PyVO and the End User

NAVO's experience from running 4 AAS workshops

Tess Jaffe for NAVO's Python Working group:
Antara Basu-Zych, Clara Brasseur, Vandana Desai, Tom Donaldson, Theresa Dower, David Shupe, Xiuqin Wu

Background

- NASA Astronomical Virtual Observatories consists of HEASARC, IRSA, MAST, NED, each specializing in different regimes, none particularly large (either in bytes or workforce).
- Vandana Desai proposed in 2017 we teach people how to access our archives through Python. So we had to learn! And we've been doing these workshops at most AAS meetings since.
- All of our archives have VO APIs, and we explicitly wanted our methods to be archive agnostic, unlike Astroquery.
- But PyVO was languishing at the time (i.e., didn't quite work for us), so we started with constructing queries with Python's requests library and writing our own wrappers.
- In summer 2019, we switched it all over to PyVO, which is fantastic.
- We think these tutorials should therefore go in PyVO, but we should make sure they are the way we want to teach people to use PyVO.

Goal today

- Make sure PyVO not only powerful, which it already is, but easy to use, which it kind of is.
- Discuss how to teach our users to use PyVO, i.e. what should be the workflow of a standard user, and what functions should be most exposed.

The PyVO workflow

Whether interactively or running an automated script — *and most likely, an iterative combination of the two* — the basic steps are:

1. Step I: search the Registry for services, e.g., offering
   - UV images,
   - the latest Gaia catalog,
   - x-ray spectra,
   - etc.

This isn't common, because each archive is focusing on tutorials specific to their archive and hardwire the service URLs.

2. Step II: ask each service about what it has, e.g.,
   - is there a Swift UVOT observation of Cen A?
   - what information (columns) does Gaia DR2 have?
   - is there a Chandra ACIS HETG spectrum for NGC 1365?

3. Step III: access the data, e.g.,
   - retrieve and view the images,
   - cross-correlate against your catalog, or
   - retrieve and analyze the spectra.

Simple example with PyVO

- Let's find all the available images of M51

```
services = vo.regsearch(servicetype='image')
for service in services:
    try:
        results=service.search(pos=m51_pos, radius=0.1)
        for result in results:
            try:
                print("Downloading {}".format(result.getdataurl()))
                download_file(result.getdataurl())
            except:
                print("Unable to download from {}".format(result.getdataurl()))
            break
    except:
        print("Exception from {};  skipping".format(service.ivoid))
```

Step I: find services

Step II: ask what it has

Step III: get the data

Under the hood (for our reference, not for tutorials or end users to have to know!):

- services:
  - RegistryResults
- services[0]:
  - RegistryResource, with a search() function that exposes the search function and other attributes of the underlying class (TAPService, SIAService etc.)
- services[0].service:
  - the service itself, e.g. TAPService
- results:
  - TAPResults or SIAResults etc., containing the returned VOTable
- results[0]:
  - TAPRecord or SIARecord etc. corresponding to a row in the VOTable

Things to know

- Each archive is responsible for its own backends. They should obey the VO standard, but occasionally there are mistakes.
  - We invite you to contact the archive itself, or file an issue on github.com/nasa-navo/navo-workshop if you want us to look into it.

(?)

- Each archive has its own response and uptime issues. There *will* be servers that sometimes do not respond.
  - Ditto. Furthermore, for scripting loops over services, do NOT forget to enclose each in a **try:except** so that you can continue to the next. (And log what happens at each step so you can figure out after the fact what you got and why.)

- Each archive is a living archive. Things change as a function of time, so what you did yesterday might not come out identically today.

- The VO is a work in progress (almost by definition), but it is a powerful one!

Go to NAVO tutorials

- We have developed the following notebooks

  https://github.com/NASA-NAVO/navo-workshop

  - Download them to run locally and adapt, or
  - view them rendered on GitHub at

    https://nasa-navo.github.io/navo-workshop/
  - run them in MyBinder through link at bottom of GitHub repo page.

- Contents:
  - QuickReference.ipynb — example of each type of search;
  - Use Case I — inspecting a candidate list;
  - Use Case II — preparing a proposal;
  - a set of more detailed cheat-sheets for each type of search;
  - KNOWN_ISSUES.md — list of known oddities/errors/workarounds.

Examples from CDS tutorials:

https://github.com/cds-astro/tutorials/blob/master/Notebooks/

SSA

```
mast_ssa_service = pyvo.dal.SSAService('https://archive.stsci.edu/ssap/search2.php?id=HST&')
diameter = u.Quantity(2 * 40.0, unit="arcmin")
position = SkyCoord.from_name('A1656')
mast_hst_results = mast_ssa_service.search(pos=position, diameter=diameter)
mast_hst_results
```

```
interesting_obs = mast_hst_results[-1]
obs_url = interesting_obs.getdataurl()
```

```
spectrum_fits = fits.open(obs_url)
spectrum_fits.info()
```

TAP

```
tap_vizier = pyvo.dal.TAPService('http://tapvizier.u-strasbg.fr/TAPVizieR/tap')
mass_psc_set = tap_vizier.search("SELECT  *  FROM tables " +
                                 "WHERE description LIKE '%2MASS%Cutri%'").to_table()
mass_psc_set['table_name', 'description']
```

Another brief one: http://exoplanet.eu/API/ (TAP queries)

M Demleitner's tutorial:

advanced user, some differences, some things to learn

Registry

```
for svc_rec in pyvo.registry.search(datamodel="obscore"):
  svc = pyvo.dal.TAPService(svc_rec.access_url)
  result = svc.run_sync("SELECT DISTINCT obs_collection"
    " FROM ivoa.obscore")
  print("\n>>>>{}\n{}\n".format(
```

Advantage to defining a TAPService instead of using the RegistryResource?

UCDs

```
UCD_TO_WL = {
  "phot.mag;em.opt.u": 3.5e-7,
  "phot.mag;em.opt.b": 4.5e-7,
  "phot.mag;em.opt.v": 5.5e-7,
  "phot.mag;em.opt.r": 6.75e-7, ...

  for row in rows:
    for index, col in enumerate(row):
      ucd = row.columns[index].meta.get("ucd", "").lower())
      if ucd.startswith("phot.mag"):
        if ucd in UCD_TO_WL:
          phots.append((UCD_TO_WL[ucd], col))
```

TAP

Run queries via TAP:

```
access_url = "http://dc.g-vo.org/tap"

service = pyvo.dal.TAPService(access_url)
result = service.run_sync(
  """SELECT raj2000, dej2000, jmag, hmag, kmag
        FROM twomass.data
        WHERE jmag<3""")
for row in result:
```

DataLink and semantics

Each link has a URL, a description, and machine-readable semantics[15]. E.g

```
for dl in matches.iter_datalinks():
    prev_url = dl.bysemantics("#preview").next()["access_url"]
    im = Image.open(io.BytesIO(requests.get(prev_url).content))
    ...
```

Everything SAMP, SODA; managing queries; custom scripts…

Registry search

Simple example: Find Simple Cone Search (conesearch) services related to SWIFT.

```
services = vo.regsearch(servicetype='conesearch', keywords=['swift'])
```

Argument	Description	Examples
servicetype	Type of service	`conesearch` or `scs` for **Simple Cone Search** `image` or `sia` for **Simple Image Access** `spectrum` or `ssa` for **Simple Spectral Access** `table` or `tap` for **Table Access Protocol**
keyword	List of one or more keyword(s) to match service's metadata. Both ORs and ANDs may be specified. • (OR) A list of keywords match a service if **any** of the keywords match the service. • (AND) If a keyword contains multiple space-delimited words, **all** the words must match the metadata.	`['galex', 'swift']` matches 'galex' or 'swift' `['hst survey']` matches services mentioning both 'hst' and 'survey'
waveband	Resulting services have data in the specified waveband(s)	'radio', 'millimeter', 'infrared', 'optical', 'uv', 'euv', 'x-ray' 'gamma-ray'

Searching?
Other tricks?

Depends on metadata

```
stsci_services = [s for s in services if 'stsci.edu' in s.ivoid]
for s in stsci_services:
    print (f'(STScI): {s.short_name} - {s.res_title}')
```

```
(STScI): MAST CS - MAST ConeSearch
```

But thinking about a user browsing this way may make you re-consider how you describe your services etc.

Should region size specifications be standardized?

```
coords = coord.SkyCoord.from_name("m51")

im_table = uvot_services[0].search(pos=coords,size=0.2,format='image/jpeg')
```

PyVO SIA uses ‘size’

```
# Search for a spectrum search service that has x-ray data.
services = vo.regsearch(servicetype='spectrum', waveband='x-ray')

# Assuming there are services and the first one is OK...
results = services[0].search(pos=SkyCoord.from_name("Delta Ori"),
                             diameter=Angle(10 * u.arcmin))
```

PyVO SSA uses ‘diameter’

```
: m51_pos = SkyCoord.from_name("m51")
services = vo.regsearch(servicetype='conesearch', keywords='usno-b')
results = services[0].search(pos=m51_pos, radius=0.1)
```

PyVO SCS uses ‘radius’

query_region

Query a region around a coordinate.

Astroquery API gives two more options

One of these keywords *must* be specified (no default is assumed):

```
radius - an astropy Quantity object, or a string that can be parsed into one.
         e.g., '1 degree' or 1*u.degree.
         If radius is specified, the shape is assumed to be a circle
width - a Quantity.  Specifies the edge length of a square box
height - a Quantity.  Specifies the height of a rectangular box.  Must be passed with wi
```

Selecting results that we want

```
services=vo.regsearch(servicetype='image',keywords=['heasarc swift'])
m83_pos = SkyCoord('13h37m00.950s -29d51m55.51s')
sia_results=services[0].search(pos=m83_pos, size=.2,format='image/fits')
sia_results.to_table()
```

Table length=6

Survey	Ra	Dec	Dim	Size	Scale	Format	PixFlags	
object	float64	float64	int32	object	object	object	object	
swiftuvotvint	204.25395833333332	-29.865419444444445	2	[300 300]	[-0.0006666666666666668 0.0006666666666666668]	image/fits	F	29.86
swiftuvotbint	204.25395833333332	-29.865419444444445	2	[300 300]	[-0.0006666666666666668 0.0006666666666666668]	image/fits	F	29.86
swiftuvotuint	204.25395833333332	-29.865419444444445	2	[300 300]	[-0.0006666666666666668 0.0006666666666666668]	image/fits	F	29.86
swiftuvotuvw1int	204.25395833333332	-29.865419444444445	2	[300 300]	[-0.0006666666666666668 0.0006666666666666668]	image/fits	F	29.86541
swiftuvotuvw2int	204.25395833333332	-29.865419444444445	2	[300	[-0.0006666666666666668	image/fits	F	29.86541
swiftuvotuvm2int	204.25395833333332	-29.8654194444					F	29.86541

This hideous thing gets the row of an Astropy Table but not a callable SIAResult object

```
import numpy as np
sia_results.to_table()[ np.isin( results.to_table()['Survey'], b'swiftuvotvint')]
```

Table length=1

Survey	Ra	Dec	Dim	Size	Scale	Format	PixFlags	
object	float64	float64	int32	object	object	object	object	
swiftuvotvint	204.25395833333332	-29.865419444444445	2	[300 300]	[-0.0006666666666666668 0.0006666666666666668]	image/fits	F	29.865419444

Table searches

```
services = vo.regsearch(servicetype='tap', keywords=['heasarc'])
print(f'{len(services)} service(s) found.')
# We found only one service.  Print some info about the service and its tables.
print(f'{services[0].describe()}')
tables = services[0].service.tables # Queries for details of the service's tables
print(f'{len(tables)} tables:')
for t in tables:
    print(f'{t.name:30s} - {t.description}')  # A more succinct option than t.describe()
```

TAPService.tables exposed in RegistryResource?

```
1 service(s) found.
Table Access Protocol Service
HEASARC Xamin Catalog Interface
Short Name: HEASARC
IVOA Identifier: ivo://nasa.heasarc/services/xamin
Base URL: https://heasarc.gsfc.nasa.gov/xamin/vo/tap

The HEASARC is NASA domain archive for high-energy and microwave astronomy.
The Xamin interface provides access to over 600 observation and object tables.
This includes observation tables for more than 30 missions and observatories
and hundreds of derived object tables. Non-high energy tables are included to
make it easier for users to compare information.

Subjects: HEASARC
Waveband Coverage:
None
993 tables:
```

snip

```
"first"                        - Faint Images of the Radio Sky at Twenty cm (FIRST)
a1                             - HEAO 1 A1 X-Ray Source Catalog
a1point                        - HEAO 1 A1 Lightcurves
a2lcpoint                      - HEAO 1 A2 Pointed Lightcurves
a2lcscan                       - HEAO 1 A2 Scanned Lightcurves
a2led                          - HEAO 1 A2 LED Catalog
```

Tables metadata

Maybe a describe() method (like each service and table has)?

```
for c in tables['zcat'].columns:
    print(f'{c.name:30s} - {c.description}')
```

```
class                          - Browse Object Classification
dec                            - Declination
morph_type                     - Morphological (T) Type
bt_mag                         - B_T Magnitude
notes                          - Diameter (arcmin) or Spectrum ID
bmag                           - B Magnitude
radial_velocity_error          - Radial Velocity Error
"__y_ra_dec"                   - System unit vector column
radial_velocity                - Heliocentric Radial Velocity (km/s)
comments                       - Comments
name                           - Catalog Designation
"__x_ra_dec"                   - System unit vector column
ref_radial_velocity            - Reference Code for Radial Velocity
bar_type                       - Bar Type
```

Table example queries
(PR under review)

```
for example in heasarc_tap_services[0].service.examples:
    print(example['QUERY'])
    result=example.execute()
    #  Stop at one
    break
result.to_table()
```

TAPService.examples exposed in RegistryResource

```
            SELECT * FROM rosmaster                          WHERE exposure > 10000 and
1=CONTAINS(POINT('ICRS', ra, dec),CIRCLE('ICRS', 50, -85, 1))
```

Table length=2

__row	seq_id	ra	dec	lii	bii	instrument	filter	site	exposure	requested_exposure	fits_type
		deg	deg	deg	deg				s	s	
object	object	float64	float64	float64	float64	object	object	object	int32	int32	object
1	RH202299N00	49.3200	-85.5400	299.8517	-30.6815	HRI	N	MPE	36146	70000	RDF 4_2
2	RH202299A01	49.3200	-85.5400	299.8517	-30.6815	HRI	N	MPE	43683	70000	RDF 3_6

Tables queries

```
results = heasarc_tap_services[0].service.run_async(query)
#results = heasarc_tap_services[0].search(query)
results.to_table()
```

Teach sync or async?

Table length=1120

ra	dec	radial_velocity	radial_velocity_error	bmag	morph_type
float64	**float64**	**int32**	**int16**	**float32**	**int16**
10.6847	41.2688	-297	1	4.3	3
189.2076	13.1627	-223	18	10.58	2

UCDs: how to motivate and explain their use?

(work in progress, can be very useful)

```
#  Look for all TAP services with x-ray and optical data
collection={}
for s in vo.regsearch(servicetype='tap',keywords=['x-ray','optical']):
    print(f"Looking at service from {s.ivoid}")
    tables=s.service.tables
    #  Find all the tables that have an RA,DEC and a start and end time
    for t in tables:
        names={}
        for ucd in ['pos.eq.ra','pos.eq.dec','time.start','time.end']:
            cols=[c.name for c in t.columns if c.ucd and ucd in c.ucd]
            if len(cols) > 0:
                names[ucd]=cols[0]  # use the first that matches
        if len(names.keys()) == 4:
            query="select top 10 {}, {}, {}, {} from {}".format(
                names['pos.eq.ra'],
                names['pos.eq.dec'],
                names['time.start'],
                names['time.end'],
                t.name)
            print(f"Table {t.name} has the right columns.  Executing query:\n{query}")
            results=s.search(query)
            print("Found {} results\n".format(len(results)))
            #  Careful.  We're assuming the table names are unique
            collection[t.name]=results
```

```
Looking at service from ivo://cxc.harvard.edu/cda
Looking at service from ivo://cxc.harvard.edu/csc
Looking at service from ivo://cxc.harvard.edu/cscr1
Table csc1.obi_source has the right columns.  Executing query:
select top 10 ra_aper, dec_aper, gti_mjd_obs, gti_stop from csc1.obi_source
Found 10 results

Looking at service from ivo://cxc.harvard.edu/cscr2
Looking at service from ivo://esavo/xmm/tap
Looking at service from ivo://eso.org/tap_cat
Table XQ_100_summary_fits_V1 has the right columns.  Executing query:
```

Using UCDs?
PyVO methods for this in
pyvo.io.vosi.vodataservice.Table ?
(like fieldname_with_ucd is for *Results)

DataLink

(work in progress, much more complicated)

```
#  Get the HEASARC TAP resource from the Registry
services=vo.regsearch(servicetype='tap',keywords=['heasarc'])
#  Construct a query to get objects near our source:
query="""SELECT * FROM chanmaster WHERE
        1=CONTAINS(POINT('ICRS', ra, dec),
        CIRCLE('ICRS', {}, {}, 1))""".format(pos.ra.deg,pos.dec.deg)
results = services[0].search(query)
links=results[0].getdatalink()
for links in results.iter_datalinks():
    #  Then look at all the linked objects for each TAP result
    #   and find the type we want
    for link in links:
        print(link.description)
        if "ADS" in link.description:
            # Let's just look at one of them
            l=link
    break
display(HTML(l.getdataset().data.decode()))
```

Error trapping? Ugly exception if no DataLink defined

```
ASCA: Nearby (1 deg) ASCA Observations
XMM: Nearby (1 deg) XMM-Newton Observations
Chandra Observation
Chandra Observation
Chandra Proposal Abstract
RXTE: Nearby (1 deg) RXTE Observations
HEASARC page for pointing to ADS
ROSAT: Nearby (1 deg) ROSAT Observations
```

Multiple (9) bibcodes found.

- 2010A&A...516L...8F
- 2013ApJS..209...26F
- 2013ApJS..209...27K
- 2013MsT.....19B

DataLink
(work in progress)

```
def linkwalker( result, level ):
    print("LEVEL {}".format(level))
    try:
        result2=result.getdatalink()
        print(result2.to_table()['description','content_type'])
    except Exception as e:
        print("Exception {}".format(e))
        return
    for link in [l for l in result2 if "datalink" in l.content_type]:
        linkwalker(link, level+1)
    return
```

```
linkwalker(results[0], 0)
```

```
LEVEL 0
                  description                       ...
------------------------------------------- ...
XMM: Nearby (1 deg) XMM-Newton Observations ...
                  Chandra Proposal Abstract ...
     RXTE: Nearby (1 deg) RXTE Observations ...
   ROSAT: Nearby (1 deg) ROSAT Observations ...
          HEASARC page for pointing to ADS ...
                       Chandra Observation ...
                       Chandra Observation ...
     ASCA: Nearby (1 deg) ASCA Observations ...
LEVEL 1
     description                      content_type
--------------------- ----------------------------------------
   Chandra Observation                                text/html
  FITS and JPEG Images application/x-votable+xml;content=datalink
Orbit and Aspect Files application/x-votable+xml;content=datalink
   Miscellaneous Files application/x-votable+xml;content=datalink
          Events List                          application/fits
    Field of View File                         application/fits
LEVEL 2
description    content_type
----------- ---------------
Center Image application/fits
```

Recursive DataLink walker in PyVO? To find certain types of data, e.g. with semantics?

Other issues

- See https://github.com/nasa-navo/navo-workshop/blob/master/KNOWN_ISSUES.md
- (Partly out of date already.)