Astropy, PyVO and the Radio realm

Hendrik Heinl, Dave Morris

October 24, 2021

Abstract

In the past years, VO standards were widely accepted and spread and many astronomers get more and more familiar with standards and protocols like Cone Search, SAMP and TAP/ADQL. Being skilled in these in combination with Astropy and PyVO opens almost unlimited opportunities to find, to access and to combine different datasets for further analysis.
In this tutorial the participants will learn how to use PyVO and ObsTAP to find services, how to explore the data on these services, they will use Datalink and SODA to perform cutouts on large images, and will use PyVO, Matplotlib and Astropy to write a SAMP handler to combine functionality of TOPCAT and Aladin with PyVO.
This tutorial is not an introduction in any of the used standards, protocol tools or Python. To get more insight into this, have a look at the References section of this document.

Software: TOPCAT V4.8, Aladin V11.0, PyVO

1 Starting out

We will use the VO tools TOPCAT and Aladin. These client software can be found here:

Aladin: `http://aladin.u-strasbg.fr/`

TOPCAT: `http://www.star.bris.ac.uk/~mbt/topcat/`

We will also use PyVO, Matplotlib and Astropy. These are best installed from your distribution packages.

A comprehensive collection of the examples in this tutorial as well as installation scripts you will find on github. Checkout the taenaris repository.

2 The use case

This tutorial will show how we will show that base on a list of objects we find photometry in SDSS, plot SEDs to find objects of interest and eventually access radio image data from LOFAR. The premise here is that we don't simply use uploads and x-matches, but instead will write supporting software in Python to do a lot of the service handling for us. A key standard for this will be the Simple Application Messaging Protocol. The exercises of this tutorial will be in Python. To follow them, we advice to checkout the git repository. For execises demanding coding, we provide solutions (named in the exercise). If you struggle, you may want to have a look into the "Helpful Links" section below.

3 Finding data: the VO registry

For the first step of finding data we start Topcat. Then we load the table objects.vot from the folde pysrc to Topcat. The content is just a list of 19 positions to start with. For our further steps we will make use of colours, so we are going to search for a service providing us with data from SDSS. We will do so by using the VO-Registry from Topcat.

VO Registry

The Registry is the central point for searches for services in the VO. VO services identify themselves to the Registry, providing metadata about the data they serve and the service protocols they support. Thus, the Registry is the entry point for data discovery in the VO.

▷ **1** *Finding services for SDSS colours* – In Topcat click on VO → Table Access Protocol. In the window at Keywords type SDSS12 and wait for the results. Double click on the Vizier service. The TAP Query window will open and it may take a few seconds for the metadata to load. On the left you see a tree of the catalogues and tables on the service, on the right is the metadata browser on these tables. Mark any table and see what the description of the table and the columns tell you about the data in the table. This meta data still is part of the TAP standard and is very helpful for data discovery and of course for the data access using ADQL.

In the upper left of the meta data at Find: enter SDSS12 to find tables containing this survey. In this example we just want to collect the identifier of an object, the position in ra and dec and the colours in the u, g, r, and i band. We will make use of the TAP feature TAP UPLOAD, to join our local table with results from the remote service. In the lower window at ADQL Text click on Examples → Upload → Upload Join. You see an example ADQL query appearing in the field. Modify it so it looks as following:

```
SELECT
   SDSS12, mine.*, sdss.gmag, sdss.imag, sdss.rmag, sdss.umag
   FROM "V/147/sdss12" AS sdss
   JOIN TAP_UPLOAD.t1 AS mine
   ON 1=CONTAINS(POINT('ICRS', sdss.RA_ICRS, sdss.DE_ICRS),
                 CIRCLE('ICRS', mine.ra, mine.dec, 2./3600.))
```

In case anythong goes wrong, you can use our fall back and load the file objects_colours.vot from the repository into Topcat to continue the tutorial. We will need the colours in a later stage.

Exercise: Have a look into the metadata browser. You can find information about the service, the catalogue, the table and the columns. This rich metadata enables a lot of the functionality of the VO Services.

Exercise: You can play around with the VO registry. Search for TAP services that provide you with data you are interested in. Try gamma ray data, AGN, or gravitational waves and see what the VO has to offer.

▷ **2** *Example1: Finding VO services for radio images* – The good part of VO standards is: you can develop your own clients to use them. Luckily some people did this work already: the PyVO package provides us with an API to the VO. Here, we start with the module pyvo.registry.search. Look into folder example1 and open the script reg_search.py. You see the actual code is pretty simply and self explanatory. Run the script and look at the list of results.

Exercise: Modify the script in a way that it searches for TAP-services providing gaia data (Solution: example1/exercise_gaia.py)

Exercise: Modify the script in a way that it searches for SIAP-services providing HST data (Solution: example1/exercise_HST.py)

Exercise: Modify the script so that you search for ObsTAP services providing radio data (Solution: example1/exercise_radio.py)

Have a brief look at the few lines in example1/exercise_radio.py. Note, that two lines are necessary to supress warnings – services and clients often are a bit off, especially in regard of recent standards. They still work, but warnings will be raised. We don't want to pretend that everything in the VO is perfect, but within a tutorial, warnings might be confusing. Of course when you are in the process of development, warnings are very useful, so be smart with this tool of ignorance.

Remember to take a look at the helpful link section. There you will find hints and documentation. Use a python interpreter like ipython to recapitulate the registry query and to play with the resulting object.

Exercise: Try to figure out the properties of the services. In particular we are interested in a reasonable name and a url which we can use to access the services in a later tutorial step. (Solution: example1/exercise_svc_info.py)

4 Finding data: Obscore

The first two of above exercises are relatively easy to solve, but the last one is a bit tricky, because though we talk about ObsTAP-services, these actually are regular tap services providing a special data model: Obscore.

ObsCore

ObsCore Is a common data model for describing astronomical observations. It defines the core components of metadata needed to discover what observational data is available from a service. Every service that advertises itself as an ObsTAP service must provide this standard view of metadata lsiting the observational data available from that service.. This means that a user can build an ADQL query based on the ObsCore data table and apply the same query to all the ObsTAP services.

▷ **3** *Example2: Making an obscore query –* The big deal about obscore is: every ObsTAP-service is providing this exact datamodel. Thus, one can send the same query to several ObsTAP-services. Let's have a look at a obscore service: return to topcat, reopen the TAP service window, and at Keywords enter `GAVO`m click Find Service and access the GAVO data center at ARI. Search for `ivoa.obscore`, mark the table and look and open the metadata tab for columns. These columns are equal on all obscore services. Hence queries written with these columns names can be performed on all obscore service to figure out if a service provides observational at a given position (s_ra, s_dec), in a specific range of time (t_min, t_max) and wavelength (em_min, em_max). To get a bit familiar on how to use Obscore and TAP, open example2/obscore.py. After defining the access_url you find the adql_query. This is a very basic one for now. Note that at the bottom, we send the result of the query to Topcat. Now run the query to get an idea what how the script behaves.

Exercise: Change the ADQL-query so that it is performing a Cone selection around a given position at `(ra:240, dec:47)` of 1.2 degrees. For the sake of the tutorial keep the limit of 3 by using TOP 3(Solution: example2/exercise_coneselection.py)

Exercise: Take a closer look at the metadata browser and think about alternative solutions you could use to ask the question: "Is my given position within the coverage of a dataproduct on your service?" (Solution: example2/exercise_region.py) Think a moment about the difference of these two queries.

▷ 4 *Bringing Registry and Obscore together* – After completing the last exercise we now have the ADQL-query we can send to the list of services that we found by searching the VO registry. We just have to bring these scripts together and run all of it at once.

> **Exercise:** Combine the scripts. Use example1/exercise_svc_info.py to iterate over the list of services, but instead of printing the information of each service, use the content of example2/exercise_region.py and modify it to your needs. Set the limit in the ADQL-query to 15. Send the results of every ObsTAP-Service to Topcat via SAMP. (Solution: example2/exercise_iter_svc.py)

It's time to take a step back and look what we just did: We asked the VO-Registry for services providing observational radio data accordingly to the (easy access) obscore protocol, and then performed an ObsTAP-query on each of these services to ask them if they provide data at a given position. Compare this with the length of the python script: skipping the comments, the actual code is 13 lines (it could even be shorter, but then lose readability). Of course the secret here lies in the VO-standards, the metadata, and the datamodels which in combination can make data discovery very smoothly and efficient.

5 Accessing data: Datalink and SODA

Of course before the data access we still have to figure out if the data we found will suffice for our purpose. So we return to Topcat and have a look into the tables we just obtained. The first surprise: tough we queried several services, only one table (from the Astron Service) did return. The reason is quite simple: in our Obscore-query we specified a point that should be contained in area covered by the data product, defined by the obscore column s_region. So apparently the other service do not provide data products matching this condition. That makes it simpler for this tutorial – in the real world you may find yourself wondering which of the data products now really is useful for you. So have a look into the data in Topcat. Mark the table and click on Views → Column info. A new window will open which shows the metadata describing the columns of the table. Not surprisingly these are the very same meta data we saw in Topcat TAP meta data browser. Esepcially of interest is the column dataproduct_type: here we are searching for the entry `image` and luckily all obtained data products match this. The next column further describes a subtype of data product and here we see that one of the rows has the entry `intensity map`. A bit below one finds the description helpful for accessing the data e.g. access_url, access_format and access_estsize. Providing the url would make it easy to access the file, but the size (here given in kB – it's explained in the metadata!) shows that this intensity map is more than 1 GB, so not something one would want to download on a slow line, and process localy. The last column source_table describes in which table this very map would be accessible on the Astron-TAP-service. So let's look at the table `lolss.mosaic` in Topcat. Open the TAP window, search for `Astron`, select the service and wait for the metadata to arrive and then search for the table. In the table description we find that indeed the table only provides this one single mosaic image of the LBA survey of HETDEX, which we just assume would be feasable for the use case, but still we don't want to download the whole mosaic. If we look into the column metadata we easily figure that by querying this table we would not download the image as such, but rather the metadata of the image, and eventually there is a column pubdid which gives a good hunt what to do with the table. If you are familiar with VO-standards, then seeing this very column in a dataset, there's a good chance that the service is providing a datalink standard and further service capabilities.

Datalink

Datalink The Datalink standard is used for a variety of use cases. On can be that many datasets actually manifest as multiple files of several types, in which case datalink provides the metadata and links so that a client can access the elements of this dataset. Another use case is the description of Server Side Operations provided on datasets, e.g. image cutouts or slicing of cubes. In this case Datalink is used to describe the capabilities of a SODA-Service.

So let's look into this by daring to download a subset of this table:

```sql
SELECT TOP 3 * FROM lolss.mosaic
```

We will receive only one row (as there is only one mosaic image provided). Now in the Topcat main window click on Views → Activations Actions. In this window Topcat let's you configure the behaviour of how to deal with elements in a table that may demand or trigger interactivity and interoperability e.g. the behaviour of SAMP, or datalink. Select (not: check!) Invoke Service. You see the dropdown menug in the Configurations, which should be at `View Data Link Table`. Click on Invoke now on row 1 and you will get some information about the additional capability the astron service provides to this dataproduct.

Here you can see a description of of SODA service, which will let us perform image cutouts. So for each of our objects, we now could define the position and some space around them to perform a cutout, but doing so for a single position already is a bit annoying, not to speak of the 23 we have in our dataset right now. Yet, again: datalink and SODA are standardized, so maybe PyVO provides us with it ?

SODA

SODA (Server-side Operations for Data Access) describes the standard to provide the capability of performing server side operation that are not covered by other standards, e.g. an image cutout around a given position. SODA often is used in combination with Datalink

▷ **5** *Example3: Image cutouts in Python* – Indeed in the PyVO documentation (see the Helpful Links below!) explaines how we can use datalink to invoke a server side action. So in principle we could perform cutouts around our positions. In example3/astron_cutouts we show how this cutout can be performed for a single source given by position and a radius we just predefined.

You see that the actual cutout performance is surprisingly short and simple. A good bit of the script contains code that deals with real world problems: the Astropy SAMP Client (which is the same we use in pyvo) is a bit clunky when it comes to sending FITS files. To deal with this we defined a workaround and put it in a Context manager, which is very feasable: Context management makes sure that if you build some kind of infrastructure with python (e.g. open files), python will automatically deconstruct forgotten infrastructure (e.g. if you forgit to explicitly .close() a file). We will see later how that comes in handy. Don't worry if you don't understand this part of the code immediately, (in fact, when writing this script I had to remind myself what actually is happening there), just understand: we take the FITS file and prepare it in a PyVO-SAMP-Client compatible format.

Before we actually run the script, make sure that both, Topcat and Aladin are running. Topcat will provide us with a SAMP hub to connect to, and Aladin shall receive our data via SAMP. If yo run the script, after a few seconds, an image will be loaded to Aladin.

Now with this image comes the issue: we performed the cutour on the assumption that there will be a radio source at this very position, and we also guessed the size of the source. It would be much more precise if we had access to a service providing us with the capability of a crossmatch of a position, and some information about the size of a source. Luckily, we do have exactly such a servicen at hand.

Return to Topcat's TAP window on the Astron service where we earlier selected the table `lolss.mosaic`. Below this table you see `lolss.source_catalog` and though the name seems to give it away, we still need to look into the metadata to find the columns we want to query.

Exercise: What are the columns in lolss.mosaic which provide us with a position in (ra,dec) and the size of the source ?

Exercise: How would an ADQL query look like that performs a cone selection of the table `lolss.source_catalogue` around 100 arcsecs of the position (ra:240.484, dec:46.768) with the result only containing the columns from the exercise above ?

You will find the solutions to both above exercises in example3/exercise_adql.py. Run the script to get the solutions. Then open the script and have a look at what's happening there: especially the string interpolation will be useful for the next step.

Exercise: Extend the script example3/astron_cutout.py in a way so:
- You use the ADQL query of above example to perform a coneselection
- Use string interpolation to make the ADQL query flexible in regard of position
- as a bonus: when run standalone, the script shall search for (ra:240.484, dec:46.768), but the whole process shall be callable from an external script (this will become useful in the next chapter).
(Solution: example3/astron_smart_cutout.py)

We now have a PyVO script that will let us perform cutouts on the lolss-mosaic on the Astron-service for any position we feed to the script. Now let's see how to make use of this.

6 Interoperability: SAMP

In the script above we made use of SAMP to send tables to Topcat and FITS files to Aladin. Of course this kind of interoperability is nice to have and very useful already. But it's not a one-way: of course we can use SAMP to receive data from other client software in our Python scripts. This chapter will deal with this kind of interoperability.

SAMP

SAMP (Simple Application Messaging Protocol) enables astronomy applications on the same desktop or laptop machine to share data with each other. It also enables applications to notify each other about what data points a user has selected, enabling the two applications to coordinate their display and selection views.

▷ **6** *Example4: SAMP handles Tables –* Open the script example4/samp_table.py. Proceed to the main loop to figure out what we are doing here. As you see we are defining two functions within the main loop, which feels a bit odd in Python (not the best style practice). But in order to be able to call the connector object (which we use to connect to the SAMP hub), we have to do it. Alternatively one of course could use a global variable to workaround this namespace issue, but that would be way worse than defining functions in a the main loop.

The first of these two functions receive_call_table() (line 49) is capable of taking in the data from the pyvo.samp.SAMPIntegratedClient(). Note that this function has to take exactly 6 arguments. Some clients expect a reply to SAMP messages they sent, so let's be polite here and send this reply. At this time, ignore the second function, because the script does not make use of it yet.

Let's jump to line 81 where we build the SAMP object to connect to the SAMP hub. With client_name and description we defined parameters which be be published to the hub and other clients. But a good bit of the SAMP magic lies in the capabilities which we bind to the mtypes in the following lines after client.connect(). With client.bind_receive_call () we bind a function to a SAMP message type (mtype). So in our case this means: when a SAMP message of the type `"table.load.votable"` comes in, the script shall call the function receive_call_table(), from which we call def magic_table(params). So in order to perform some magic in reaction of this SAMP action, we can write our Python logic there. For now not much is happening, but let's run it to see if it works as expected. Make sure you have Topcat up and running. After you start the script, switch to Topcat, mark the table objects.vot (the first one we started with), and then go to Interop → Send table to... and select `PyVO: simple SAMP`. Then return to the running script and look at the data that came in. As you see it's a python dictionary containing the name, the table_id and a local url. So actually we did not receive the table yet but instead received information to access the table on the local machine. For the following examples don't forget to look into the Helpful Links below.

Exercise: Use the params to load the table into the script and let the script print the table. (Solution: example4/exercise_read_table.py)

See that we added just two line to read the table from the url in the parameters. That was fairly easy. Now look a bit down and see that we pulled part of the SAMP connector out of the main loop and added a @pyvo.samp.contextlib.contextmanager. Remember the construction and deconstruction of infrastructure above ? Now have a look into the SAMP hub status in Topcat – by now there surely will be quite a few Zombie-Client from Python already. With the context manager we avoid this mess. So stop all scripts and close Topcat and Aladin to clean the hub. Start Topcat again and load the table objects_colours.vot – that is the table already containing SDSS colours. Now rerun the script, load the table to the PYVO SAMP handler, end the script and see the clean SAMP hub.

▷ **7** *Example4: SAMP handles Coordinates* – So now you should have a bit of a feeling how to configure your SAMP-Client in Python. Let's see how well you do:

> **Exercise:** Modify example4/exercise_read_table.py so that
> - Make the SAMP Client handle Sky Coordinates too
> - Use the binding and calling functions
> - Just use print to show (ra,de).
> (Solution: example4/exercise_coords.py)

Note: You will have to configure Topcat's Activiation Actions Send sky coordinates for testing.

So now we have a samp handler which can deal with Sky coordinates. Remember that our cutout script actually was able to work with position? Well, wouldn't it be useful if we can select a single source in Topcat, send the Coordinates to the script to perform the image cutout and from there sent the FITS-file to Aladin ? Would this be hard to accomplish ? Let's try.

▷ **8** *Example5: Cutouts via SAMP* –

> **Exercise:** Based on example4/exercise_coords.py :
> - Configure the SAMP Client to deal with Coordinates only.
> - Give it a meaningful name e.g. PYVO: Astron cutouts
> - import example3/astron_smart_cutout.py and pass the Coordinates so that the cutout can be performed. Best put both scripts in the same folder.
> Hint: all changes will apply to the samphandling script, not to the cutout script.
> (Solution: example5/)

Now that was fairly easy: we moved the scripts in the same folder, and in astron_samp_handler.py removed everything dealing loading a table, added the import of the cutout script and passed ra and dec to the function `recipe(ra,dec)`. We now have some control over the cutouts through Topcat. Now let's deal with row highlighting

▷ **9** *Example6: Row highlighting* – In this final part we want to configure a SAMP Client to react to row highlighting from Topcat. There are a few caveats here: In order to be able to highlight a table row, the table needs to be available for both SAMP-Clients, the Sender and the Receiver. So our SAMP handler in Python needs to accept the table and be able to react to the row highlighting.

> **Exercise:** Based on example4/exercise_coords.py :
> - Configure the SAMP Client to deal with row highlighting.
> - Give it a meaningful name e.g. PYVO: row highlighting

- As a reaction to row highlighting let the script print the params received from topcat.
Hint: An easy way to work with this: store the received table in a dictionary so you can easily identify it and access it.
(Solution: example6/exercise_row_hl.py)

7 All together

In a final step we want to present what you actually can achieve with the magic of a lot of the VO standards. Have a look in the folder all_together: You see 4 scripts. You are already familiar with the astron part of it, new is the sed part. There we make use of the colours in our table (remember the first steps of this tutorial ?) for SED plotting. Make sure you have Topcat and Aladin running and the table objects_colours.vot loaded into Topcat. Then start the scripts sed_samp_handler.py and astron_samp_handler.py from two different commandlines. Go to Topcat and send the table via SAMP to the Client `PyVO: SED`. Then open the Table Data View. Eventually you want to activate row highlighting also with this Client. Click on one of the rows and wait for the plot. Click on a different row and wait for the plot to refresh. If you find a plot you are satisfied with, use Send Coordinate to send the coordinates to the PyVO: Astron cutout Client and find the cutout in Aladin.

To fully understand what is going on in the SED plotting, you will have to have a look into UCDs. Look into the files sed_samp_handler.py and sed.py to figure out which data we needed from the SAMP handler, and how the SED plotting works with the UCDs.

8 Acknowledgements

The Authors want to thank for making this contribution to ADASS 2021 possible:

- ESCAPE - The European Science Cluster of Astronomy & Particle Physics ESFRI Research Infrastructures has received funding from the European Union´s Horizon 2020 research and innovation programme under Grant Agreement no. 824064.

- Markus Demleitner for sharing ideas and quite a bit of code snippets.

- Everybody who was involved in coming up with the SAMP standard. Well done indeed!

Dave Morris works as a Research Software Engineer at the Royal Observatory, Edinburgh (ROE)

Hendrik Heinl works as Ingénieur d'études at the Centre de Données astronomiques de Strasbourg (CDS)

9 Helpful Links

Example1: Registry

pyvo.registry.search

Registry Search Results

Example2: Obscore

ADQL-Geometries

Obscore Standard

Example3 Datalink and SODA

Datalink and SODA in PyVO

ADQL-Geometries

Example4: SAMP

Astropy and VOTable import

SAMP Client in Astropy

SAMP Mtypes

Example6: SAMP-row highlighting

Astropy and VOTable import

SAMP Client in Astropy

SAMP Mtypes

10 Standards

ADQL

Datalink

Obscore

Registry

SAMP

SODA

UCD

VOTable

11 References

Demleitner, M. ADQL-Course

Demleitner, M., Becker, S. PyVO Documentation

Fernique, P. Aladin Documentation

Taylor, M. TOPCAT Documentation