Community Engagement via Python

Tom Donaldson

IVOA Interop

Northern Spring, 2018

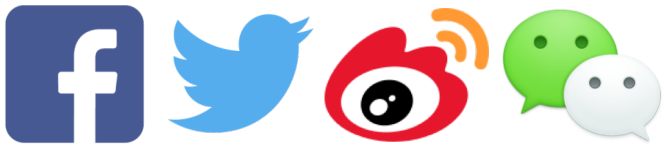
The Need for Community Outreach

VO standards face a perception problem in parts of the astronomy community.

- Partially valid.
  - Some standards created painfully slowly.
  - IVOA seen as a somewhat insular organization, not driven by community needs.
- Partially misplaced expectations
  - "Virtual Observatory" sounds like an end-user tool.
    - Standards intended to enable an ecosystem of such tools.
  - End users not aware they are using VO standards.

IVOA outreach and process improvements

- Actively soliciting input and participation from major projects
- New social media presence (IVOAastro)



- Beginning retooled web presence to make VO more approachable for general community
- Streamlined standards approval process
  - But with added emphasis on demonstrating the utility of the new standards.

Python

Python now commonly used in astronomy data processing and analysis.

- Used everywhere from large-scale data pipelines to end-user applications.
- Success fueled by ease of programming, versatile development environments, and a wide array of general-use and astronomy-specific libraries.
- Central to that success is the Astropy Project, which is

> *"a community effort to develop a common core package for Astronomy in Python and foster an ecosystem of interoperable astronomy packages. "*

- Growing numbers of data collections are now searchable through the affiliated package, AstroQuery.
  - AstroQuery provides a common pattern for data queries, but each query and result is somewhat unique.
  - The VO provides homogenous queries, with results that are somewhat unique.

Integrating VO into astropy seems a natural and necessary step for community engagement

- Astropy and VO have overlapping goals of interoperability and data access
- Enable homogeneous queries (e.g., in a loop) across multiple archives
- Working more directly with end users helps ensure that evolving standards are relevant and useful.

Python Workshop

At AAS 231, NAVO held a workshop: *Using Python to Search NASA's Astrophysics Archives*

- Created Jupyter Notebooks with "real" science cases that used VO services to find data.
  - Main notebook: https://github.com/NASA-NAVO/aas_workshop_2018/blob/master/workshop.ipynb

Forced us to view things from the users' perspective.

- Registry search
  - RegTAP is complex, and thus a barrier to entry for some VO use cases.
  - Nature of metadata makes it hard to find just the desired resources
- Python VO utilities scattered among multiple packages
- Discovered several non-compliant services
  - Caused exceptions in compliant Python utilities
  - Made the results difficult to use
- Some Python utilities did not work with all compliant services.
- Interpreting results difficult to do and explain (UCDs vs. Utypes, required vs not required)

NAVO Python Working Group

NAVO formed a Python working group (T Donaldson initial chair) to help foster collaborations between the IVOA and the Python dev community

Survey what VO software currently exists in Python

- This survey is not complete!
- Excellent VOTable parser (astropy)
- Astroquery
  - Astropy affiliated package
  - Tap and TapPlus
- PyVO
  - Also astropy affiliated package
  - Started in the US
  - Recent contributions from GAVO

NAVO Python Working Group (2)

Continue implementing Python VO use cases

- Service validation
- NAVO performance testing
- Replace HEASARC's Datascope service
- Feedback on what's working well and what's challenging can inform:
  - IVOA protocols and priorities
  - Python package enhancements
- Support more workshops and demonstration notebooks
  - Helps focus efforts on end user experience

API Improvements – Registry Query

```
tap_params = {
    "request":"doQuery",
    "lang":"ADQL",
    "query":"""
        select cap.ivoid,res.short_name,res.res_description,res.reference_url,int.access_url
        from rr.capability cap
        natural join rr.resource res
        natural join rr.interface int
        where cap.cap_type='simplespectralaccess' and cap.ivoid like '%heasarc%'
        order by short_name;
    """
    }
```

```
from astroquery.vo import Registry

# Find all SIA services from HEASARC.
heasarc_image_services = Registry.query(source='heasarc', service_type='image')
```

API Improvements – Image Query

```
m82=coord.SkyCoord.from_name("m82")
pos='{},{}'.format(m82.ra.deg,m82.dec.deg)

params = {'table': 'chanmaster',"POS":pos,"SIZE":".01", "REQUEST":"queryData"}

r = requests.get('https://heasarc.nasa.gov/xamin/vo/ssa', params=params)
```

```
# The query expects a SkyCoord, or any string that can be parsed in to a SkyCoord.
m82=coord.SkyCoord.from_name("m82")

# Perform the query on the 2MASS service.
result_list = Image.query(coords=m82, radius=0, service=heasarc_image_services[24])
```

API Improvements – Image Result Columns

Indirect columns access by UCDs

```
for key in table.columns:

    #Get the data in the column.
    col = table.columns[key]

    #Get the universal content descriptor (UCD) for the column.
    ucdval = col.meta.get('ucd')

    #Find the columns of FITS images and get the images names
    if (ucdval is not None) and (ucdval == 'VOX:Image_Format'):
        imageformats = table[key]
    if (ucdval is not None) and (ucdval == 'VOX:Image_AccessReference'):
        imagenames = table[key]
    if (ucdval is not None) and (ucdval == 'VOX:BandPass_ID'):
        bandpasses = table[key]
```

API Improvements – Image Result Columns (2)

Direct column access by constants

```
# Access the TITLE value in the first row.
print(table[0][ImageColumn.TITLE])

# Note that the title can also be accessed via its actual column name.
print(table[0]['sia_title'])
```

```
2MASS All-Sky Data Release K-Band Atlas Image: 000225 s 088 0080
2MASS All-Sky Data Release K-Band Atlas Image: 000225 s 088 0080
```

```
# Access the entire ACCESS_URL column.
print(table[ImageColumn.ACCESS_URL])
```

```
                                                sia_url
------------------------------------------------------------------------------------------------
https://irsa.ipac.caltech.edu/ibe/data/twomass/allsky/allsky/000225s/s088/image/ki0880080.fits.gz
https://irsa.ipac.caltech.edu/ibe/data/twomass/allsky/allsky/000225s/s088/image/ji0880080.fits.gz
https://irsa.ipac.caltech.edu/ibe/data/twomass/allsky/allsky/000225s/s088/image/hi0880080.fits.gz
```

```
# Or loop through the table row by row.
for row in table:
    print(row[ImageColumn.ACCESS_URL])
```

```
https://irsa.ipac.caltech.edu/ibe/data/twomass/allsky/allsky/000225s/s088/image/ki0880080.fits.gz
https://irsa.ipac.caltech.edu/ibe/data/twomass/allsky/allsky/000225s/s088/image/ji0880080.fits.gz
https://irsa.ipac.caltech.edu/ibe/data/twomass/allsky/allsky/000225s/s088/image/hi0880080.fits.gz
```

API Improvements – Image Result Columns (3)

Image column constants have documentation

```
for imgcol in ImageColumn:
    name = imgcol.name
    required = 'Required' if imgcol.value['required'] else 'Optional'
    ucd = imgcol.value['ucd']
    desc = imgcol.value['description']

    print(f'ImageColumn.{name}  (UCD = {ucd})')
    print(f'({required}) {desc}')
```

```
ImageColumn.TITLE  (UCD = VOX:Image_Title)
(Required)
    A short (usually one line) description of the image. This should concisely
    describe the image to a user, typically identifying the image source
    (e.g., survey name), object name or field coordinates, bandpass/filter, and so forth.

ImageColumn.RA  (UCD = POS_EQ_RA_MAIN)
(Required)
    ICRS right-ascension of the center of the image.

ImageColumn.DEC  (UCD = POS_EQ_DEC_MAIN)
(Required)
    ICRS declination of the center of the image.

ImageColumn.NAXES  (UCD = VOX:Image_Naxes)
(Required)
    The number of image axes.
```

Simplistic Comparison: VO versus Astropy/Astroquery for Data Discovery

Intend to foster an ecosystem of interoperable astronomy components.	
Used by many client tools and data providers.	
Defines web service APIs.	Defines Python APIs.
Supports clients of any language.	Called only by Python.
A short Python loop can locate and download images of M42 for each VO-enabled archive.	Needs separate, but similar, code to access each archive.
Documentation is complex to code to.	Documentation is straightforward to code to.
Queries limited by protocol definitions.	Can ask any query supported by the archive.
APIs evolve slowly through a relatively closed process.	APIs evolve quickly (when needed) in an open process.
Results are difficult to analyze programmatically without robust metadata.	
Challenged by “big” and/or widely distributed data.	

How best to get benefits from both?

- Should we ("the community") consolidate VO client library features?
  - E.g.,an astroquery.vo module
  - Client libraries that implement VO protocols could help manage VO complexity.
    - Lots already written(!), but scattered a bit throughout astropy, astroquery and beyond, with varying levels of consistency and upkeep.
  - How much should we coordinate that development?
  - Coordination could help
    - Make cleaner, more predictable APIs
    - The IVOA identify problematic standards and services.
- Could the IVOA benefit from employing some Astropy-style processes?
- Could Astropy and IVOA collaborate on:
  - Improved metadata description?
  - Data modelling?
  - Scalability of discovery across large or distributed data sets?
- Other ways to leverage each other's benefits?