Warning

The WDB (Web DataBase) API is being deprecated and replaced by TAP (Table Access Protocol), a standardized interface for querying astronomical datasets using ADQL (Astronomical Data Query Language). While the Python interface remains the same, the values accepted by the `columns` and `column_filters` parameters must reflect TAP field names and ADQL syntax. This means that, although the structure of your code won't need to change, **the values you pass to the arguments `columns` and `column_filters` must be revised** to comply with the new format.

In TAP, `column_filters` accepts ADQL expressions. For example:

```
column_filters = {
    'some_int_column': "< 5",
    'some_float_column_2': ">= 1.23",
    'some_char_column': "like '%John%'",
    'some_generic_column': "in ('mango', 'apple', 'kiwi')",
    'other_generic_column': "between '2024-01-01' and '2024-12-31'"
}
```

Please review your queries carefully and update them accordingly to ensure compatibility with the new astroquery versions. See section Using the correct column_filters

Getting started

This is a python interface for querying the ESO archive web service. For now, it supports the following:

- listing available instruments
- listing available surveys (phase 3)
- searching INSTRUMENT SPECIFIC raw data (table `ist.<instrument_name>`) via the ESO TAP service*
- searching data products (phase 3; table `ivoa.ObsCore`) via the ESO TAP service*
- searching raw data (table `dbo.raw`) via the ESO TAP service*
- downloading data by dataset identifiers: http://archive.eso.org/cms/eso-data/eso-data-direct-retrieval.html

* ESO TAP web interface: https://archive.eso.org/programmatic/#TAP

Requirements

The following packages are required for the use of this module:

- keyring
- lxml
- requests >= 2.4.0

Authentication with ESO User Portal

Most of the datasets in the ESO Science Archive are public and can be downloaded anonymously without authenticating with the ESO User Portal (https://www.eso.org/sso/login). Data with restricted access like datasets under proprietary period can be downloaded by authorised users (for example PIs of the corresponding observing programmes and their delegates) after authentication with the ESO User Portal. This authentication is performed directly with the provided **`login()`** command, as illustrated in the example below. This method uses your keyring to securely store the password in your operating system. As such you should have to enter your correct password only once, and later be able to use this package for automated interaction with the ESO archive.

```
>>> from astroquery.eso import Eso
>>> eso = Eso()
>>> # First example: TEST is not a valid username, it will fail
>>> eso.login(username="TEST")
WARNING: No password was found in the keychain for the provided username. [astroquery.qu
TEST, enter your password:

INFO: Authenticating TEST on https://www.eso.org/sso ... [astroquery.eso.core]
ERROR: Authentication failed! [astroquery.eso.core]
>>> # Second example: pretend ICONDOR is a valid username
>>> eso.login(username="ICONDOR", store_password=True)
WARNING: No password was found in the keychain for the provided username. [astroquery.qu
ICONDOR, enter your password:

INFO: Authenticating ICONDOR on https://www.eso.org/sso ... [astroquery.eso.core]
INFO: Authentication successful! [astroquery.eso.core]
>>> # After the first login, your password has been stored
>>> eso.login(username="ICONDOR")
INFO: Authenticating ICONDOR on https://www.eso.org/sso ... [astroquery.eso.core]
INFO: Authentication successful! [astroquery.eso.core]

>>> # Successful download of a public file (with or without login)
>>> eso.retrieve_data('AMBER.2006-03-14T07:40:19.830')
INFO: Downloading file 1/1 https://dataportal.eso.org/dataPortal/file/AMBER.2006-03-14T0
INFO: Successfully downloaded dataset AMBER.2006-03-14T07:40:19.830

>>> # Access denied to a restricted-access file (as anonymous user or as authenticated b
>>> eso.retrieve_data('ADP.2023-03-02T01:01:24.355')
INFO: Downloading file 1/1 https://dataportal.eso.org/dataPortal/file/ADP.2023-03-02T01:
ERROR: Access denied to https://dataportal.eso.org/dataPortal/file/ADP.2023-03-02T01:01:
```

Automatic password

As shown above, your password can be stored by the keyring module, if you pass the argument `store_password=True` to `Eso.login()`. For security reason, storing the password is turned off by default.

MAKE SURE YOU TRUST THE MACHINE WHERE YOU USE THIS FUNCTIONALITY!!!

NB: You can delete your password later with the command `keyring.delete_password('astroquery:www.eso.org', 'username')`.

Automatic login

You can further automate the authentication process by configuring a default username. The astroquery configuration file, which can be found following the procedure detailed in astropy.config, needs to be edited by adding `username = ICONDOR` in the `[eso]` section.

When configured, the username in the `login()` method call can be omitted as follows:

```
>>> from astroquery.eso import Eso
>>> eso = Eso()
>>> eso.login()
ICONDOR, enter your ESO password:
```

NB: If an automatic login is configured, other Eso methods can log you in automatically when needed.

Query the ESO archive for raw data

Identifying available instrument-specific queries

The direct retrieval of datasets is better explained with a running example, continuing from the authentication example above. The first thing to do is to identify the instrument to query. The list of available instrument-specific queries can be obtained with the **`list_instruments()`** method.

```
>>> from astroquery.eso import Eso
>>> eso = Eso()
>>> eso.list_instruments()
['alpaca', 'amber', 'apex', 'crires', 'efosc', 'eris', 'espresso', 'fiat',
 'fors1', 'fors2', 'giraffe', 'gravity', 'harps', 'hawki', 'isaac', 'kmos',
 'matisse', 'midi', 'muse', 'naco', 'nirps', 'omegacam', 'pionier', 'sinfoni',
 'sofi', 'sphere', 'uves', 'vimos', 'vircam', 'visir', 'wlgsu', 'xshooter']
```

In the example above, the instruments listed correspond to those retrieved by running the following query on the ESO **Programmatic Access** website (https://archive.eso.org/programmatic/#TAP):

```
select table_name from TAP_SCHEMA.tables where schema_name='ist' order by table_name
```

Inspecting available query options

Once an instrument is chosen, `midi` for example, the columns available for that instrument can be inspected by setting the `help=True` keyword of the **`query_instrument()`** method. The list of columns contains its datatype and unit. The xtype is to be more specific, as certain columns with datatype `char` actually define timestamps or regions in the sky.

```
>>> eso.query_instrument('midi', help=True)
INFO:
Columns present in the table ist.midi:
    column_name     datatype    xtype     unit
------------------- -------- ----------- ------
     access_estsize     long              kbyte
         access_url     char
       datalink_url     char
           date_obs     char
                dec   double                deg
      del_ft_sensor     char
      del_ft_status     char
            det_dit    float                  s
          det_ndit      int
      dimm_fwhm_avg    float             arcsec
      dimm_fwhm_rms    float             arcsec
             dp_cat     char
              dp_id     char
                ...      ...
       release_date     char   timestamp
           s_region     char adql:REGION
                ...      ...
          telescope     char
          tpl_expno      int
             tpl_id     char
           tpl_name     char
           tpl_nexp      int
```

```
          tpl_start     char
                utc    float                 s

Number of records present in the table ist.midi:
421764
 [astroquery.eso.core]
```

Note: for a deeper description of each column, the following query can be issued on the ESO **Programmatic Access** website (https://archive.eso.org/programmatic/#TAP):

```
select column_name, description from TAP_SCHEMA.columns where table_name = 'ist.midi'
```

Querying with constraints

It is now time to query the `midi` instrument for datasets. In the following example, observations of target `NGC 4151` between `2008-01-01` and `2009-05-12` are searched, and the query is configured to return two columns: the date of observation and the name of the object.

```
>>> table = eso.query_instrument(
...             'midi',
...             column_filters={
...                 'object':'NGC4151',
...                 'exp_start': "between '2008-01-01' and '2009-05-12'"
...             },
...             columns=['object', 'date_obs']
...         )
>>> table
<Table length=196>
object         date_obs
------- ----------------------
NGC4151 2008-04-22T02:07:50.154
NGC4151 2008-04-22T02:08:20.345
NGC4151 2008-04-22T02:09:47.846
NGC4151 2008-04-22T02:10:18.038
    ...                     ...
NGC4151 2009-05-11T01:39:09.750
NGC4151 2009-05-11T01:40:24.235
NGC4151 2009-05-11T01:41:38.742
NGC4151 2009-05-11T01:42:08.432
```

Querying all instruments

The ESO database can also be queried without a specific instrument in mind. This is what the method **`query_main()`** is for. The associated table on the ESO **Programmatic Access** website (https://archive.eso.org/programmatic/#TAP) is `dbo.raw`, and the simplest query would be: `select * from dbo.raw`. Except for the keyword specifying the instrument,the behaviour of **`query_main()`** is identical to **`query_instrument()`**.

ESO instruments without a specific query interface can be queried with **`query_main()`**, specifying the `instrument` constraint. This is the case of e.g. `harps`, `feros` or the all sky cameras APICAM and MASCOT. Here is an example to query all-sky images from APICAM with `luminance` filter.

```
>>> eso.maxrec = -1   # Return all results
                      # (i.e. do not truncate the query even if it is slow)
>>> table = eso.query_main(
...                     column_filters={
...                         'instrument': 'APICAM',
```

```
...                         'filter_path': 'LUMINANCE',
...                         'exp_start': "between '2019-04-26' and '2019-04-27'"
...                     }
...                 )
>>> print(len(table))
215
>>> print(table.columns)
<TableColumns names=('access_estsize','access_url','datalink_url','date_obs',
        'dec','dec_pnt','det_chip1id','det_chop_ncycles','det_dit','det_expid','det_ndit
        'dp_cat','dp_id','dp_tech','dp_type','ecl_lat','ecl_lon','exp_start','exposure',
        'filter_path','gal_lat','gal_lon','grat_path','gris_path','ins_mode','instrument
        'lambda_max','lambda_min','last_mod_date','mjd_obs','ob_id','ob_name','object',
        'obs_mode','origfile','period','pi_coi','prog_id','prog_title','prog_type','ra',
        'ra_pnt','release_date','s_region','slit_path','target','tel_airm_end',
        'tel_airm_start','tel_alt','tel_ambi_fwhm_end','tel_ambi_fwhm_start',
        'tel_ambi_pres_end','tel_ambi_pres_start','tel_ambi_rhum','tel_az','telescope',
        'tpl_expno','tpl_id','tpl_name','tpl_nexp','tpl_seqno','tpl_start')>
>>> table[["object", "ra", "dec", "date_obs", "prog_id"]].pprint(max_width=200)
 object      ra          dec              date_obs          prog_id
            deg          deg
------- ------------ ------------ ----------------------- ------------
ALL SKY 145.29212694 -24.53624194 2019-04-26T00:08:49.000 60.A-9008(A)
ALL SKY 145.92251305 -24.53560305 2019-04-26T00:11:20.000 60.A-9008(A)
ALL SKY    146.55707 -24.53497111 2019-04-26T00:13:52.000 60.A-9008(A)
ALL SKY    147.18745 -24.53435388 2019-04-26T00:16:23.000 60.A-9008(A)
ALL SKY 147.81365305 -24.53375305 2019-04-26T00:18:53.000 60.A-9008(A)
ALL SKY 148.56509194   -24.533045 2019-04-26T00:21:53.000 60.A-9008(A)
ALL SKY 149.19963805    -24.53246 2019-04-26T00:24:25.000 60.A-9008(A)
ALL SKY 149.83418111 -24.53188611 2019-04-26T00:26:57.000 60.A-9008(A)
ALL SKY 150.46037194 -24.53133111 2019-04-26T00:29:27.000 60.A-9008(A)
ALL SKY 151.08656111 -24.53078805 2019-04-26T00:31:57.000 60.A-9008(A)
ALL SKY 151.85050805    -24.53014 2019-04-26T00:35:00.000 60.A-9008(A)
ALL SKY    152.48504   -24.529615 2019-04-26T00:37:32.000 60.A-9008(A)
    ...          ...          ...                     ...          ...
ALL SKY 289.40910694 -24.66412305 2019-04-26T09:44:00.000 60.A-9008(A)
ALL SKY 290.04024305 -24.66522194 2019-04-26T09:46:31.000 60.A-9008(A)
ALL SKY 290.67974305    -24.66633 2019-04-26T09:49:04.000 60.A-9008(A)
ALL SKY    291.30671 -24.66741111 2019-04-26T09:51:34.000 60.A-9008(A)
ALL SKY 291.93786305 -24.66849388 2019-04-26T09:54:05.000 60.A-9008(A)
ALL SKY   139.655775   -24.542425 2019-04-26T23:42:23.000 60.A-9008(A)
ALL SKY   140.282015 -24.54169694 2019-04-26T23:44:53.000 60.A-9008(A)
ALL SKY 140.91242694 -24.54097305 2019-04-26T23:47:24.000 60.A-9008(A)
ALL SKY 141.54283388    -24.54026 2019-04-26T23:49:55.000 60.A-9008(A)
ALL SKY 142.16906388 -24.53956194 2019-04-26T23:52:25.000 60.A-9008(A)
ALL SKY    142.93306 -24.53872388 2019-04-26T23:55:28.000 60.A-9008(A)
ALL SKY 143.56345694 -24.53804388 2019-04-26T23:57:59.000 60.A-9008(A)
Length = 215 rows
```

Query the ESO archive for reduced data

In addition to raw data, ESO makes available processed data. In this section, we show how to obtain these processed survey data from the archive.

Identify available surveys

The list of available surveys can be obtained with **`astroquery.eso.EsoClass.list_surveys()`** as follows:

```
>>> surveys = eso.list_surveys()
```

Query a specific survey with constraints

Let's assume that we work with the `HARPS` survey, and that we are interested in target `HD203608`. The archive can be queried as follows:

```
>>> table = eso.query_surveys(surveys='HARPS', target_name="HD203608")
```

The returned table has a `dp_id` column, which can be used to retrieve the datasets with **`astroquery.eso.EsoClass.retrieve_data()`**: `eso.retrieve_data(table["dp_id"][0])`. More details about this method in the next section.

Obtaining extended information on data products

Only a small subset of the keywords present in the data products can be obtained with **`query_instrument()`** or **`query_main()`**. There is however a way to get the full primary header of the FITS data products, using **`get_headers()`**. This method is detailed in the example below.

```
>>> table = eso.query_instrument('midi',
...                      column_filters={
...                          'object': 'NGC4151',
...                          'date_obs': "<='2008-01-01'"
...                      },
...                      columns=['object', 'date_obs', 'dp_id'])
>>> table_headers = eso.get_headers(table["dp_id"])
>>> len(table_headers.columns)
336
>>> table_headers.pprint()
           DP.ID             SIMPLE BITPIX ...   HIERARCH ESO OCS EXPO7 FNAME2     HIERA
--------------------------- ------ ------ ... ---------------------------------- -------
MIDI.2007-02-07T07:01:51.000   True     16 ...
MIDI.2007-02-07T07:02:49.000   True     16 ...
MIDI.2007-02-07T07:03:30.695   True     16 ...
MIDI.2007-02-07T07:05:47.000   True     16 ...
MIDI.2007-02-07T07:06:28.695   True     16 ...
MIDI.2007-02-07T07:09:03.000   True     16 ...
MIDI.2007-02-07T07:09:44.695   True     16 ...
MIDI.2007-02-07T07:13:09.000   True     16 ...
MIDI.2007-02-07T07:13:50.695   True     16 ...
MIDI.2007-02-07T07:15:55.000   True     16 ...
MIDI.2007-02-07T07:16:36.694   True     16 ...
MIDI.2007-02-07T07:19:25.000   True     16 ...
MIDI.2007-02-07T07:20:06.695   True     16 ... MIDI.2007-02-07T07:20:06.695.fits
MIDI.2007-02-07T07:22:57.000   True     16 ... MIDI.2007-02-07T07:20:06.695.fits MIDI.20
MIDI.2007-02-07T07:23:38.695   True     16 ... MIDI.2007-02-07T07:20:06.695.fits MIDI.20
```

As shown above, for each data product ID (`DP.ID`), the full header (336 columns in our case) of the archive FITS file is collected. In the above table `table_headers`, there are as many rows as in the column `table['DP.ID']`.

Downloading datasets from the archive

Continuing from the query with constraints example, the first two datasets are selected, using their data product IDs `dp_id`, and retrieved from the ESO archive.

```
>>> data_files = eso.retrieve_data(table['dp_id'][:2])
INFO: Downloading datasets ... [astroquery.eso.core]
INFO: Downloading 2 files ... [astroquery.eso.core]
INFO: Downloading file 1/2 https://dataportal.eso.org/dataPortal/file/MIDI.2007-02-07T07
INFO: Successfully downloaded dataset MIDI.2007-02-07T07:01:51.000 to /Users/foobar/.ast
INFO: Downloading file 2/2 https://dataportal.eso.org/dataPortal/file/MIDI.2007-02-07T07
INFO: Successfully downloaded dataset MIDI.2007-02-07T07:02:49.000 to /Users/foobar/.ast
INFO: Uncompressing file /Users/foobar/.astropy/cache/astroquery/Eso/MIDI.2007-02-07T07:
INFO: Uncompressing file /Users/foobar/.astropy/cache/astroquery/Eso/MIDI.2007-02-07T07:
INFO: Done! [astroquery.eso.core]
>>> data_files
['/Users/foobar/.astropy/cache/astroquery/Eso/MIDI.2007-02-07T07:01:51.000.fits',
 '/Users/foobar/.astropy/cache/astroquery/Eso/MIDI.2007-02-07T07:02:49.000.fits']
```

The file names, returned in data_files, points to the decompressed datasets (without the .Z extension) that have been locally downloaded. They are ready to be used with **`fits`**.

The default location (in the astropy cache) of the decompressed datasets can be adjusted by providing a `destination` keyword in the call to **`retrieve_data()`**.

By default, if a requested dataset is already found, it is not downloaded again from the archive. To force the retrieval of data that are present in the destination directory, use `continuation=True` in the call to **`retrieve_data()`**.

Troubleshooting

Clearing the cache

If you are repeatedly getting failed queries, or bad/out-of-date results, try clearing your cache:

```
>>> from astroquery.eso import Eso
>>> Eso.clear_cache()
```

If this function is unavailable, upgrade your version of astroquery. The `clear_cache` function was introduced in version 0.4.7.dev8479.

Using the correct `column_filters`

Two concrete and relevant examples of fields present in WDB but not present in TAP/ADQL are `stime` and `etime`. The following snippet shows how to adapt the filters to the TAP / ADQL syntax:

```
# The following filters won't work:
column_filters = {
    'stime': '2024-01-01'
    'etime': '2024-12-31'
}

# Replace by:
column_filters = {
    'exp_start': "between '2024-01-01' and '2024-12-31'"
}

# --- #
```

```
# The following filters won't work:
column_filters = {
    'stime': '2024-01-01'
}

# Replace by:
column_filters = {
    'exp_start': "> '2024-01-01'"
}

# --- #

# The following filters won't work:
column_filters = {
    'etime': '2024-12-31'
}

# Replace by:
column_filters = {
    'exp_start': "< '2024-12-31'"
}
```

Reference/API

astroquery.eso Package

ESO service.

Classes

EsoClass()	User facing class to query the ESO archive
Conf()	Configuration parameters for **astroquery.eso**.