ESOAsg Documentation

Release 0.1.0

Ema and Associates

Jul 20, 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 document 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 4.0 or later
- astroquery – version 0.4 or later
- IPython – version 7.13 or later
- matplotlib – version 3.2 or later
- numpy – version 1.18.0 or later
- packaging – version 20.0 or later
- pytest version 5.0 or later
- pyvo – version 1.0 or later
- requests – version 2.23 or later

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

- healpy – version 1.13 or later
- ligo.skymap – version 0.3 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If any of the packages are out of date, they can be updated with a command like: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 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. 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 Develop

Current developments are:

- full refactoring of the *Archive Catalogues* module
- full refactoring of the *archive_observations* module
- introducing data containers
- fixing header manipulation tools
- improving documentation

1.2.2 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 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 catalogue 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.

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

2.2.1 Overview

The module *archive_catalogue* is based on the *query.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
```

(continues on next page)

(continued from previous page)

```
catalogue_list = query_catalogues.ESOCatalogues(query=query)
# run the query
catalogue_list.run_query()
# print the result on terminal
catalogue_list.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 availabe 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(all_versions=True)
```

This returns an *astropy.table* containing:

Column name	Description
collection	Name of the Phase 3 collection the catalog belongs to
title	Title of the catalog
version	Version of the catalog
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 catalog was published
description	Describes tables in the tableset
number_rows	Number of rows present in this version of the catalog
number_columns	Number of columns present in this version of the catalog
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 catalog
cat_id	Internal catalog identifier,
mjd_obs	The observational data this catalog is based were taken between mjd_obs and mjd_end
mjd_end	The observational data this catalog is based were taken between mjd_obs and mjd_end
skysqdeg	Area of the sky (in square degrees) covered by this catalog
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 catalog
RA_id	Identifier for RA in the catalog
Dec_id	Identifier for Dec in the catalog

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

Which columns are in a catalog

It is possible to get information on all columns present in a catalogue by running the following TAP query for the VIKING DR4 catalogue:

```
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 catalog is reachable via the URL computed appending the *cat_id* to the string: https://www.eso.org/qi/catalogQuery/index/

2.3 Archive Observations

2.4 Archive Science Portal

2.5 Scripts

2.5.1 get_data_from_ra_dec

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

2.6 The ESOAsg API reference

2.6.1 ESOAsg

The archive_catalogues module

ESOAsg.archive_catalogues.**all_catalogues_info**(*all_versions=False*, *verbose=False*)

Load an *astropy.table* with information on all catalogues present in the ESO archive

The output table will contain the following columns: *collection*, *title*, *version*, *table_name*, *filter*, *instrument*, *telescope*, *publication_date*, *description*, *number_rows*, *number_columns*, *rel_descr_url*, *acknowledgment*, *cat_id*, *mjd_obs*, *mjd_end*, *skysqdeg*, *bibliography*, *document_id*, *from_column*, *target_table*, *target_column*, *last_version*

For further information check the ESO catalogue facility

Note: This is analogue to run:

```
>>> catalogues_info(collections=None, tables=None)
```

with the difference that, given that no constraints are set, this returns the master table with all catalogues present in the ESO archive

---

Parameters

- **verbose** (*bool*) – if set to *True* additional info will be displayed
- **all_versions** (*bool*) – if set to *True* also obsolete versions of the catalogues are listed

Returns table containing the information on all catalogues present in the ESO archive

Return type astropy.table

ESOAsg.archive_catalogues.**catalogues_info**(*all_versions=False*, *collections=None*, *tables=None*, *verbose=False*)

Load an *astropy.table* with information on the selected catalogues

Specific catalogues can be selected by selecting a list of collections or a list of table_names. If both *collections* and *tables* are set to *None* information on all ESO catalogues will be returned. For further information check the ESO catalogue facility

The output table will contain the following columns: *collection*, *title*, *version*, *table_name*, *filter*, *instrument*, *telescope*, *publication_date*, *description*, *number_rows*, *number_columns*, *rel_descr_url*, *acknowledgment*, *cat_id*, *mjd_obs*, *mjd_end*, *skysqdeg*, *bibliography*, *document_id*, *from_column*, *target_table*, *target_column*, *last_version*

Note: The way the query is created is to set as input or *collections* or *tables*. Particular attention should be given if both *collections* and *tables* are not *None*. Given that the connector between the two conditions is an *AND* this may give rise to an un-expected behaviour

---

Parameters

- **all_versions** (*bool*) – if set to *True* also obsolete versions of the catalogues are listed

- **collections** (*any, optional*) – list of *str* containing the names of the collections (or single *str*) for which the query will be limited
- **tables** (*any, optional*) – list of *str* containing the table_name of the tables (or single *str*) for which the query will be limited
- **verbose** (*bool*) – if set to *True* additional info will be displayed

Returns table containing the information on the selected catalogues

Return type astropy.table

ESOAsg.archive_catalogues.**columns_info**(*collections=None*, *tables=None*, *verbose=False*)

Load a query that get names (and corresponding ucd) of the columns present in a collection

If *collections* and *tables* are *None* the query for the column of all collections and tables in the ESO archive is returned.

Note: The way the query is created is to set as input or *collections* or *tables*. Particular attention should be given if both *collections* and *tables* are not *None*. Given that the connector between the two conditions is an *AND* this may give rise to an un-expected behaviour

---

Parameters

- **collections** (*any, optional*) – list of *str* containing the names of the collections (or single *str*) from which the columns will be extracted
- **tables** (*any, optional*) – list of *str* containing the names of the tables (or single *str*) from which the columns will be extracted
- **verbose** (*bool*) – if set to *True* additional info will be displayed

Returns

table of all columns present in a table/collection. Information are stored in *table_name*, *column_name*, *ucd*, *datatype*, *description*, and *unit*

Return type astropy.table

ESOAsg.archive_catalogues.**get_catalogues**(*collections=None*, *tables=None*, *columns=None*, *type_of_query='sync'*, *all_versions=False*, *maxrec=None*, *verbose=False*)

Query the ESO tap_cat service for specific catalogues

There are two ways to select the catalogues you are interested in. Either you select directly the table_name (or the list of table_names) that you want to query, or you select a collection (or a list of collections). If you select this latter option, what happens in the background is that the code is going to search for the table(s) corresponding to the given collection and query them

If you are asking for more than one table, the result will be listed in a list of *astropy.tables* one per table

Parameters

- **collections** (*any*) – list of *str* containing the names (or a single *str*) of the collections for which the query will be limited
- **tables** (*any*) – list of *str* containing the table_name of the tables for which the query will be limited
- **columns** (*any*) – list of the *column_name* that you want to download. The full list of the columns in a table can be found by running *columns_info()*

- **all_versions** (*bool*) – if set to *True* also obsolete versions of the catalogues are searched in case *collections* is given
- **type_of_query** (*str*) – type of query to be run
- **maxrec** (*int, optional*) – define the maximum number of entries that a single query can return. If it is *None* the value is set by the limit of the service.
- **verbose** (*bool*) – if set to *True* additional info will be displayed

Returns *astropy.table* or *list* of *astropy.tables* containing the queried catalogues

Return type any

The archive_observations module

ESOAsg.archive_observations.**download**(*dp_ids*, *min_disk_space=6.0*)

Given a filename in the ADP format, the code download the file from the ESO archive

Parameters

- **dp_ids** (*any*) – list data product ID (or single product ID) to be downloaded
- **min_disk_space** (*float*) – the file will be downloaded only if there is this amount of space (in Gb) free on the disk

Returns None

ESOAsg.archive_observations.**query_from_polygons**(*polygons*, *instruments=None*, *data_types=None*, *verbose=False*, *maxrec=None*)

Query the ESO archive for data at a area in the sky defined by 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)

The output is in an (list of) *astropy.table* with columns defined in: *core.tap_queries.COLUMNS_FROM_OBSCORE*

Parameters

- **polygons** (*list*) – ist 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
- **verbose** (*bool*) – if set to *True* additional info will be displayed
- **maxrec** (*int, optional*) – define the maximum number of entries that a single query can return. If it is *None* the value is set by the limit of the service.

Returns results from the queries

Return type any

ESOAsg.archive_observations.**query_from_radec**(*positions=None*, *radius=None*, *instruments=None*, *data_types=None*, *verbose=False*, *maxrec=None*)

Query the ESO archive for data at a given position in RA and Dec

The *positions* value (or list) needs to be given as an astropy.coordinates.SkyCoord <https://docs.astropy.org/en/stable/coordinates/>`_ object.

The output is in an (list of) *astropy.table* with columns defined in: *core.tap_queries.COLUMNS_FROM_OBSCORE*

Note: In case you are querying radius=`None` is set, the query will performed with: *INTERSECT(POINT('',RA,Dec), s_region)* instead of: *INTERSECT(s_region,CIRCLE('',RA,Dec,radius/3600.))*. See here for further examples: tap obs examples

---

Parameters

- **positions** (*astropy.coordinates.SkyCoord*) – coordinates (or list of coordinates) of the sky you want to query
- **radius** (*float, optional*) – 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
- **verbose** (*bool*) – if set to *True* additional info will be displayed
- **maxrec** (*int, optional*) – define the maximum number of entries that a single query can return. If it is *None* the value is set by the limit of the service.

Returns results from the queries

Return type any

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

2.6.2 ESOAsg.queries

The query.Query class

class ESOAsg.queries.query.**Query**(*tap_service=None*, *query=None*, *type_of_query='sync'*, *result_from_query=None*, *maxrec=None*)

Base class that dictate the general behaviour of a query

tap_service
TAP service that will be used for the query

Type pyvo.dal.tap.TAPService

query
String containing the query

Type str

type_of_query
type of query to be run

Type str

maxrec
define the maximum number of entries that a single query can return

Type int, optional

result_from_query
result from the query to the TAP service

Type astropy.table.Table

clean_query()
Set the *query* attribute to None

clean_result_from_query()
Set the *result_from_query* attribute to None

get_result_from_query()
Returns a copy of *result_from_query*

print_query()
Print the query on the terminal

run_query(*to_string=True*)
Run the query and store results in the *result_from_query* attribute

Parameters **to_string** (*bool*, optional) – if set to True, if a column is in *bytes* format it transform it to *str*

which_columns()
Return a list with all the names of the columns in the attribute *result_from_query*

Returns List of all the names of the columns

Return type list_of_columns (*list*)

which_service()
Return the *tap_service* that is used together with a description of it

Returns None

The query_catalogues.ESOCatalogues class

Child class to run queries to the ESO catalogue

class ESOAsg.queries.query_catalogues.**ESOCatalogues**(*query=None*, *result_from_query=None*, *type_of_query='sync'*, *maxrec='20000'*)
This class is designed to query the scientific catalogues provided by the principal investigators of ESO observing programmes

This is a child of *ESOAsg.queries.query.Query* with the *tap_service* defined to be:

```
>>> tap_service=tap_queries.define_tap_service('eso_tap_cat')
```

Parameters

- **query** (*str*) – String containing the query
- **type_of_query** (*str*) – type of query to be run
- **maxrec** (*int, optional*) – define the maximum number of entries that a single query can return
- **result_from_query** (*astropy.table.Table*) – result from the query to the TAP service

set_last_version(*update=True*)
Set the *last_version* column to the *result_from_query* attribute

last_version is a column of *bool* where *False* means that there is a more update version of a catalogue

This works only if *result_from_query* contains the columns: *version* and *title*. In case the *last_version* column is already present, a warning is raised.

Parameters **update** (*bool*) – in case the *last_version* column is already present the code will update the value only if *update* is set to *True*

Returns None

The query_observations.ESOObservations class

class ESOAsg.queries.query_observations.**ESOObservations**(*query=None*, *result_from_query=None*, *type_of_query='sync'*, *maxrec='20000'*)

This class is designed to query the ESO archive for raw, reduced, and ambient data.

This is a child of *ESOAsg.queries.query.Query* with the *tap_service* defined to be:

```
>>> tap_service=tap_queries.define_tap_service('eso_tap_obs')
```

Parameters

- **query** (*str*) – String containing the query
- **type_of_query** (*str*) – type of query to be run
- **maxrec** (*int, optional*) – define the maximum number of entries that a single query can return
- **result_from_query** (*astropy.table.Table*) – result from the query to the TAP service

2.6.3 ESOAsg.core

The tap_queries module

Module to create and run TAP queries

The Table Access Protocol (TAP) is a web-service protocol that gives access to collections of tabular data referred to collectively as a *tableset*. TAP services accept queries posed against the *tableset* available via the service and return the query response as another table, in accord with the relational model. Queries to the ESO TAP services are submitted using the Astronomical Data Query Language ADQL [O08].

Currently, ESOAsg offers two TAP services:

- *eso_tap_obs*: to query both the database tables describing the observed raw and reduced data obtained at the La Silla Paranal Observatory, and the database table containing the ESO ambient conditions and meteorological measurements (seeing, isoplanatic angle, precipitable water, turbulence profiles, etc.)
- *eso_tap_cat*: to query the scientific catalogues provided by the principal investigators of ESO observing programmes

More information on the ESO TAP service are at this web-page and some examples are given in these notebooks

ESOAsg.core.tap_queries.**condition_collections_like**(*collections=None*)

Create a *LIKE - OR* condition over a list of collections

If *collections* is *None* the query the conditions will be substitute with '%' meaning that will have no effect

Parameters **collections** (*list, optional*) – list of *str* containing the names of the collections for which columns information will be returned

Returns set of conditions in the format: collection LIKE — OR

Return type str

ESOAsg.core.tap_queries.**condition_data_types_like**(*data_types=None*)

Create condition string to select only specific data product types in *ivoa.ObsCore*

Parameters **data_types** (*list*) – limit the search to the selected list of dataproduct types (e.g., *spectrum*)

Returns string containing the *dataproduct_type* condition for a query

Return type str

ESOAsg.core.tap_queries.**condition_instruments_like**(*instruments=None*)
Create condition string to select only specific instruments in *ivoa.ObsCore*

Parameters **instruments** (*list*) – limit the search to the selected list of instruments (e.g., *XSHOOTER*)

Returns string containing the *instrument_name* condition for a query

Return type str

ESOAsg.core.tap_queries.**condition_intersects_polygon**(*polygon*)
Create the WHERE INTERSECTS polygon condition string for a query

Parameters **polygon** (*str*) – coordinates of the verices of the polygon in the sky

Returns String containing the WHERE INTERSECT condition for a query

Return type str

ESOAsg.core.tap_queries.**condition_intersects_ra_dec**(*ra*, *dec*, *radius=None*)
Create the WHERE INTERSECTS condition string for a query

Parameters

- **ra** (*float*) – RA of the target in degrees and in the ICRS system
- **dec** (*float*) – Dec of the target in degrees and in the ICRS system
- **radius** (*float, optional*) – Search radius in arcsec. If set to *None* no radius will be considered in the INTERSECT condition

Returns String containing the WHERE INTERSECT condition for a query

Return type str

ESOAsg.core.tap_queries.**condition_tables_like**(*tables=None*)
Create a *LIKE - OR* condition over a list of table_names

If *tables* is *None* the query the conditions will be substitute with '%' meaning that will have no effect

Parameters **tables** (*list*) – list of *str* containing the table_name for which columns information will be returned

Returns set of conditions in the format: table_name LIKE — OR

Return type str

ESOAsg.core.tap_queries.**create_query_all_catalogues**(*all_versions=False*, *collections=None*, *tables=None*)
Create TAP query that returns info on all catalogues in the ESO archive

If *collections* or *tables* are not *None* only the query for the selected collections/tables is returned

Parameters

- **all_versions** (*bool*) – if set to *True* also obsolete versions of the catalogues are listed
- **collections** (*list, optional*) – list of *str* containing the names of the collections that are going to be selected
- **tables** (*list, optional*) – list of *str* containing the table_name of the tables that are going to be selected

Returns string containing the query to obtain all catalogues present in the ESO archive

Return type str

ESOAsg.core.tap_queries.**create_query_all_columns**(*collections=None*, *tables=None*)
Create a query that get names (and corresponding info) of the columns present in a collection (or in a table)

If *collections* and *tables* are *None* the query for the column of all collections in the ESO archive is returned.

Parameters

- **collections** (*list, optional*) – list of *str* containing the names of the collections for which columns information will be returned
- **tables** (*list, optional*) – list of *str* containing the table_name for which columns information will be returned

Returns string containing the query to obtain information on all columns present in the ESO archive

Return type str

ESOAsg.core.tap_queries.**create_query_obscore_base**()
Create the base string for a query to *ivoa.ObsCore*

Returns Base for the *ivoa.ObsCore* queries

Return type str

ESOAsg.core.tap_queries.**create_query_table**(*table_name*, *columns=None*)
Create a query to return selected columns from a table

Parameters

- **columns** (*list, optional*) – list of *str* containing the columns that will be queried
- **table_name** (*str*) – name of the table that will be queried

Returns query to obtain data from a table

Return type str

ESOAsg.core.tap_queries.**define_tap_service**(*which_tap_service*)
Load a Table Access Protocol (TAP) service from defaults

Currently the supported TAP services are: * *eso_tap_cat*: TAP service for scientific catalogues generated by ESO observing teams * *eso_tap_obs*: TAP service for raw, reduced, and ambient data

See pyvo docs for further details

Parameters **which_tap_service** (*str*) – Select the TAP services to be queried

Returns TAP service used for the queries

Return type pyvo.dal.tap.TAPService

ESOAsg.core.tap_queries.**print_query**(*query*)
Print the query on the terminal

In case the *query* is empty, a warning is raised

Parameters **query** (*str*) – String containing the query

Returns None

ESOAsg.core.tap_queries.**run_query**(*tap_service*, *query*, *type_of_query*, *maxrec='20000'*)
Run query to TAP service and return result as an *astropy.Table*

If the job requires to much time to run, the code will move to an asynchronous query.

Parameters

- **tap_service** (*pyvo.dal.tap.TAPService*) – TAP service that will be used for the query

- **query** (*str*) – query to be run
- **type_of_query** (*str*) – type of query to be run
- **maxrec** (*int*) – define the maximum number of entries that a single query can return. Default is set by default.get_value('maxrec')

Returns result from the query to the TAP service

Return type astropy.table

ESOAsg.core.tap_queries.**run_query_async**(*tap_service*, *query*, *maxrec='20000'*)
Run an asynchronous query to TAP service and return result as an *astropy.Table*

Parameters

- **tap_service** (*pyvo.dal.tap.TAPService*) – TAP service that will be used for the query
- **query** (*str*) – query to be run
- **maxrec** (*int*) – define the maximum number of entries that a single query can return. Default is set by default.get_value('maxrec')

Returns result from the query to the TAP service

Return type astropy.table

ESOAsg.core.tap_queries.**run_query_sync**(*tap_service*, *query*, *maxrec='20000'*)
Run a synchronous query to TAP service and return result as an *astropy.Table*

If the synchronous query fails, the code automatically tries to run the same query asynchronously

Parameters

- **tap_service** (*pyvo.dal.tap.TAPService*) – TAP service that will be used for the query
- **query** (*str*) – query to be run
- **maxrec** (*int*) – define the maximum number of entries that a single query can return. Default is set by default.get_value('maxrec')

Returns result from the query to the TAP service

Return type astropy.table

ESOAsg.core.tap_queries.**which_service**(*tap_service*)
Print a summary description of the TAP service used

Parameters **tap_service** (*pyvo.dal.tap.TAPService*) – TAP service used for the queries

Returns None

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*, *show_link=True*, *open_link=True*)
Run the ASP query

Parameters

- **query** (*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

2.6.4 ESOAsg.datacontainers

ligthcurves

Class to work on light curves

class ESOAsg.datacontainers.lightcurves.**LightCurves**(*primary_header=None*, *header=None*, *time=None*, *flux=None*, *error=None*, *time_bin=None*, *background=None*, *quality=None*, *others=None*)
A class used to define and make simple operations on time series

This allows to perform some basic tasks on a time series and to save it in a format that is ESO Phase3 compliant.

Attributes:

Methods:

check(*autocorrect=False*)
Checks that a LightCurves objects is in a format compatible with the ESO standard

load_from_table(*table*, *primary_header=None*, *copy_header=True*, *where_time='TIME'*, *where_time_bin='TIME_BIN'*, *where_flux='FLUX'*, *where_error='ERROR'*, *where_background='BACKGROUND'*, *where_quality='QUAL'*)
Given a table put it in a LightCurves object

Parameters

- **where_quality** –
- **where_background** –
- **where_error** –
- **where_time_bin** –
- **where_time** –
- **copy_header** –

- **primary_header** –
- **where_flux** –

ESOAsg.datacontainers.lightcurves.**save_into_fits**(*fits_file_name*, *primary_header*, *light_curve_headers*, *light_curve_names*, *light_curves*, *overwrite=True*)

Parameters

- **light_curves** –
- **light_curve_names** –
- **light_curve_headers** –
- **fits_file_name** –
- **primary_header** –
- **lightcurves** –
- **overwrite** (*bool*) –

Returns:

CHAPTER

THREE

INDICES AND TABLES

- genindex
- modindex
- search

BIBLIOGRAPHY

[O08] Oritz et al., (2008) IVOA Astronomical Data Query Language

PYTHON MODULE INDEX

e

INDEX

G

L

M

P

Q

R

S

T

W