The Astronomical Journal, 157:98 (7pp), 2019 March https://doi.org/10.3847/1538-3881/aafc33

astroquery: An Astronomical Web-querying Package in Python

Adam Ginsburg[1], Brigitta M. Sipőcz[2,20], C. E. Brasseur[3], Philip S. Cowperthwaite[4], Matthew W. Craig[5], Christoph Deil[6], James Guillochon[4], Giannina Guzman[7,8], Simon Liedtke[21], Pey Lian Lim[3], Kelly E. Lockhart[4], Michael Mommert[9], Brett M. Morris[10], Henrik Norman[11,12], Madhura Parikh[22], Magnus V. Persson[13], Thomas P. Robitaille[14], Juan-Carlos Segovia[12], Leo P. Singer[15,16], Erik J. Tollerud[3], Miguel de Val-Borro[8,17], Ivan Valtchanov[18], and Julien Woillez[19]

The Astroquery collaboration, a subset of the astropy collaboration

[1] Jansky fellow of the National Radio Astronomy Observatory, 1003 Lopezville Road, Socorro, NM 87801 USA; aginsbur@nrao.edu, adam.g.ginsburg@gmail.com
[2] Institute of Astronomy, University of Cambridge, Madingley Road, Cambridge CB3 0HA, UK
[3] Space Telescope Science Institute, 3700 San Martin Drive, Baltimore, MD 21218, USA
[4] Harvard-Smithsonian Center for Astrophysics, 60 Garden Street, Cambridge, MA 02138, USA
[5] Department of Physics and Astronomy, Minnesota State University Moorhead, 1104 7th Avenue South, Moorhead, MN 56563, USA
[6] Max-Planck-Institut für Kernphysik, Heidelberg, Germany
[7] Department of Astrophysics and Planetary Science, Villanova University, 800 East Lancaster Avenue, Villanova, PA 19085, USA
[8] NASA Goddard Space Flight Center, Astrochemistry Laboratory, 8800 Greenbelt Road, Greenbelt, MD 20771, USA
[9] Lowell Observatory, 1400 W Mars Hill Road, Flagstaff, AZ 86001, USA
[10] Astronomy Department, University of Washington, Seattle, WA 98195, USA
[11] Winter Way, Uppsala, Sweden
[12] ESAC Science Data Centre, European Space Agency, Madrid, Spain
[13] Department of Space, Earth and Environment, Chalmers University of Technology, Onsala Space Observatory, 439 92, Onsala, Sweden
[14] Aperio Software Ltd., Headingley Enterprise and Arts Centre, Bennett Road, Leeds, LS6 3HN, UK
[15] Astroparticle Physics Laboratory, NASA Goddard Space Flight Center, Mail Code 661, Greenbelt, MD 20771, USA
[16] Joint Space-Science Institute, University of Maryland, College Park, MD 20742, USA
[17] Department of Physics, Catholic University of America, Washington, DC 20064, USA
[18] European Space Astronomy Centre, European Space Agency, Madrid, Spain
[19] European Southern Observatory, Karl-Schwarzschild-Str. 2, D-85748 Garching bei München, Germany
[20] DIRAC Institute, Department of Astronomy, University of Washington, 3910 15th Avenue NE, Seattle, WA 98195, USA
Received 2018 August 28; revised 2019 January 2; accepted 2019 January 3; published 2019 February 6

Abstract

`astroquery` is a collection of tools for requesting data from databases hosted on remote servers with interfaces exposed on the internet, including those with web pages but without formal application program interfaces. These tools are built on the Python `requests` package, which is used to make HTTP requests, and `astropy`, which provides most of the data parsing functionality. `astroquery` modules generally attempt to replicate the web page interface provided by a given service as closely as possible, making the transition from browser-based to command-line interaction easy. `astroquery` has received significant contributions from throughout the astronomical community, including several from telescope archives. `astroquery` enables the creation of fully reproducible workflows from data acquisition through publication. This paper describes the philosophy, basic structure, and development model of the `astroquery` package. The complete documentation for astroquery can be found at http://astroquery.readthedocs.io/.

Key words: astronomical databases: miscellaneous – virtual observatory tools

1. Introduction

In the past few decades, large-scale surveys have played a huge role in advancing our understanding of the universe, and these surveys have produced enormous reservoirs of data that astronomers regularly access. However, tools for accessing these reservoirs are heterogeneous and often only available via graphical user interfaces or web sites.

One of the cornerstones of research is reproducibility. To be able to reproduce research, the data need to be available to everyone. Many scientific journals encourage or demand that the underlying data accompany the article or be uploaded to a hosting service. Data sharing is not only important for new results, but also to provide the ability to test and verify published results. While many different efforts to promote data sharing have made the practice more common, it is difficult to keep track of how and where to retrieve a given data set. A common scripted interface to tie all these services together is a good way to make all the different data more accessible, and it provides authors with the ability to make the full analysis process they used—from data download to publication—repeatable. A centrally maintained library also safeguards against inevitable "link rot" on data archives, moving some of the responsibility for maintaining long-term reproducibility from each individual researcher to the broader community.

Data sharing has taken on a variety of forms. The most prominent are the major observatory archives: MAST, NOAO, ESO, ESA, IPAC, CDS, NRAO, CXC, HEASARC, and CADC are the main organizations hosting raw and processed data from ground- and space-based telescopes. These data archives also serve as the primary means for serving data to users when the data are taken in queue mode, i.e., while the observer is not on-site.

In addition to observatories and telescopes, individual surveys often share their full data sets. In some cases, these

[21] Google Summer of Code Student
[22] Google Summer of Code

are shared via the observatory that acquired them, for example, the all-sky data acquired with Planck, *Wilkinson Microwave Anisotropy Probe*, and COBE. Other surveys, particularly ground-based surveys, serve their own data. Examples include the SDSS, 2MASS, UKIDSS, and VSA.

Individual teams and small groups often share their data via their own custom websites. These services do not follow any particular standard and can be widely varied in the type and amount of data shared. Sometimes these data are shared via the archive systems (e.g., IRSA at IPAC hosts many individual survey data sets), while others use their own web hosting systems (e.g., MAGPIS).

Finally, there are other data types relevant to astronomy that are not served by the typical astronomical databases. Examples include databases of molecular and atomic properties, such as those provided by Splatalogue and the NIST Atomic Spectra Database, bibliographic databases such as the NASA Astrophysics Data System (ADS), or services that are computationally intensive or require regular updates, like solar system ephemerides provided by services like JPL HORIZONS, or the Minor Planet Center.

`astroquery` arose from a desire to access these databases from the Python command line in a scriptable fashion. Script-based data access provides astronomers with the ability to make reproducible analysis scripts and pipelines in which the data are retrieved and processed into scientifically relevant results with minimal user interaction.

In this paper, we provide an overview of the `astroquery` package. Section 2 describes the basic layout of the software and the shared API concept underlying all modules. Section 3 describes the development model. Finally, Section 4 describes how `astroquery` is documented.

2. The Software

`astroquery` consists of a collection of modules that mostly share a similar interface, but are meant to be used independently. They are primarily based on a common framework that uses the Python `requests`[23] package to perform HTTP requests to communicate with web services. A list of supported services at the time of publication is given in Table 1.

For new module development, there is a `template_module` consisting of a folder with several individual python code files that lays out the basic framework of any new module. All modules have a single core `class` that has some number of `query_*` methods. The most common query method is `query_region`, which usually provides a "cone search" functionality, i.e., they search for data within a circular region. The results of the queries then are returned in an `astropy` (Astropy Collaboration et al. 2013, 2018) `table`.[24]

An example using the SIMBAD interface is shown below:[25]

Example 1. Query SIMBAD for a region around M81

```
from astroquery.simbad import Simbad
result_table = Simbad.query_region("m81")
```

In this example, `SIMBAD` is an instance of `astroquery.simbad.SimbadClass`, and `result_table` is an `astropy.table.Table` containing the objects near M81. This common interface allows users to use different services and process the resulting data in the same manner despite the differences in the underlying methods and services (e.g., `SDSS.query_region()`, `Simbad.query_region()`, `NED.query_region()`, etc.)

While there is a common suggested API described in the `template_module`, individual packages are not *required* to support this API because, for some, it is not possible. For example, the atomic and molecular databases refer to physical data that are not related to positions on the sky and therefore their `astroquery` modules cannot include `query_region` methods. The same applies to solar system object ephemerides queries. Differences in the API are discussed in the `astroquery` documentation[26] (see Section 4).

2.1. Version Numbers

`astroquery` uses the same format as traditional semantic versioning, with versions indicated in the format `MAJOR.MINOR.PATCH.devCOMMIT_ID` (for example, `0.3.9.dev4581`).

`astroquery` patches are frequently made to accommodate upstream changes, i.e., changes made to the remote service, and as such are not guaranteed to be backward-compatible. Thus, starting in mid-2018, `astroquery` switched from a manual release model to a continuous deployment model. Prior to this change, the `MAJOR.MINOR.PATCH` versions were each created manually by one of the maintainers, then pushed to package release services. After this change, each accepted pull request automatically triggered a new release via the Python package index.[27] We created a new manual release, v0.3.9, to accompany the publication of this paper.

2.2. HTTP User-Agent

`astroquery` identifies itself to host services using the `HTTP User-Agent` header data, which is automatically produced and sent to the archives with every request. Users do not need to be aware of these metadata being sent with their queries, but the information can be used by data-hosting services to determine how many users are accessing their service via `astroquery` and to assist in debugging if improper queries are being submitted.

The format of the user agent string is:

```
astroquery/{version}{requests_version}
```

where `{version}` is a version number of the form described in Section 2.1 and `{requests_version}` is the corresponding version of the Python `requests` package. For example:

```
astroquery/0.3.9.dev4863python-requests/2.14.2
```

2.3. The API

The common API has a few features defined in the template module. Each service is expected to provide the following interfaces, assuming they are applicable:

[23] http://docs.python-requests.org/
[24] http://docs.astropy.org/en/stable/table/
[25] http://astroquery.readthedocs.io/en/latest/simbad/simbad.html
[26] The repository associated with this paper is: https://github.com/adamginsburg/astroquery-paper.
[27] https://pypi.org/

Table 1
List of All Services and Surveys `astroquery` Modules Support

Module Name	Service or Organization	URL
`alfalfa`	ALFALFA data repository	http://arecibo.tc.cornell.edu/hiarchive/alfalfa
`alma`	Atacama Large Millimeter/submillimeter Array Archive	http://almascience.org
`atomic`	Atomic Line List	http://www.pa.uky.edu/~peter/atomic
`besancon`	Besancon model of the Galaxy	http://model.obs-besancon.fr
`cds`	Centre de Donnes astronomiques de Strasbourg	http://cds.u-strasbg.fr
`cosmosim`	CosmoSim database	https://www.cosmosim.org/uws/query
`esasky`	ESASky of the European Space Agency	http://sky.esa.int
`eso`	European Southern Observatory Science Archive	http://archive.eso.org/cms.html
`exoplanet_orbit_database`	Exoplanet Orbit Database	http://exoplanets.org
`fermi`	Fermi Gamma-ray Space Telescope Data	https://fermi.gsfc.nasa.gov/ssc/data
`gaia`	*Gaia* Archive of the European Space Agency	https://gea.esac.esa.int/archive
`gama`	Galaxy and Mass Assembly Survey	http://www.gama-survey.org/dr2/query
`heasarc`	High Energy Astrophysics Science Archive Research Center	https://heasarc.gsfc.nasa.gov
`hitran` (Kochanov et al. 2016)	HIgh-resolution TRANsmission molecular absorption database	http://hitran.org/hapi
`ibe`	IRSA Image Server	http://irsa.ipac.caltech.edu/ibe
`irsa`	IRSA Catalog Query Service	https://irsa.ipac.caltech.edu
`irsa_dust`	IRSA Galactic Dust Reddening and Extinction Query	https://irsa.ipac.caltech.edu/applications/DUST
`jplhorizons`	JPL's HORIZONS system	https://ssd.jpl.nasa.gov/horizons_batch.cgi
`jplsbdb`	JPL's Small-Body DataBase	https://ssd-api.jpl.nasa.gov/doc/sbdb.html
`jplspec`	JPL's Spectral Catalog	https://spec.jpl.nasa.gov/cgi-bin/catform
`lamda`	Leiden Atomic and Molecular Database	http://home.strw.leidenuniv.nl/~moldata
`magpis`	The Multi-Array Galactic Plane Imaging Survey	https://third.ucllnl.org/gps
`mast`	Barbara A. Mikulski Archive for Space Telescopes	https://mast.stsci.edu
`mpc`	Minor Planet Center Ephemeris Service	https://minorplanetcenter.net
`nasa_ads`	SAO/NASA Astrophysics Data System	https://api.adsabs.harvard.edu
`nasa_exoplanet_archive`	NASA Exoplanet Archive	https://exoplanetarchive.ipac.caltech.edu
`ned`	NASA Extragalactic Database	https://ned.ipac.caltech.edu
`nist`	NIST Atomic Spectra Database	https://physics.nist.gov/PhysRefData/ASD
`nrao`	National Radio Astronomy Observatory Data Archive	https://archive.nrao.edu/archive
`nvas`	NRAO VLA Archive Survey Images Page	https://archive.nrao.edu/nvas
`oac`	Open Astronomy Catalog	https://astrocats.space
`ogle`	Interstellar Extinction toward the Galactic Bulge from OGLE-III data	http://ogle.astrouw.edu.pl/cgi-ogle/getext.py
`open_exoplanet_catalogue`	Open Exoplanet Catalogue	http://openexoplanetcatalogue.com
`sdss`	Sloan Digital Sky Survey	http://skyserver.sdss.org
`sha`	Spitzer Heritage Archive	http://sha.ipac.caltech.edu/applications/Spitzer/SHA
`simbad`	CDS SIMBAD Astronomical Database	http://simbad.u-strasbg.fr
`skyview`	NASA's SkyView Query	http://skyview.gsfc.nasa.gov
`splatalogue`	Splatalogue Database for astronomical spectroscopy query	https://www.cv.nrao.edu/php/splat
`ukidss`	UKIRT Infrared Deep Sky Survey	http://wsa.roe.ac.uk
`vamdc`	VAMDC molecular line database	https://vamdclib.readthedocs.io/
`vizier`	CDS VizieR Astronomical Catalogues	http://vizier.u-strasbg.fr
`vo_conesearch`	Simple Cone Search Databases	https://astropy.stsci.edu/aux/vo_databases
`vsa`	Vista Science Archive	http://vsa.roe.ac.uk
`xmatch`	CDS X-Match Service	http://cdsxmatch.u-strasbg.fr

1. `query_region` A method that accepts an Astropy `SkyCoord` object representing a point on the sky plus a specification of the radius around which to search. The returned object is an Astropy table.
2. `query_object` A method that accepts the name of an object. This method relies on the service to resolve the object name, i.e., it does not use a name resolver like `SESAME`.[28] The returned object is an Astropy table.
3. `get_images` For services that provide image data, this method accepts an Astropy `SkyCoord` object and a radius to search for data that cover the specified target. The returned object is a list of `astropy.io.fits.HDUList` objects.

We also require a low-level interface to the services so that queries with very large results can be handled by other methods (e.g., data streaming) if needed. The low-level interface consists of a series of methods with the same names, but with the additional suffix `_async` (e.g., `query_async`). The `query*_async` methods return a `requests.Response` object from the accessed website, providing developers with the ability to access the data in a stream or access only the response metadata (i.e., the `async` methods do not download the corresponding data, so they may be useful for collecting metadata for very large files). The `get_images_async`

[28] http://cds.u-strasbg.fr/cgi-bin/Sesame

method returns `FileContainer` objects that similarly provide "lazy" access to the data, but specifically for FITS files. Contributors need only implement these `_async` methods because there is a wrapper tool that converts `_async` methods into their corresponding non-asynchronous versions.

Deviations from this standard API are documented in the `astroquery` documentation (see Section 4). Most deviations are for services for which `query_region` methods are not defined, such as atomic and molecular line databases.

2.4. Caching and Login Functionality

astroquery provides tools to handle multiple aspects of querying that are common to all modules. The `BaseQuery` metaclass provides tools for caching requests and downloaded data, reducing the duration and the network load for repeated queries. Cached data are stored in the user's `~/.astropy/cache/astroquery` directory. The `BaseQuery` metaclass is also responsible for setting the User-Agent (Section 2.2). The `QueryWithLogin` metaclass provides a framework for logging in securely to services that require user authentication, including a credential storage mechanism.

2.5. Error Handling

Some queries will inevitably fail. Failures can take on different modes. For common and expected modes, such as searching for an object or location on the sky and getting no results, the result is clearly communicated as a simple null result or empty table. For unpredictable and unexpected errors, such as server failures, timeouts, and other related communication issues, the errors are handled by the `requests` module, and normal HTTP responses are returned (e.g., HTTP 200 means the request was successful, while 503 indicates the request was forbidden by server-side permissions; a complete list can be found at https://en.wikipedia.org/wiki/List_of_HTTP_status_codes).

In some cases, when we know a particular failure mode is likely (because the developers have encountered it at least once), we catch and raise a specific `Exception` or `Warning`. The full list of these is in the `exceptions.py` file. Developers can use these custom exceptions to build in additional robustness to data pipelines using astroquery by either implementing workarounds to known issues or correctly informing users of the problem.

2.6. Testing

Astroquery testing is somewhat different from most other packages in the scientific Python ecosystem. While the tests are based on the Astropy testing infrastructure and use pytest to run and check the outputs, the astroquery tests are split into *remote* and *local*. The remote tests exactly replicate what a user would enter at the command line, but they are dependent on the stability of the remote services.

In our experience it is quite rare for all of the astroquery-supported services to be accessible simultaneously.[29] We therefore require that each module provide some tests that do not rely on having an internet connection. These tests rely on *monkeypatching*[30] to replace the remote requests. Instead of downloading data, the test suite uses locally available files to test the query mechanisms and the data parsers. Monkeypatching in the context of pytest results in code that is generally more difficult to understand than typical Python code, but a set of tests independent of the remote services is necessary.

The local tests are run as part of the continuous integration for the project with each commit. The remote tests are run for merges and as part of a regularly-scheduled cron job. Running the remote tests less frequently helps reduce the burden on the remote services.

2.7. Other Utilities

There are several general-use utilities implemented as part of astroquery, such as a bulk FITS file downloader and renamer and a download progressbar (these tools complement similar features in Astropy). There is also a schema system implemented to allow user-side parameter validation. The schema systems are basic syntax-checking tools that verify that the parameters the user has input are of the right type and format for the target service; for those services without schemas, the user can hypothetically send queries that the service will be unable to handle. The schema tool is only implemented in the ESO and Vizier modules, but it could be expanded to other modules to reduce the number of doomed-to-fail queries sent through astroquery.

3. Development History and Status

Anyone can contribute to astroquery. The maintainers are committed to helping developers make new modules that meet the requirements of astroquery. This section describes how astroquery has been developed, but we welcome all sorts of new contributions, including new modules, upgrades to existing modules, and minor corrections to existing tools from both individuals and institutions.

astroquery is an Astropy coordinated package (Tollerud 2018) and is a critical component of the Astropy Project ecosystem (Astropy Collaboration et al. 2018). It is a standalone project and will remain independent of the `astropy` core package,[31] but is coordinated by the Astropy Project to ensure sustainability and maintenance.

astroquery has received contributions from 77 people as of 2018 August. While the primary maintenance burden is shouldered by two people at any given time (the first two authors), most individual modules have been implemented independently by interested contributors.

Some contributions have come with direct institutional support. The ESA *Gaia* and ESASky modules were provided and supported by developers working for ESA. The ADS module is maintained by developers working at ADS. The MAST and Virtual Observatory (VO) Cone Search query tools were added by developers at STScI, with the latter moved over from `astropy.vo` (see Section 3.1).

[29] While this issue affects testing, it rarely affects users, since simply retrying a query is often enough to fix user issues. When the servers are simply down or broken, astroquery is affected, and the resulting errors are sometimes unpredictable; users are encouraged to report such failures as github issues (https://github.com/astropy/astroquery/issues) so that better error messages can be provided.

[30] Monkeypatching is the dynamic replacement of attributes at runtime, i.e., changing what functions do after they are imported.

[31] Many Astropy affiliated packages are developed with the intent of eventually including them in the core of `astropy`. In contrast, astroquery intends to remain a separate package indefinitely largely because of its need to rapidly adapt to changes in the remote services; `astropy` cannot make such rapid changes because users rely on its stability.

astroquery also receives contributions from other funded programs. For instance, the JPLHorizons module has been implemented as part of the `sbpy` project[32] with support from NASA. Further solar system-related services are planned to be added to astroquery through this support. astroquery has also received support from the Google Summer of Code program, with two students (co-authors Madhura Parikh and Simon Liedtke) from 2013 to 2014.

Due to its nature as an openly developed package, new directions in astroquery are primarily driven by contributors and data providers adding or updating modules to reflect new or changed data sources. The underlying software architecture has been demonstrably sufficient to meet the needs of the current generation of data sources (proven by the user base of astroquery). While this policy may change in the future, the user-focused nature of astroquery means that making such architecture changes is unnecessary until there are specific data sources or use cases to drive them.

3.1. Relation to the VO

The VO has some goals similar to astroquery, though their approach and philosophy is different. Where VO services provide a single point of access for all VO-compatible services, astroquery provides a collection of access points that do not require a specific API from the hosting service. The general philosophy in astroquery is to replicate the web page interface provided by a given service as closely as possible. While this approach makes some versions of cross-archive searches more difficult, it keeps the barrier to entry for new users fairly low and limits the maintenance burden for upstream developers.

However, there are developments in progress to allow more VO-like queries within astroquery, such as searching for databases by keywords. As more services implement VO-based access, some query modules may adopt VO as a backend, but these changes should be transparent to users (i.e., the astroquery interfaces will remain unchanged). The documentation may guide users on how to use the more sophisticated VO tools that underlie these tools.

Some general VO tools are available in astroquery. The `vo_conesearch` package, which originally resided in `astropy`, is now part of astroquery. VO Cone Search has a `query_region` interface like the other astroquery services in addition to the existing interfaces ported over from Astropy. As of `astropy` 3.0, `astropy.vo` no longer exists; therefore, astroquery is now the primary provider of this VO Cone Search service. From a typical user's standpoint, switching over from `astropy.vo` should result in no difference except for updating their Python `import` statements (e.g., `from astroquery.vo_conesearch import conesearch` instead of `from astropy.vo.client import conesearch`).

4. Documentation and References

4.1. Online Documentation

The astroquery modules are documented online and can be accessed at https://astroquery.readthedocs.io/. We include one detailed example of how to use astroquery in the Appendix, but interested users will find many more on the documentation page and in the example gallery.[33]

[32] http://sbpy.org

[33] https://astroquery.readthedocs.io/en/latest/gallery.html

4.2. Other Documents

Several authors have independently described how to use various astroquery modules, which is a helpful practice we encourage.

1. Cosmosim:[34] a worked example of downloading data from the cosmosim database, including logging in.
2. Paletou & Zolotukhin (2014): a worked example of querying Vizier and SIMBAD to make a surface gravity–effective temperature plot for a star survey.
3. Guillochon & Cowperthwaite (2018): the definition of the Open Astronomy Catalog API and a description of the astroquery module built to use it.
4. MAST:[35] A tutorial on the MAST astroquery interface.
5. *Gaia*:[36] A tutorial on the *Gaia* astroquery interface.

5. Summary

astroquery is a toolkit for accessing remotely hosted astronomical data through Python. It is part of the Astropy affiliated package system. We have described its general layout, its development model, and its role in developing reproducible workflows. astroquery is developed for and by our community: we welcome any new contributions, and such contributions will continue to define the future directions of the package.

We would like to thank the members of the community that have contributed to `astroquery`, that have opened issues and provided feedback, and have supported the project in a number of different ways. We are grateful for the infrastructural support the Astropy community provides. `astroquery` is supported by and makes use of a number of organizations and services outside the traditional academic community: GitHub, Travis CI, Appveyor, and Read the Docs. Our package relies heavily on the following Python dependencies; we are grateful for their maintainers and contributors: `requests` `beautifulsoup`, and `keyring`.

We thank Google for financing and organizing the Google Summer of Code program, that has funded two students (S.L. and M.P.) to work on `astroquery` in 2013 and 2014.

The following individuals would like to recognize support for their personal contributions. B.M.S. is supported by the NSF grant AST-1715122 and acknowledges support from the DIRAC Institute in the Department of Astronomy at the University of Washington. The DIRAC Institute is supported through generous gifts from the Charles and Lisa Simonyi Fund for Arts and Sciences, and the Washington Research Foundation. The contributions of M.M., M.V.B., G.G. are supported by the NASA PDART grant 80NSSC18K0987. The National Radio Astronomy Observatory is a facility of the National Science Foundation operated under cooperative agreement by Associated Universities, Inc.

Software: Astropy (Astropy Collaboration et al. 2018), `numpy` (Van der Walt et al. 2011), `requests`, `keyring`, `beautifulsoup4`, `html5lib`, `matplotlib` (Hunter 2007), `APLpy` (Robitaille & Bressert 2012), `pyregions`,[37] `regions`[38]

[34] https://www.cosmosim.org/cms/news/cosmosim-package-for-astroquery/

[35] https://github.com/spacetelescope/MAST-API-Notebooks/blob/master/AstroqueryIntro/AstroqueryFunctionalityDemo.ipynb

[36] https://gea.esac.esa.int/archive-help/tutorials/python_cluster/index.html

[37] https://github.com/astropy/pyregions.

[38] https://github.com/astropy/regions.

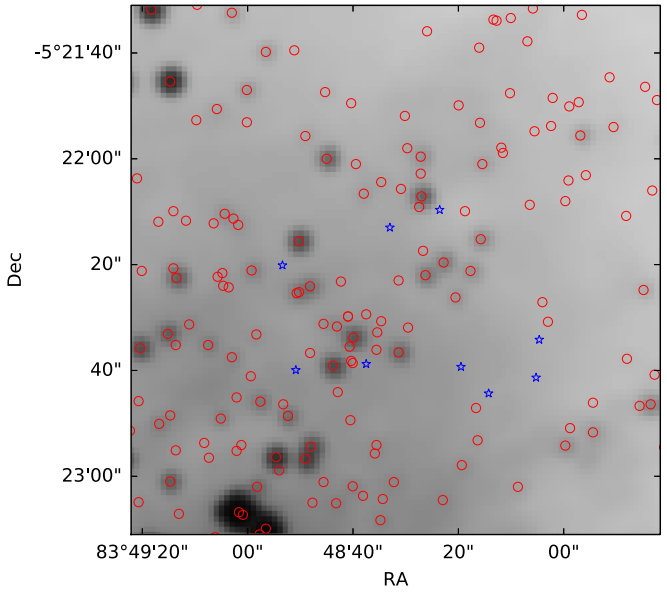

Figure 1. Example figure made using astroquery. The `skyview` package was used to download a 2MASS *J*-band image. The `vizier` was used to download two star catalogs from different publications and overplot them; the blue stars show sources from the older, less complete catalog and the red circles show sources from a more recent, more complete catalog.

Appendix
Example

In this appendix, we show an example of astroquery in action, highlighting the ability to use multiple modules and interact with `astropy`'s table, coordinate, and unit tools. This example, shown in Figure 1, approximately reproduces Figure 1 of Eisner et al. (2016), but with a different background. It can also be found on astroquery's gallery page (http://astroquery.readthedocs.io/en/latest/gallery.html). Another illustration of how to use astroquery tools in a finder chart making tool is `fcmaker`, which produces charts for ESO observations using astroquery (Vogt 2018).

```
# Create a finder chart and overlay two catalogs using the
  Vizier and SkyView
# tools
from astropy import units as u
from astropy.coordinates import SkyCoord
from astropy.wcs import WCS
from astroquery.skyview import SkyView
from astroquery.vizier import Vizier
import matplotlib.pyplot as plt

center = SkyCoord.from_name("Orion KL")

# Grab an image from SkyView of the Orion KL nebula region
imglist = SkyView.get_images(position=center,
  survey="2MASS-J")

# The returned value is a list of images, but there is
  only one
img = imglist[0]

# "img" is now a fits.HDUList object; the 0th entry is the
  image
mywcs = WCS(img[0].header)

fig = plt.figure(1)
fig.clf()  # Just in case one was open before
```

(Continued)

```
# Use astropy's wcsaxes tool to create an RA/Dec image
ax = fig.add_axes([0.15, 0.1, 0.8, 0.8],
  projection=mywcs)
ax.set_xlabel("RA")
ax.set_ylabel("Dec")

ax.imshow(img[0].data, cmap="gray_r", inter-
  polation="none", origin="lower",
      norm=plt.matplotlib.colors.LogNorm())

# Retrieve a specific table from Vizier to overplot
tablelist = Vizier.query_region(
   center, radius=5*u.arcmin, catalog="J/ApJ/826/16/
  table1")
# Again, the result is a list of tables, so we"ll get the
  first one
result = tablelist[0]

# Convert the ra/dec entries in the table to astropy
  coordinates
tbl_crds = SkyCoord(result["RAJ2000"], result
  ["DEJ2000"],
           unit=(u.hour, u.deg), frame="fk5")

# We want this table too:
tablelist2 = Vizier(row_limit=10000).query_region(
   center, radius=5*u.arcmin, catalog="J/ApJ/540/236")
result2 = tablelist2[0]
tbl_crds2 = SkyCoord(result2["RAJ2000"], result2
  ["DEJ2000"],
           unit=(u.hour, u.deg), frame="fk5")

# Overplot the data in the image
ax.plot(tbl_crds.ra, tbl_crds.dec, "*", transform=ax.
  get_transform("fk5"),
     mec="b", mfc="none")
ax.plot(tbl_crds2.ra, tbl_crds2.dec, "o", transform=ax.
  get_transform("fk5"),
     mec="r", mfc="none")
# Zoom in on the relevant region
ax.axis([100,200,100,200])

plt.show()
```

ORCID iDs

Adam Ginsburg https://orcid.org/0000-0001-6431-9633
Brigitta M. Sipőcz https://orcid.org/0000-0002-3713-6337
C. E. Brasseur https://orcid.org/0000-0002-9314-960X
Philip S. Cowperthwaite https://orcid.org/0000-0002-2478-6939
Matthew W. Craig https://orcid.org/0000-0001-7988-8919
Christoph Deil https://orcid.org/0000-0002-4198-4005
James Guillochon https://orcid.org/0000-0002-9809-8215
Giannina Guzman https://orcid.org/0000-0001-6340-8220
Pey Lian Lim https://orcid.org/0000-0003-0079-4114
Kelly E. Lockhart https://orcid.org/0000-0002-8130-1440
Michael Mommert https://orcid.org/0000-0002-8132-778X
Brett M. Morris https://orcid.org/0000-0003-2528-3409
Henrik Norman https://orcid.org/0000-0002-8898-4015
Magnus V. Persson https://orcid.org/0000-0002-1100-5734
Thomas P. Robitaille https://orcid.org/0000-0002-8642-1329
Leo P. Singer https://orcid.org/0000-0001-9898-5597

Erik J. Tollerud https://orcid.org/0000-0002-9599-310X
Miguel de Val-Borro https://orcid.org/0000-0002-0455-9384
Ivan Valtchanov https://orcid.org/0000-0001-9930-7886
Julien Woillez https://orcid.org/0000-0002-2958-4738

References

Astropy Collaboration, Price-Whelan, A. M., Sipőcz, B. M., et al. 2018, arXiv:1801.02634
Astropy Collaboration, Robitaille, T. P., Tollerud, E. J., et al. 2013, A&A, 558, A33
Eisner, J. A., Bally, J. M., Ginsburg, A., & Sheehan, P. D. 2016, ApJ, 826, 16
Guillochon, J., & Cowperthwaite, P. S. 2018, RNAAS, 2, 27
Hunter, J. D. 2007, CSE, 9, 90
Kochanov, R. V., Gordon, I. E., Rothman, L. S., et al. 2016, JQSRT, 117, 15
Paletou, F., & Zolotukhin, I. 2014, arXiv:1408.7026
Robitaille, T., & Bressert, E. 2012, APLpy: Astronomical Plotting Library in Python, Astrophysics Source Code Library, ascl:1208.017
Tollerud, E. 2018, Astropy Proposal for Enhancement 15: An Updated Model for the Affiliated Package Ecosystem (APE 15), Zenedo, doi:10.5281/zenodo.1246834
Van der Walt, S., Colbert, S. C., & Varoquaux, G. 2011, CSE, 13, 22
Vogt, F. P. A. 2018, arXiv:1807.02114v1