Gaia on TAP

Accessing Gaia data with ESA's API using Python

Andy Casey (Cambridge)

What the hell's an API?

- Application Programming Interface
- Access data by making requests to ESA's website
- Follow a protocol: **TAP** (Table Access Protocol)
- Query language: **ADQL** (Astronomical Data Query Language)
- Return data formats: VO table / CSV / JSON

gea.esac.esa.int/tap-server/tap

TAP HOME PAGE

Available resources:

- jobs
- capabilities
- functions
- sync
- async
- admin
- availability
- users
- scheduler
- notification
- tables
- stats
- deletejobs
- share
- event
- tasks

astropy: a community library for astronomy

astropy

A Community Python Library for Astronomy

Welcome to the Astropy documentation! Astropy is a community-driven package intended to contain much of the core functionality and some common tools needed for performing astronomy and astrophysics with Python.

User Documentation

What's New in Astropy 1.2?

Astropy at a glance

- Overview
- Installation
- Getting Started with Astropy
- Example gallery

Core data structures and transformations

- Constants (`astropy.constants`)
- Units and Quantities (`astropy.units`)
- N-dimensional datasets (`astropy.nddata`)
- Data Tables (`astropy.table`)
- Time and Dates (`astropy.time`)
- Astronomical Coordinate Systems (`astropy.coordinates`)
- World Coordinate System (`astropy.wcs`)
- Models and Fitting (`astropy.modeling`)
- Analytic Functions (`astropy.analytic_functions`)

Connecting up: Files and I/O

This repository Search Pull requests Issues Gist

andycasey / gaia-on-tap

Unwatch 1 Star 0 Fork 0

Code Issues 0 Pull requests 0 Projects 0 Wiki Pulse Graphs Settings

No description or website provided. — Edit

6 commits 1 branch 0 releases 1 contributor MIT

Branch: master New pull request Create new file Upload files Find file Clone or download

andycasey Auth utilities and a cone search Latest commit 5815125 an hour ago

code	Auth utilities and a cone search	an hour ago
.gitignore	Update .gitignore	2 hours ago
LICENSE	Initial commit	a day ago
README.md	Config and other shit	a day ago
REQUIREMENTS.md	Initial commit	a day ago
setup.py		a day ago

github.com/andycasey/gaia-on-tap

README.md

Gaia on TAP

Python utilities and examples for accessing ESA Gaia data using Table Access Protocol (TAP).

Basic usage

1. You provide some ADQL query, or position for a cone-search.

2. Code executes the query on the Gaia archive *(including authentication)*, and returns an astropy table.

3. You can do what you want with the results, write it to disk in any format (CSV/FITS/ASCII,.. and many others).

Basic usage

```python
import gaia.tap


# Execute an ADQL query on any of the ESA Gaia tables
sources = gaia.tap.query("SELECT TOP 5 source_id, ra, dec FROM gaiadr1.gaia_source")


# Perform a basic cone search.
sources = gaia.tap.cone_search(ra, dec, radius, table="gaiadr1.gaia_source")


# Upload a local table and perform some cross-match on it.
sources = gaia.tap.query()


# Perform queries on private tables stored in the ESA Gaia archive.
gaia.config.read("credentials.yaml") # Read our credentials from a file
sources = gaia.tap.query(" ... ", authenticate=True)
```

Example #1: Basic cone search

In [1]:
```
from gaia.tap import cone_search

# Get all sources within a 0.25 degree radius
sources = cone_search(32.341, -1.4245, 0.25)
```

...

In [2]:
```
print(len(sources))
```

```
412
```

In [5]:
```
print(sources.dtype.names)
```

```
('solution_id', 'source_id', 'random_index', 'ref_epoch', 'ra', 'ra_error', 'dec', 'dec_erro
r', 'parallax', 'parallax_error', 'pmra', 'pmra_error', 'pmdec', 'pmdec_error', 'ra_dec_cor
r', 'ra_parallax_corr', 'ra_pmra_corr', 'ra_pmdec_corr', 'dec_parallax_corr', 'dec_pmra_cor
r', 'dec_pmdec_corr', 'parallax_pmra_corr', 'parallax_pmdec_corr', 'pmra_pmdec_corr', 'astrom
etric_n_obs_al', 'astrometric_n_obs_ac', 'astrometric_n_good_obs_al', 'astrometric_n_good_obs
_ac', 'astrometric_n_bad_obs_al', 'astrometric_n_bad_obs_ac', 'astrometric_delta_q', 'astrome
tric_excess_noise', 'astrometric_excess_noise_sig', 'astrometric_primary_flag', 'astrometric_
relegation_factor', 'astrometric_weight_al', 'astrometric_weight_ac', 'astrometric_priors_use
d', 'matched_observations', 'duplicated_source', 'scan_direction_strength_k1', 'scan_directio
n_strength_k2', 'scan_direction_strength_k3', 'scan_direction_strength_k4', 'scan_direction_m
ean_k1', 'scan_direction_mean_k2', 'scan_direction_mean_k3', 'scan_direction_mean_k4', 'phot_
g_n_obs', 'phot_g_mean_flux', 'phot_g_mean_flux_error', 'phot_g_mean_mag', 'phot_variable_fla
g', 'l', 'b', 'ecl_lon', 'ecl_lat')
```

Example #1: Basic cone search

In [7]:
```
print(sources)
```

```
    solution_id          source_id      ...       ecl_lat
                                        ...     Angle[deg]
------------------- ------------------- ... ------------------
1635378410781933568 2506440212688204928 ... -13.422344016244464
1635378410781933568 2506438769579515520 ... -13.427802981366289
1635378410781933568 2506438013664903552 ... -13.454276092921727
1635378410781933568 2506439766011608192 ... -13.390650272792959
1635378410781933568 2506440212688206208 ... -13.418639066177315
1635378410781933568 2506440453206374656 ... -13.407713414419314
1635378410781933568 2506816215599741696 ... -13.384253796101744
1635378410781933568 2506432138149868672 ...  -13.47837119292898
1635378410781933568 2506439594212909568 ... -13.402140010311244
1635378410781933568 2506431347875552256 ... -13.517394621370308
                  ...                 ... ...                ...
1635378410781933568 2494403996337491712 ... -13.739208842106907
1635378410781933568 2494426708124785024 ... -13.526261094035627
1635378410781933568 2494426879923478016 ... -13.579813497377675
1635378410781933568 2494425574253487744 ... -13.574072870321281
1635378410781933568 2494429044587062656 ... -13.540296495127981
1635378410781933568 2494399289053303936 ... -13.856374668530997
1635378410781933568 2494427326600076288 ... -13.530523143647232
1635378410781933568 2494428323032557312 ...  -13.48487739099413
1635378410781933568 2494412414473630080 ... -13.680409701156519
1635378410781933568 2494802294424784640 ... -13.502758909638393
Length = 412 rows
```

In [8]:
```
sources.write("cone_search.csv")
```

Example #2: Cone search with SIMBAD

In [1]:
```
import astropy.coordinates as coord
import matplotlib.pyplot as plt

from gaia.tap import cone_search


cluster = coord.SkyCoord.from_name("M67")

# Get everything within 1 degree radius of the cluster.
cluster_candidates = cone_search(cluster.ra.deg, cluster.dec.deg, 1.0)

# Plot it.
fig, ax = plt.subplots()
ax.scatter(cluster_candidates["ra"], cluster_candidates["dec"],
    s=1, c="#000000")
ax.set_xlabel(r"$\alpha$")
ax.set_ylabel(r"$\delta$")
```

Example #2: Cone search with SIMBAD

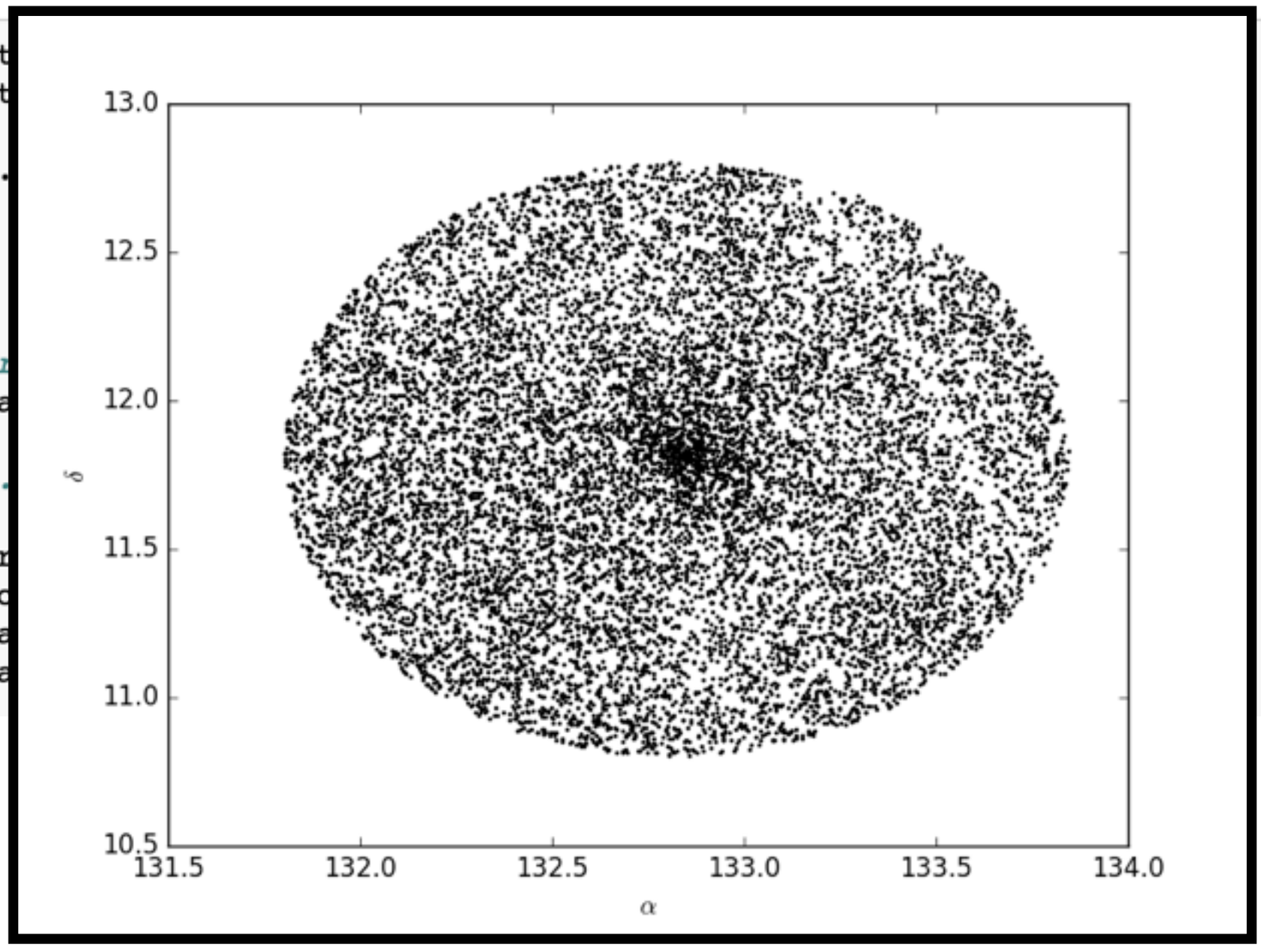
```
In [1]: import ast
        import mat

        from gaia.

        cluster =

        # Get ever
        cluster_ca

        # Plot it.
        fig, ax =
        ax.scatter
            s=1, c
        ax.set_xla
        ax.set_yla
```



Example #3:
Upload a table and cross-match it against Gaia

```
from code import config, tap

# We *must* authenticate to upload tables to our private user space.
config.read("credentials.yaml")

# Upload the table.
tap.upload_table("foobar", "test.votable")

# Download the table.
# (Note that if you don't authenticate, you can't access your tables - duh!)
sources = tap.query("SELECT * FROM user_acasey.foobar", authenticate=True)
```

(There are some practical technical caveats here — ask me later)

Example #3:
Upload a table and cross-match it against Gaia

```
# Cross-match your uploaded table against tables that are in the ESA/Gaia archive

# Use a 1.5" cone.
sources = tap.query(
    """ SELECT  *
        FROM    gaiadr1.gaia_source as gaia,
                user_acasey.foobar as my_table
    WHERE   1=CONTAINS(
                POINT('ICRS', my_table.ra, my_table.dec),
                CIRCLE('ICRS', gaia.ra, gaia.dec, 1.5/3600)
                )
                """,
authenticate=True)
```

Example #4:
Find hypervelocity stars in TGAS

Find stars that have high tangential velocities in TGAS

```
import astropy.coordinates as coord
import astropy.units as u
import matplotlib.pyplot as plt

import gaia.tap

# Identify potential hypervelocity stars (Vt > 500 km/s).
hvs_candidates = gaia.tap.query(
    """ SELECT  *
        FROM    gaiadr1.tgas_source
        WHERE   parallax_error/parallax < 0.2
          AND   (4.74 * SQRT(POWER(pmra, 2) + POWER(pmdec, 2)))/parallax > 500 """)
```

In [4]:
```
print(hvs_candidates)
```

```
 hip    tycho2_id  ...      ecl_lon             ecl_lat
                   ...    Angle[deg]          Angle[deg]
------ ---------- ... ------------------ ------------------
    --  2349-686-1 ... 57.245902898732503  14.87231382451788
    -- 3011-2171-1 ... 146.78809068069862 34.967067046368655
    --  5319-474-1 ... 59.799149450714083 -34.027135797044259
    --  2116-128-1 ... 287.54539005450567 50.778095475109254
   756          -- ... 345.05625746791395 -34.741929147520679
```

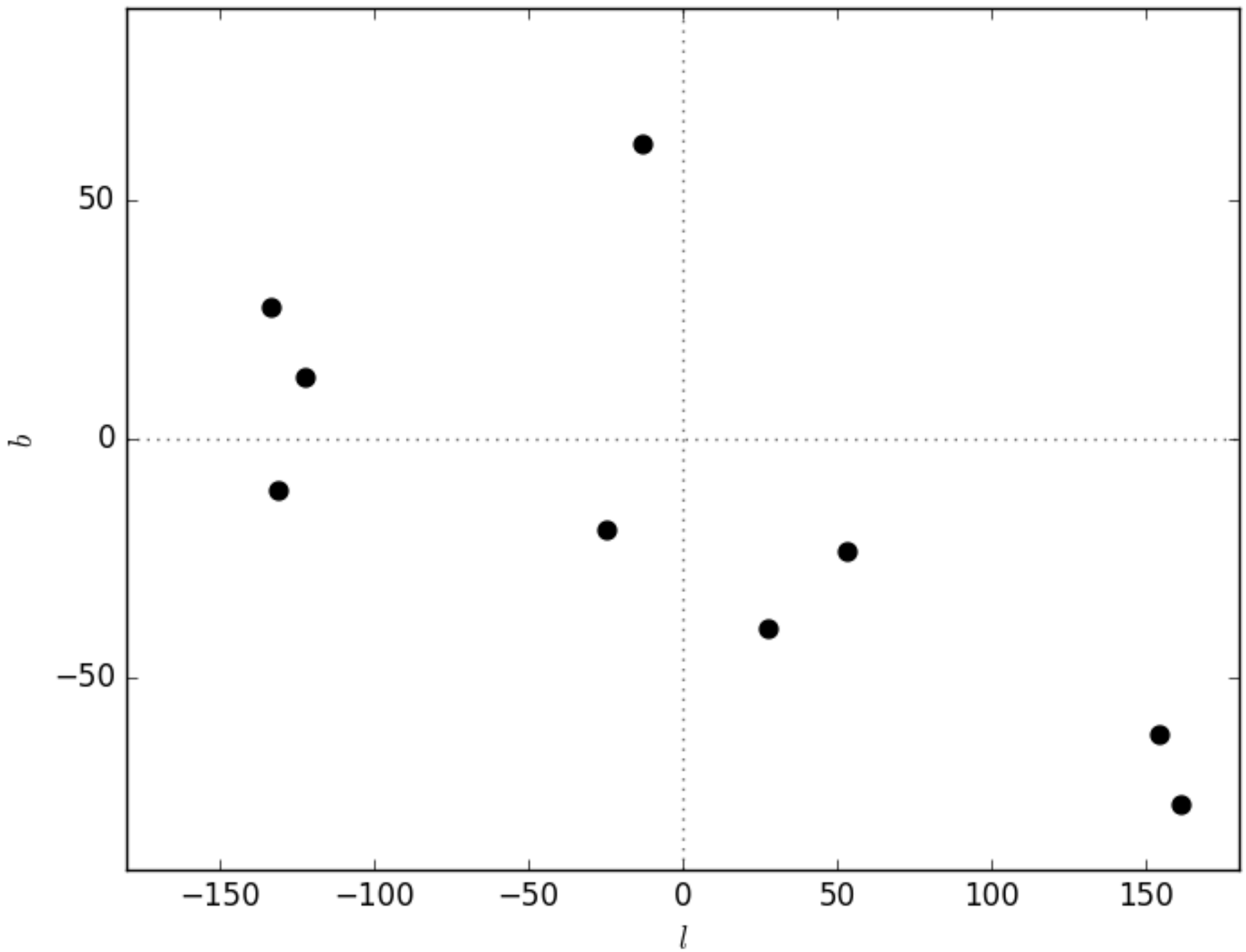
Example #4:
Find hypervelocity stars in TGAS

```
# Identify potential hypervelocity stars.
hvs_candidates = gaia.tap.query(
    """ SELECT  *
        FROM    gaiadr1.tgas_source
        WHERE   parallax_error/parallax < 0.2
          AND   (4.74 * SQRT(POWER(pmra, 2) + POWER(pmdec, 2)))/parallax > 500 """)

# Use astropy to convert the observed positions.
c = coord.SkyCoord(
    ra=hvs_candidates["ra"].data * u.degree,
    dec=hvs_candidates["dec"].data * u.degree,
    distance=1.0/hvs_candidates["parallax"] * u.kpc)


fig, ax = plt.subplots()
ax.scatter(c.galactic.l.deg - 180, c.galactic.b.deg, facecolor="k", s=50)
ax.axhline(0, c="#666666", ls=":", zorder=-1)
ax.axvline(0, c="#666666", ls=":", zorder=-1)
ax.set_xlabel(r"$l$")
ax.set_ylabel(r"$b$")
ax.set_xlim(-180, 180)
ax.set_ylim(-90, 90)
```

Example #4:
Find hypervelocity stars in TGAS



Example #5:
Authenticate and access private tables

```
import gaia

# All of the configuration details are stored in gaia.config
# Here you can specify your username and password to access private tables

"""
One way to do this is by having a file (e.g., 'credentials.yaml') that
contains your details in this format:
---
username: acasey
password: my-super-awesome-password
---
"""

# Read our credentials from a file. You only need to do this once.
gaia.config.read("credentials.yaml")

# The authenticate parameter will log you in first, allowing you to access
# any of your private tables, including those shared with you by others.
sources = gaia.tap.query(" ... ", authenticate=True)
```

Example #5:
Have Python and TOPCAT talk to each other

```
import os
import astropy.coordinates as coord
from astropy.vo import samp
from urlparse import urljoin

from gaia.tap import cone_search


# Get the location of 47 Tuc
cluster_name = "NGC 104"
cluster = coord.SkyCoord.from_name(cluster_name)

# Select sources within 0.5 deg of the cluster
cluster_candidates = cone_search(cluster.ra.deg, cluster.dec.deg, 0.5)
cluster_candidates.write("cluster.votable", format="votable")

# Create a SAMP client and send the table to other clients (incl TOPCAT)
client = samp.SAMPIntegratedClient()
client.connect()

client.notify_all({
    "samp.mtype": "table.load.votable",
    "samp.params": {
        "name": cluster_name,
        "url": urljoin("file:", os.path.abspath("cluster.votable"))
    }
})
```

Code, examples, and more resources at:

github.com/andycasey/gaia-on-tap