Use cases

Please list examples of data and service discovery by the end user interactively in Python. If applicable, show what the current way to do that in PyVO looks like. Add comments about issues, whatever you think that we'll need to discuss during the session.

See also Markus' data discovery demo notebook and his very similar slides for the Apps II session.

TJ consolidating the below into slides for the Registry session on Nov 11, 2023 (permissions updated Sat. 9am).

- 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)
```

- Same IVOA-aware user knowing SIA vs SIA2, as above. But:
- User looking for data, not sure what kind is available. Markus's reg example:

```
>>> 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.

- 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 x-ray spectra, same as image search, without the v2 issue
- 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?

[Renaud Savalle] Here is one way if the user knows UCDs:

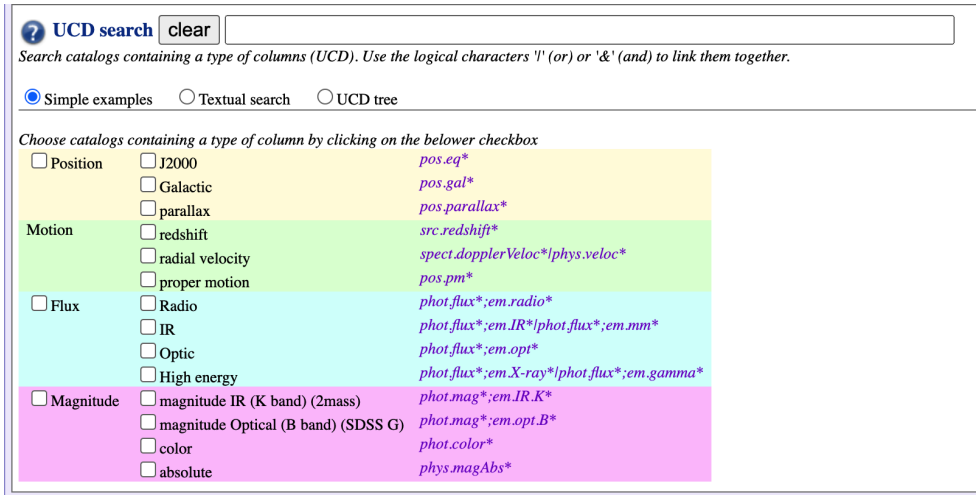
```
# (RS) 2023-10-30: 3967 resources found with:
from pyvo import registry
resources = registry.search(
    registry.UCD("src.redshift%"))
for r in resources:
    print(r.ivoid,r.access_modes())
```

[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/assign?d={urllib.parse.qu
ote_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)
```

NB: See also UCD Search interface in VizieR which mentions some commonly used UCDs in its “Simple examples” section:



Issues:

1. “Servicetype” and “includeaux”: users shouldn’t need to know VO jargon to search for “images” or “spectra”. Need a slightly higher level, user-friendly search method.

2. Need a user friendly routine to show the user how to use a service. Some of this exists but could be better.
3. For SIA, they need to know if they can use SIAv2 parameters or not. How?
    a. SIA v1 being deprecated. So have PyVO just error/warn if parameters have been specified that the service doesn't understand?
4. (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)

Retrieving the list of distinct pairs of "/instrument" and /"facility" from the VOResource records, keeping track of the ivoid's of the resources:

```
>>> from pyvo.dal import TAPService
>>> from collections import namedtuple
>>> rr_server = "http://voparis-rr.obspm.fr/tap"
>>> rr_tap = TAPService(rr_server)
>>> source_type = 'registry'
>>> query = """select distinct t1.ivoid as ivoid, t1.detail_value as facility, t2.detail_value as instrument
from rr.res_detail as t1 inner join rr.res_detail as t2 on t1.ivoid = t2.ivoid
where t1.detail_xpath='/facility' and t2.detail_xpath='/instrument'"""
>>> results = rr_tap.search(query)
>>> ObsSystem = namedtuple('ObsSystem', ['facility', 'instrument'])
>>> Resource = namedtuple('Resource', ['source_type', 'ivoid', 'table_name'])
>>> facilities_ivo = dict()  # IVOA registry outputs
>>> for row in results:
    name = ObsSystem(row['facility'], row['instrument'])
    source = Resource(source_type, row['ivoid'], None)
    if name in facilities_ivo.keys():
        facilities_ivo[name].add(source)
    else:
        facilities_ivo[name] = {source}
```

From this, we can obtain all VOResource nodes claiming to contain HST/WFC3 observations, as follows=

```
>>> facilities_ivo[ObsSystem(facility='HST', instrument='WFC3')]

{Resource(source_type='registry', ivoid='ivo://archive.stsci.edu/borg', table_name=None),
 Resource(source_type='registry', ivoid='ivo://archive.stsci.edu/catalogs/hlsp_30dor', table_name=None),
 Resource(source_type='registry', ivoid='ivo://mast.stsci/andromeda', table_name=None),
 Resource(source_type='registry', ivoid='ivo://mast.stsci/clash', table_name=None),
 Resource(source_type='registry', ivoid='ivo://mast.stsci/hippies', table_name=None),
 Resource(source_type='registry', ivoid='ivo://mast.stsci/hudf09', table_name=None)}
```

Same query with ObsTAP services (scanning "instrument_name" and "facility_name" columns), using the registry to find them:

```
>>> query = """SELECT table_name, access_url, ivoid FROM ( SELECT ivoid, table_name FROM rr.res_table WHERE
 table_name = 'ivoa.obscore') as ivotables
NATURAL JOIN rr.capability
NATURAL JOIN rr.interface WHERE
standard_id = 'ivo://ivoa.net/std/tap' AND intf_role='std'"""
>>> obscore_results = rr_tap.search(query)
```

(since there are several server not answering or throwing errors, we need to manage exceptions)

```
>>> from pyvo.dal import DALServiceError, DALQueryError, DALFormatError
>>> from requests.exceptions import ConnectionError
>>> source_type = 'obscore'
>>> for table in obscore_results:
    url = table['access_url']
    ivoid = table['ivoid']
    print(url)
    try:
        tap_service = TAPService(url)
        query = "SELECT distinct facility_name as facility, instrument_name as instrument FROM ivoa.obscore"
        results = tap_service.run_async(query)

        for row in results:
            name = ObsSystem(row['facility'], row['instrument'])
            source = Resource(source_type, table['ivoid'], table['table_name'])
            if name in facilities_obs.keys():
                facilities_obs[name].add(source)
            else:
                facilities_obs[name] = {source}
        except DALServiceError as e:
            print(e)
            status[ivoid] = str(e)
        except DALQueryError as e:
            print(e)
            status[ivoid] = str(e)
        except DALFormatError as e:
            print(e)
            status[ivoid] = str(e)
        except ConnectionError as e:
            print(e)
            status[ivoid] = str(e)
```

The list of servers not responding correctly is:

Finally, same query to explore EPN-TAP services (obtained from the registry), scanning the “instrument_name” and “instrument_host_name” columns:

```
>>> query = """SELECT table_name, access_url, ivoid FROM ( SELECT ivoid, table_name, table_utype FROM rr.res_table
WHERE
 table_utype like 'ivo://vopdc.obspm/std/epncore%' OR table_utype LIKE 'ivo://ivoa.net/std/epntap#table-2.%' OR
table_utype='ivo.//vopdc.obspm/std/epncore#schema-2.0') as epntables
NATURAL JOIN rr.capability
NATURAL JOIN rr.interface WHERE
standard_id = 'ivo://ivoa.net/std/tap' AND intf_role='std'"""
epncore_results = rr_tap.search(query)

source_type = 'epncore'


epncore_tables = {}
for row in epncore_results:
    access_url = row['access_url']
    if access_url in epncore_tables.keys():
        epncore_tables[access_url].append((row['ivoid'], row['table_name']))
    else:
        epncore_tables[access_url] = [(row['ivoid'], row['table_name'])]
```

NB1: the correct table_utype is “ivo://ivoa.net/std/epntap#table-2.0”, but many resources have an old table_utype, predating the IVOA recommendation adoption; some have erroneous strings; and some have no table_utype (and thus can‘t be found automatically).
NB2: since there can be several epn_core tables in a TAP server, we group the tables by access_url (TAP endpoint)

```
>>> # adding major tables which are still incorrectly registered:
>>> epncore_tables['https://archives.esac.esa.int/psa-tap/tap'] = [('ivo://esavo/psa/epntap', 'epn_core')]
>>> epncore_tables['http://tapvizier.cds.unistra.fr/TAPVizieR/tap'] = [('ivo://cds.vizier/b/planets',
'\"B/planets/epn_core\"')]
```

Then we can run the queries to the TAP servers:

```
>>> for url, tables in epncore_tables.items():
    print(url)
    try:
        tap_service = TAPService(url)

        for ivoid, table in tables:
            print(table)
            if table.startswith("sshade"):
                print("skip...")
                continue

            query = f"select distinct instrument_host_name, instrument_name from {table}"
            results = tap_service.search(query)

            for row in results:
                name = System(row['instrument_host_name'], row['instrument_name'])
                source = Source(source_type, ivoid, table)
                if name in facilities_epn.keys():
                    facilities_epn[name].add(source)
                else:
                    facilities_epn[name] = {source}

    except DALServiceError as e:
        print(e)
    except DALQueryError as e:
        print(e)
    except DALFormatError as e:
        print(e)
```

[Gilles Landais] **Context**: Free text search

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*"

[Gilles Landais] **Context**: user-friendly output for access_method for non specialist users.
{'sia#aux', 'tap#aux'} -> {Image Access, Table Access (SQL) }

[Gilles Landais]**Context:** Service discovery: add a search by contents (I would like data having magnitude column, radial velocity column)
Extend the search (based on keyword) using column content (UCD?)

[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)
```

[Hendrik Heinl]: A use case would be that 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
`dataproduct type='timeseries'`
That would be great, because in a second step pyvo.registry could be extended to do something like this:

```
resources = registry.search(
      registry.Freetext("Tess"),
      registry.dataproducttype("timeseries"),
      registry.Spatial(SkyCoord("79d 34d")))
```

Or

```
resources = registry.search(
      registry.UCD(phot.mag;em.opt.rb),
      registry.dataproducttype("timeseries"),
      registry.Spatial(SkyCoord("79d 34d")))
```

[1] https://wiki.ivoa.net/internal/IVOA/InterOpMay2023Registry/hendrik_heinl.pdf
[2] https://wiki.ivoa.net/internal/IVOA/InterOpMay2023Registry/htbns.ipynb