – version 0.5 or later

If you are using Anaconda, you can check the presence of these packages with:

```
conda list "^python$|astropy$|astroquery
↪$|IPython|matplotlib|numpy|packaging|pytest|pyvo|requests|healpy|ligo.skymap"
```

If the packages have been installed, this command should print out all the packages and their version numbers.

If any of the packages are out of date, they can be updated with a command like:

```
conda update astropy
```

You might have troubles in installing pyvo. In case, you can get it from GitHub running in the directory where you want to install it:

```
git clone https://github.com/astropy/pyvo.git
cd pyvo
python setup.py install
```

git clone

To install the package via GitHub run:

```
git clone https://github.com/EmAstro/ESOAsg.git
```

And, given that the packages is still work in progress and you may want to updated on-the-fly, we then recommend to install it with the *develop* option:

```
cd ESOAsg
python setup.py develop
```

1.1.2 Testing the Installation

In order to assess whether ESOAsg has been properly installed, we suggest you run the following tests:

1. Run the default tests

In the directory where ESOAsg is installed run:

```
pytest
```

You should see that all the current test passed.

2. Ensure that the scripts work

Go to a directory outside of the ESOAsg directory, then type mod_header:

```
cd
mod_header -h
```

1.2 Versions

1.2.1 Developments in progress

Current developments are:

- introducing data containers
- including new scripts
- fixing header manipulation tools
- improving documentation

1.2.2 ESOAsg V0.1.1dev

- New HowTo on obtaining data from catalogues
- Improved tap query to obtain ESO catalogues

1.2.3 ESOAsg V0.1

This release deals with some major improvement on how the query to the archive are performed. In particular:

- *Archive Catalogues*
- *Archive Observations*
- *Archive Science Portal*

Some effort to improve documentation has been made

1.2.4 ESOAsg V0.0

This is an early version of the *ESOAsg* package. It provides some general scripts to deal with fits header modification and data access.

CHAPTER

TWO

ARCHIVE ACCESS

2.1 ESO Data Access Policy

The downloaded data are subject to the ESO Data Access Policy.

In particular, you are requested to acknowledge the usage of the ESO archive and of the ESO data; please refer to the *Acknowledgement policies* section.

If you plan to redistribute the downloaded data, please refer to the *Requirements for third parties distributing ESO data* section.

2.2 Archive Catalogues

The ESO Catalogue Facility provides access to the collection of catalogues and data that were produced by PIs of ESO programmes and then integrated into the ESO science archive through the Phase 3 process. The full list of available catalogues could be found here.

To access the data you can use the programmatic access via the *tap_cat* TAP Service. The module *archive_catalogues* provides some simple *python* wrapper around this.

More examples on how to use the TAP services for the ESO archive are provided in these notebooks.

2.2.1 Overview

The module *archive_catalogues* is based on the *query_catalogues.ESOCatalogues* class (a child of the *query.Query* class) that has the following attributes:

- *query* – A string that contains the query to be perfomed via the TAP Service *tap_cat*
- *result_from_query* – A table containing the result of the query
- *maxrec* – An integer that the define the maximum number of records that will be returned for a query
- *type_of_query* – A string that defines if the query will be run *synchronously* or *asynchronously*

After defining a *query* the *result_from_query* attribute is automatically filled by the method *run_query()*, for instance:

```
from ESOAsg.queries import query_catalogues
# define a query to obtain all `table_name` in the ESO Archive
query = 'SELECT schema_name, table_name from TAP_SCHEMA.tables'
# instantiate the class
all_catalogue_query = query_catalogues.ESOCatalogues(query=query)
```

(continues on next page)

(continued from previous page)

```
# run the query
all_catalogue_query.run_query()
# print the result on terminal
all_catalogue_query.result_from_query.pprint()
```

The module *archive_catalogue* provides a set of additional quality-of-life functions to circumvent the actual creation of the queries. These allow one to directly obtain the information needed in a *table* format.

2.2.2 Some examples

Note: The way the queries are created allows one to set as input either *collections* or *tables*. However, we strongly discurage to use both at the same times. Given that the connector between the two conditions is an *AND* this may give rise to an un-expected behaviour

Which catalogues are available

In general, it is possible to explore which catalogue are available in the ESO archive either with the query interface webpage, or with the archive science portal, or by running the TAP query to obtain all versions of all catalogues:

```
SELECT
    cat_id, collection, table_name, title, number_rows, number_columns,
    version, acknowledgment
FROM
    TAP_SCHEMA.tables
WHERE
    schema_name = 'safcat'
```

or this one to obtain only the latest version of the catalogues:

```
SELECT
    t1.cat_id, t1.collection, t1.table_name, t1.title,
    t1.number_rows, t1.number_columns,
    t1.version, t1.acknowledgment
FROM
    tables t1
LEFT OUTER JOIN
    tables t2 ON (t1.title = t2.title AND t1.version < t2.version)
WHERE
   t2.title IS null AND t1.cat_id IS NOT null AND t1.schema_name = 'safcat'
```

Alternatively, it is possible to obtain an *astropy.table* containing information on all catalogues and all their versions using (note that the query is more complicated of the ones above because more information are collected):

```
from ESOAsg import archive_catalogues
archive_catalogues.all_catalogues_info(all_versions=True)
```

This returns an *astropy.table* containing:

Column name	Description
collection	Name of the Phase 3 collection the catalogue belongs to
title	Title of the catalogue
version	Version of the catalogue
table_name	The fully qualified table name
filter	Name(s) of the filter bandpasses the original data were aquired with
instrument	Name(s) of the instrument(s) the original data were acquired with
telescope	Name(s) of the telescope(s) the original data were acquired with
publication_date	The data the catalogue was published
description	Describes tables in the tableset
number_rows	Number of rows present in this version of the catalogue
number_columns	Number of columns present in this version of the catalogue
rel_descr_url	Location of the release description document (typically a pdf)
acknowledgment	It provides the sentence to be used in your publication when making use of this catalogue
cat_id	Internal catalogue identifier
mjd_obs	The observational data this catalogue is based were taken between mjd_obs and mjd_end
mjd_end	The observational data this catalogue is based were taken between mjd_obs and mjd_end
skysqdeg	Area of the sky (in square degrees) covered by this catalogue
bibliography	Bibliographic reference in the form of either a BIBCODE or a DOI
document_id	Internal identifier of the release description document[1]
from_column	Column in the from_table
target_table	The table with the primary key
target_column	Column in the target_table
last_version	True if this is the latest version of the catalogue
RA_id	Identifier for RA in the catalogue
Dec_id	Identifier for Dec in the catalogue

Note: At first sight it may seem that not all cataloguess have the *RA_id* and *Dec_id*. This is because the catalogue is spread into more than one table. To identify the same source among the different tables of a catalogue the *target_table* and *target_column* should be used.

Which columns are in a catalogue

It is possible to get information on all columns present in a catalogue. For instance, the columns of the VIKING DR4 catalogue can be obtained running the following TAP query:

```
SELECT
    table_name, column_name, ucd, datatype, description, unit
FROM
    TAP_SCHEMA.columns
WHERE
    table_name = 'viking_er5_zyjj_1j_2hks_catMetaData_fits_V4'
```

A similar result can be obtained running:

```
archive_catalogues.columns_info(tables='viking_er5_zyjj_1j_2hks_catMetaData_fits_V4')
```

where the result is stored in an *astropy.table*.

[1] The web user interface for this catalogue is reachable via the URL computed appending the *cat_id* to the string: https://www.eso.org/qi/catalogQuery/index/

2.3 Archive Observations

The ESO archive currently (June 2020) contains more than 1.7 million spectra, more than 650,000 images, and more than 240,000 cubes.

There are three main ways to access the vaste amount of information present in the ESO archive:

- the Raw Data query form
- the Science Portal to browse and access the processed data (see also ref::*archive-science-portal*)
- the Programmatic and Tools access which permits direct database access to both raw and processed data, and to the ambient condition measurements

In addition, the *archive_observations* provides simple wrappers to efficiently embed the access to the ESO archive into *python* routines.

2.3.1 Overview

Similarly to the module *archive_catalogues* (see *Archive Catalogues*), the base for *archive_observations* is the *query_observations.ESOObservations* class (a child of the *query.Query* class) that has the following attributes:

- *query* – A string that contains the query to be perfomed via the TAP Service *tap_obs*
- *result_from_query* – A table containing the result of the query
- *maxrec* – An integer that the define the maximum number of records that will be returned for a query
- *type_of_query* – A string that defines if the query will be run *synchronously* or *asynchronously*

After defining a *query* you can fill the *result_from_query* attribute using the method *run_query()*, for instance:

```
from ESOAsg.queries import query_observations
# define a query to obtain all ALMA observations in the ESO Archive
query = "SELECT target_name, dp_id, s_ra, s_dec, instrument_name FROM ivoa.ObsCore WHERE␣
↪instrument_name = 'ALMA'"
# instantiate the class adn limit to the first 100 results
alma_query = query_observations.ESOObservations(query=query, maxrec=100)
# run the query
alma_query.run_query()
# print the result on terminal
alma_query.result_from_query.pprint()
```

The module *archive_observations* provides a set of functions to help a user to search and download for data without explicitly creating a TAP query.

2.3.2 Some examples

query to a specific position in the sky

The access to data in the ESO archive that intersects a cone with a certain radius (for instance around the star HD 057060: RA=07:18:40.38, Dec.=-24:33:31.3, radius=5arcsec) it is possible either with the archive science portal (see also *Archive Science Portal*), or by running the TAP query to ivoa.ObsCore:

```
SELECT
    target_name, dp_id, s_ra, s_dec, t_exptime, em_min, em_max,
    dataproduct_type, instrument_name, obstech, abmaglim,
    proposal_id, obs_collection
FROM
    ivoa.ObsCore
WHERE
    INTERSECTS(CIRCLE('ICRS',109.66824871,-24.55869773,0.0014), s_region) = 1
```

The same result, could be obtained using the function *query_from_radec()*:

```
from ESOAsg import archive_observations
from astropy.coordinates import SkyCoord
star_position = SkyCoord.from_name('HD 057060')
eso_data = archive_observations.query_from_radec(star_position, radius=5.)
```

eso_data is an *astropy.table* containing the information on the available data products. It is possible to download all this data feeding the *dp_id* column to the *download* function:

```
archive_observations.download(eso_data['dp_id']
```

A step-by-step explanation for the procedure to select and download data is provided in this jupyter notebook.

The *ESOAsg* package also provides the *get_data_from_radec* script to fully exploit this functionality (see *Archive Access Scripts* for futher details).

2.4 Archive Science Portal

The Archive Science Portal provides the primary entry point for interactive searching and browsing of the ESO Science Archive using an intuitive interactive user interface that supports iterative queries.

The archive contains both pipeline-processed data streams of many La Silla-Paranal instruments and a large collection of coherent data sets provided by users of ESO telescopes' time including the results of Public Surveys.

The module *archive_science_portal* provides some simple functions to create an *url* corresponding to your search.

Please, when you use data from the archive, follow the *ESO Data Access Policy*.

2.4.1 Overview

The module *archive_science_portal* uses *core.asp_queries* to create the *url* that links to your query at ASP. The basic idea is to start from the base url and keep adding conditions until your search criteria is satisfied. For instance, to search for all the *FORS2* and *XSHOOTER* spectra present in the archive:

```
from ESOAsg.core import asp_queries
# define the lists of instruments and data_types to be queried
instruments = ['FORS2', 'XSHOOOTER']
data_types = ['spectrum']
# create the url
url = '{0}{1}{2}'.format(asp_queries.base_url(),
                         asp_queries.condition_instruments(instruments),
                         asp_queries.condition_data_types(data_types, connector='&'))
```

(continues on next page)

(continued from previous page)

```
# run the query
asp_queries.run_query(url)
```

This will open this link on your browser.

The module *archive_science_portal* provides a set of functions to avoid the actual creation of the url by the user and to just obtain the link(s) to the page she/he is interested in.

2.4.2 Some examples

query to a specific position in the sky

In early 2019 the red supergiant star Betelgeuse began to dim (see, ESO science release #2003). To open a page containing all the reduced spectra and data cubes associated to the star, you can run:

```
from astropy.coordinates import SkyCoord
from ESOAsg import archive_science_portal
# resolve the star location from the name
star_position = SkyCoord.from_name('Betelgeuse')
# run the query with a search radius of 5 arcseconds anse selection spectrum and cube
# as data types
archive_science_portal.query_from_radec(star_position, radius=5., open_link=True,
                                        data_types=['spectrum', 'cube'])
```

This will open this page on your broswer. From there you can select the data to download (currently, Aug. 2020, 57 spectra from *HARPS*, *UVES*, and *XSHOOTER* and 10 datacubes from *ALMA* could be retrieved).

query to a polygon in the sky

The version 2.0 of the archive science portal offers the possibility of searching data withing irregular polygons. This is a relevant tool to detect electromagnetic counterpart of gravitational wave detections.

This possibility is provided by the *archive_science_portal.query_from_polygons* function. For instance, following the procedure described in this jupyter notebook, you can investigate which data are present within the 50% confidence contours of the location of the GW superevent s191205ah. The code will open the following 3 pages: Polygon-1, Polygon-2, and Polygon-3.

The *ESOAsg* package also provides the *get_data_from_gw_event* script to fully exploit this functionality (see *Archive Access Scripts* for futher details).

2.5 Archive Access Scripts

2.5.1 get_data_from_radec

Script to collect data from the ESO archive given RA and Dec

Inputs:

- **ra_degree** - RA of the target in degree [J2000]

- **dec_degree** - Dec of the target in degree [J2000]
- **radius** - Search cone radius in arcsec
- **instruments** - Limit the search to the selected ESO instruments
- **data_types** - Limit the search to the selected data type
- **maxrec** - Maximum number of files you would like to retrieve

2.5.2 get_data_from_gw_event

Script to collect data from the ESO archive given a *bayestar* file associated to a GW superevent

Inputs:

- **input_fits** - Input probability map out of the GW event pipeline
- **confidence_level** - Confidence level at which extract the contours
- **max_vertices** - Max number of vertices to be considered in the conversion from contours to polygons
- **show_sky** - Show the contours on a sky-map
- **asp_link** - Open ASP web-pages of the selected contours
- **download_data** - Download all data collected within the considered contours
- **instruments** - Limit the search to the selected ESO instruments
- **data_types** - Limit the search to the selected data type
- **maxrec** - Maximum number of files you would like to retrieve

CHAPTER

THREE

PHASE 3 VALIDATION

3.1 The ESO Phase 3 Process

The so-called Phase 3 is the process of preparation, validation and ingestion of science data product (*SDP*) for storage in the ESO science archive facility, and subsequent data pubblication to the science community. These *SDPs* are data products with instrument and atmospheric signatures removed, calibrated in physical units, and with noise properties quantified and documented.

To ensure the successful integration of *SDPs* into the archive, ESO supports the users in carrying out the Phase 3 process by defining ESO/SDP data standards, by devising procedures and providing the infrastructure for the delivery of the *SDPs*, and by supplying tools for the data preparation.

3.2 The ESOAsg API reference

3.2.1 ESOAsg

The archive_catalogues module

The archive_observations module

The archive_science_portal module

ESOAsg.archive_science_portal.**query_from_polygons**(*polygons=None*, *instruments=None*, *data_types=None*, *open_link=False*, *show_link=False*)

Query the ESO ASP service given a polygon

The *polygons* value (or list) needs to be given as a string defining the location in the sky of the polygon with RA, Dec, separated by commas and with the first RA, Dec pair that matches the last one (to close the polygon)

Parameters

- **polygons** (*list*) – list of *str* (or single *str*) containing the coordinates of the polygon in the sky you want to query
- **instruments** (*list*) – list of *str* (or single *str*) containing the instruments used to limit the search
- **data_types** (*list*) – list of *str* (or single *str*) containing the data types used to limit the search
- **open_link** (*bool*) – open a link to the ASP page

- **show_link** (*bool*) – show the link on the terminal

Returns None

ESOAsg.archive_science_portal.**query_from_radec**(*positions*, *radius=None*, *instruments=None*, *data_types=None*, *open_link=False*, *show_link=False*)

Query the ESO ASP service given a position

The *positions* value (or list) needs to be given as an *astropy.coordinates.SkyCoord* object. For further detail see here: astropy coordinates

Parameters

- **positions** (*astropy.coordinates.SkyCoord*) – coordinates (or list of coordinates) of the sky you want to query
- **radius** (*float*) – search radius in arcseconds
- **instruments** (*list*) – list of *str* (or single *str*) containing the instruments used to limit the search
- **data_types** (*list*) – list of *str* (or single *str*) containing the data types used to limit the search
- **open_link** (*bool*) – open a link to the ASP page
- **show_link** (*bool*) – show the link on the terminal

Returns None

3.2.2 ESOAsg.queries

The query.Query class

The query_catalogues.ESOCatalogues class

The query_observations.ESOObservations class

3.2.3 ESOAsg.core

The tap_queries module

The asp_queries module

Module to create and run ASP queries

The ESO Archive Science Portal (ASP) allows browsing and exploration of archive content using an intuitive interactive user interface that supports iterative queries

ESOAsg.core.asp_queries.**base_url**()

Return the url for ASP

Returns url

Return type str

ESOAsg.core.asp_queries.**condition_data_types**(*data_types*, *connector=None*)

Return condition for data types

Parameters

- **data_types** (*list*) – limit the search to the selected list of dataproduct types (e.g., *spectrum*)
- **connector** (*str, optional*) – connector to be put in front of the the condition (e.g., '&')

Returns string containing the *dp_type*= condition for ASP

Return type str

ESOAsg.core.asp_queries.**condition_instruments**(*instruments*, *connector=None*)
Return condition for instruments

Parameters

- **instruments** (*list*) – limit the search to the selected list of instruments (e.g., *XSHOOTER*)
- **connector** (*str, optional*) – connector to be put in front of the the condition (e.g., '&')

Returns string containing the *ins_id*= condition for ASP

Return type str

ESOAsg.core.asp_queries.**condition_polygon**(*polygon*, *connector=None*)
Return condition for polygon

Parameters

- **polygon** (*any*) – string containing the RA and Dec vertices (in degrees) of the polygon
- **connector** (*str, optional*) – connector to be put in front of the the condition (e.g., '&')

Returns string containing the *poly*= condition for ASP

Return type str

ESOAsg.core.asp_queries.**condition_position**(*ra*, *dec*, *radius=None*, *connector=None*)
Return condition for position

Parameters

- **ra** (*any*) – string containing the RA of the source in degree
- **dec** (*any*) – string containing the Dec of the source in degree
- **radius** (*any*) – string containing the search radius in arcseconds
- **connector** (*str, optional*) – connector to be put in front of the the condition (e.g., '&')

Returns string containing the *pos*= condition for ASP

Return type str

ESOAsg.core.asp_queries.**condition_radius**(*radius*, *connector=None*)
Return condition for radius

Parameters

- **radius** (*any*) – string containing the search radius in arcseconds
- **connector** (*str, optional*) – connector to be put in front of the the condition (e.g., '&')

Returns string containing the *r*= condition for ASP

Return type str

ESOAsg.core.asp_queries.**run_query**(*query_url*, *show_link=True*, *open_link=True*)
Run the ASP query

Parameters

- **query_url** (*str*) – url of the ASP query
- **open_link** (*bool*) – open a link to the ASP page
- **show_link** (*bool*) – show the link on the terminal

Returns None

ESOAsg.core.asp_queries.**sort_by**(*sort_type='-obs_date'*, *connector=None*)
Sort the ASP query for a specific keyword

Parameters

- **sort_type** (*str*) – value for which sort the ASP query
- **connector** (*str, optional*) – connector to be put in front of the the condition (e.g., '&')

Returns string that defines the sort type in a ASP query

Return type str

3.2.4 ESOAsg.datacontainers

eso_prodcatg

Class that defines the properties of an ESO PRODCATG

The current implementation follows the ESO Science Data Products Standard Version 6. More information in the Document ESO-044286.

class ESOAsg.datacontainers.eso_prodcatg.**ProdCatg**(*prodcatg_type=None*)
Class that defines the data product categories of the ESO Archive

Parameters **prodcatg_type** (*str*) – Name of the data product category

Raises

- **TypeError** – if *prodcatg* is not a *str*
- **ValueError** – if *prodcatg* is not a valid category accordingly to the ESO standard

data_format()
Returns a string describing the data format associated to the *prodcatg*.

Returns string containing the general description of the data format of the given *prodcatg*.

Return type str

property descriptor
Dictionary that dictates the property of a *prodcatg*

get_header_keyword_dictionary(*header_keyword*)

Parameters **header_keyword** –

Returns:

property header_keywords
Dictionary that dictates the property of the header keywords of a *prodcatg*

property prodcatg
Name of the *prodcatg*.

This set also the *descriptor* dictionary containing the relevant properties of the *prodcatg*.

show_header_keywords_info(*header_keyword*)
Given the name of a header keywords, the method prints its status for the assigned *prodcatg*

Parameters **header_keyword** (*list*) –

science_cube_ifs

Class to work on SCIENCE.CUBE.IFS

class ESOAsg.datacontainers.science_cube_ifs.**ScienceCubeIfs**(*primary_header=None*, *scidata_hdu=None*, *errdata_hdu=None*, *qualdata_hdu=None*, *others_hdu=None*)
A class used to define and make simple operations on data of type PRODCATG=`SCIENCE.CUBE.IFS`

This is a child of `ESOAsg.datacontainers.DataContainer`.

Please refer to the ESO Science Data Products Standard for a full description of the format.

Attributes:

Methods:

3.2.5 ESOASG.ancillary

checks

Module that performs some useful and basic checks and transformations

ESOAsg.ancillary.checks.**check_checksums**(*hdul*) → bool
Test if the *datasum* and *checksum* keywords in a *HDUList* are present and up-to-date

Parameters **hdul** (`astropy.io.fits.hdu.hdulist.HDUList`) – list of *astropy* HDUs to be checked

Returns *True* all the HDUs in the input HDUL have the correct *datasum* and *checksum*

Return type bool

ESOAsg.ancillary.checks.**check_disk_space**(*min_disk_space=6.0*) → bool
Check that there is enough space on the location where the code is running

Given a disk space limit in GB, the macro returns *True* if the disk where the code is running has more free GB than the given limit.

> **Warning:** The current implementation checks the disk where the code is running (i.e., from the directory: *./*). This may cause some troubles with shared disks.

Parameters **min_disk_space** (*float*) – Size of free space on disk required

Returns *True* if there is enough space on disk

Return type bool

ESOAsg.ancillary.checks.**connection_to_website**(*url*, *timeout=1.0*) → bool
Check there is an active connection to a website

Parameters

- **url** (*str*) – link to the website you want to check
- **timeout** (*float*) – timeout waiting for the website to respond

Returns *True* if there is an active connection, *False* and error raised if not.

Return type bool

ESOAsg.ancillary.checks.**fits_file_is_valid**(*fits_file*, *verify_fits=False*, *overwrite=False*) → bool

Check if a file exists and has a valid extension

The option *verify_fits* checks the header of the fits file using *astropy.io.fits.verify*

Parameters

- **fits_file** (*str*) – fits file you would like to check
- **verify_fits** (*bool*) – if set to *True*, it will verify that the fits file is complaint to the FITS standard.
- **overwrite** (*bool*) – if *True*, overwrite the input fits file with the header corrections from *verify_fits*

Returns *True* if exists *False* and warning raised if not.

Return type bool

ESOAsg.ancillary.checks.**header_is_valid**(*header*)

Check if an header is valid

ESOAsg.ancillary.checks.**image2d_is_valid**(*image2d*) → bool

Check if a 2D image is valid

Parameters **(obj** (*image2d*) – *numpy.ndarray*): image that you would like to check

Returns *True* if a valid 2D image *False* and error raised if not.

Return type bool

ESOAsg.ancillary.checks.**table_is_valid**(*table*)

Check if a table is valid

Parameters **table** (*fits.BinTableHDU* or *fits.TableHDU*) – table that you would like to check

Returns *True* if a valid table format *False* and error raised if not.

Return type is_table (*boolean*)

CHAPTER

FOUR

INDICES AND TABLES

- genindex
- modindex
- search

PYTHON MODULE INDEX

e

INDEX

Q

R

S

T