Not a presentation but a discussion

coordinated by Tess Jaffe

End users searching the Registry with PyVO

What use cases are there, what does the Registry need to do differently, what does PyVO need to do differently?

Registry session Nov 11, 2023 IVOA Interop, Tucson

Use cases: list of what we have

(gathered from Registry working group via Google doc

- I read in a paper that a dataset I am interested in using has DOI XXXX. How do I search the registry for that?
- How do I find the catalogs/services with redshift information?
- User looking for data, not sure what kind is available
- Somebody is searching for timeseries of a given survey accessible TAP
- Looking for UV images, with no knowledge of SIA vs SIAv2
- Looking for catalog data with no knowledge of cone vs TAP
- extend the text search with fuzzy search
- It’s hard to build a search for hips
- curation / exploration of terms in registry / services (BCecconi — semantics)
- Wider use of constraints (“Covers M1 in XRay”)
- Duplication issues

See also Markus’ slides for Apps II

I read in a paper that a dataset I am interested in using has DOI XXXX. How do I search the registry for that?

- Renaud: Apparently we need a new API constraint type for searching in the alt_identifier RegTAP table
-
-

Renaud: How do I find the catalogs/services with redshift information?

[Renaud Savalle] Here is a Proof Of Concept, using the CDS UCD Finder, that relies on a description rather than a UCD:

```
# POC for incorporating CDS UCD Finder with registry
search
# Author: Renaud Savalle (thanks to Gilles Landais for CDS
UCD Finder service)

import requests
import urllib
from pyvo import registry
from astropy.coordinates import SkyCoord

what = 'infrared magnitude'
where = 'm42'

print(f'Search for {what} around {where}')

def find_ucd(desc):
    """
    Use CDS UCD Finder to get ucd corresponding to a
description
    """
    resp =
requests.get(f'https://cds.unistra.fr/ucd-finder/beta/assi
gn?d={urllib.parse.quote_plus(desc)}')
    ucd = resp.content.decode('utf-8')
    return ucd
```

```
# Resolve UCD
ucd = find_ucd(what)
# Resolve object
coord = SkyCoord.from_name(where)

print(f'Looking for resources with ucd {ucd}
around {coord}')

resources = registry.search(
    registry.UCD(ucd.lower()),
    registry.Spatial(coord))

print(resources)
```

[Gilles Landais]Context: Service discovery: add a search by contents (I would like data having magnitude column, radial velocity column)

Same as previous basically?

TJ: User looking for data, not sure what kind is available or how to query it:

```
>>> rscs = registry.search(keywords="quasar",
                           ucd="src.redshift")
>>> svc = rscs["III/175"].get_service("conesearch")
>>> svc.search((126, -20), 5)
```

- But this requires users to know IVOA jargon “conesearch”. But what about when they do not? Specifically, how did the user know what parameters to give to the `search()` function for a given service?
- The `svc` object is something the user needs to be told how to use. Currently, this appears to be done with `svc.create_query?` for some types (not sure about all, but it should be there for all). Is it sufficient to teach users to look at that? More user friendly to add parameter help to `svc.describe()`? How about something that does:

```
>>> for interface in rscs["III/175"].interfaces:
        interface.describe()
```

And would see something like:

```
Interface standard:  Cone ('ivo://ivoa.net/std/conesearch')
Signature: search(pos=None, radius=None, verbosity=None, **keywords)
```

Maybe with a verbose option to print out the more detailed help currently found in the `create_query` documentation.

[Gilles Landais] Context: user-friendly output for access_method for non specialist users (same basically as previous)

{'sia#aux', 'tap#aux'} -> {Image Access, Table Access (SQL) }

[Hendrik Heinl]: Somebody is searching for timeseries of a given survey accessible TAP [1]. Timeseries as such are not reliably findable with single registry queries [2]. If we introduce something similar like the obscore

Renaud: We propose to Bring ObsCore dataproduct_types into the VODataService Registry records via update of VODataService...

TJ: Looking for UV images, with no knowledge of SIA vs SIAv2:

```
>>> uv_services=vo.regsearch( servicetype='image',keywords='galex',
waveband='uv')
```

- As above, how do they know if the services take the SIAv2 parameters?

```
>>> uv_services[0].interfaces[0].describe()
Interface standard:  Image ('ivo://ivoa.net/std/sia2')
Signature:  search(
     pos=None,
     size=None,
     format=None,
     intersect=None,
     verbosity=None,
     **keywords,
)
```

- But then suppose in the `uv_services` list of image services, some are SIA and some SIAv2. They check the usage of one of them, it shows SIAv2 options, and the user then sends the same query to all of them. What should PyVO do when it comes to the SIA(v1) services?

Looking for UV images, with no knowledge of SIA vs SIAv2: Markus' idea from PyVO PR (see also Apps II)

The Image Discovery API

What I want to offer:

```
images, log = discover.images_globally(
        space=(132, 14, 0.1),
        time=time.Time(58794.9, format="mjd"),
        spectrum=600*u.eV,
        inclusive=False)
```

images would be a list of obscore-like metadata , and *log* would be information on which services yielded how much (or how they failed).

If True, *inclusive* would make the function ask services without coverage information and return SIAP1 images that have no usable information on time and spectrum.

Catalogs: cone vs TAP

- User knows IVOA jargon and they want to find a simple cone service for the redshift catalog called Zcat:

```
>>> services = pyvo.regsearch(servicetype='conesearch', keywords=['zcat'])
>>> services = registry.search(registry.Servicetype('conesearch'),
registry.Freetext('zcat'))
```

- … a more powerful queryable table service, CfA redshift catalog:

```
>>> tap_services = pyvo.regsearch(servicetype='table', keywords=['cfa redshift'],
includeaux=True)
```

Is it possible to extend the text search with fuzzy search (eg: using Lemmatization for instance)

```
>>> resources2 = registry.search("asteroids")
>>> resources1 = registry.search("asteroid")
>>> len(resources2)
 350
>>> len(resources1)
 517
```

Search api ergonomy: ADS like search:
Eg: https://ui.adsabs.harvard.edu/
author:"^Turtle, Elizabeth"
abs:"*galact*"

(Manon) The hips are classified as ServiceCatalog in the registry, it’s hard to build a search for hips

Context: curation / exploration of terms in registry / services (BCecconi — semantics)

See Google doc
https://docs.google.com/document/d/1ayBvOMC7cHb02ERZyJGq-MYSn-ORHtvCo4Y1jhYssYo/edit

[Renaud Savalle] Context: Here is an example to combine constraints (our example from ADASS poster P106)

Example Python program from ADASS XXXIII Poster P106:
"Discover your astronomical data from python – simply!"
https://adass2023.lpl.arizona.edu/events/poster-p106

```
# Authors: Renaud Savalle, Markus Demleitner, Hendrik Heinl
# Description:
# In this example, we use registry.search() to look for VO resources that publish
# infrared magnitudes (the UCD constraint) near the Flaming Star nebula in
# Auriga (the Spatial constraint) and mention emission line stars in their
# human-readable metadata. Then, connecting to Aladin via SAMP, we loop over
# the resources we have discovered. Those that have an SCS endpoint we query
# and send whatever we get back to Aladin.

from pyvo import registry
from pyvo import samp
from astropy.coordinates import SkyCoord


with samp.connection() as conn:
    for rsc in resources: # loop over the resources found
        if "conesearch" in rsc.access_modes(): # invoke the conesearch services
            objs = rsc.get_service("conesearch").search(pos=(79, 34), radius=0.5)
            if objs: # if sources are found, send them to Aladin
                samp.send_table_to(
                    conn,
                    objs.to_table(),
                    client_name="Aladin",
                    name=rsc.short_name)
```

Markus: Use your constraints (“Covers M1 in XRay”)

But:

```
select count(*) from rr.stc_spatial natural join rr.capability
where standard_id like 'ivo://ivoa.net/std/sia%'
```

At the moment, only 83 SIAP1/2 services declare their spatial coverage (30 for spectral, 43 for temporal; SSAP has 35 spatial coverages). Please improve this!

Not sure PyVO can help, so this is a publication issue?

Similarly, second request to Registry: Please change the Obscore registration pattern to how it’s done for EPN-TAP (i.e., as a table with its own metadata).

Markus: duplication

- From ObsCore and SxA
  - Proposal (re-using auxiliary capabilities): SIAP(2) and SSAP records should include isServedBy relationships to Obscore, TAP, and sitewide SIAP2 services.

- Likewise, How many resources have both SIAP1 and SIAP2 interfaces?
  - ?

Other?