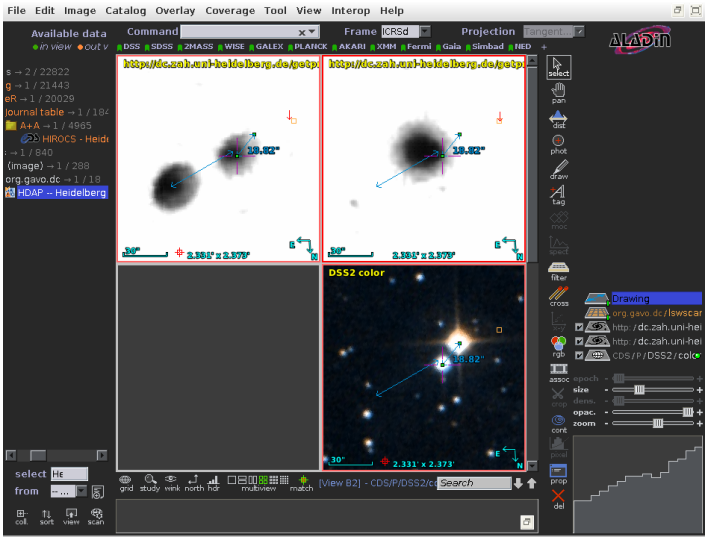
Plate Scans in the VO



Markus Demleitner

April 25, 2024

Abstract

This tutorial introduces a few techniques for working with image services in the Virtual Observatory (VO) in general, using services containing plate scans as examples. It will discuss both exploratory, interactive use, and scripting using pyVO.

Software: Aladin, pyVO

1 Prerequisites

To follow what is done here, you should at least install Aladin (Centre de Données Astronomiques de Strasbourg (CDS), 2011)[1]. and TOPCAT (Taylor, 2011). If you'd like to reproduce the part on programmatic access, you will also need Python, astropy, and pyVO (Graham and Plante et al., 2014). If your system does not already have them (or makes them available through a package manager), download links are found in the bibliography.

[1] As some of the UI elements may change in the next few years: this text has been produced against version 10.111.

2 A word of caution

At this point, images can be published in the VO through three different protocols:

- SIAP, version 1 (SIAv1) – this has been around since the dark ages of the VO and is still the protocol best supported in clients; the one reliable constraint it offers is the location of your region of interest on the sky.
- SIAP, version 2 (SIAv2) – an incompatible update to SIAP mainly intended to better support image cubes. This also lets you reliably constrain time and waveband of what you are looking for.
- ObsTAP – a powerful method to publish observations (not only images) of the celestial sphere. On the wire, you can query these with the full power of the ADQL query language.

The problem: A given observation might be available through any subset of these three, while data discovery and service operation is not done uniformly among the three by clients and libraries (and it'd be difficult to do that). If you really were to request "all images in the VO", you would have to query through all three of these and then remove duplicate (or triplicate) answers. This is possible with pyVO, but it's fairly hard. Hopefully, a few years down the road everyone will have their data at least also in ObsTAP. Meanwhile, we largely ignore the issue of completeness here and use the various technologies as convenient.

3 Simple Discovery in a Known Service

Say you're interested in the double star WDS 22468+4420, which, according to the Washington Double Star catalog (if you want to look yourself: TOPCAT is your friend), has a pair of stars with a separation of 37.6 arcsec, magnitudes 10.5 and 11.9, and a rapidly changing position angle. This looks like a nice case to follow an orbit on plate scans.

For this first use case, we play old style and use a single, pre-known archive. For the sake of example we'll be using that of the Landessternwarte in Heidelberg. The admittedly more interesting case of "blind discovery", which is what we are really after in the VO, will be discussed in the next section.

Whenever you're dealing with images in the VO, your first stop should be the Aladin "client"[2]. So, start it and enter the object name (i.e., `WDS 22468+4420`) into the Command box (anything Simbad can figure out will do here, as will coordinates).

[2] A "client" is a program speaking to services out on the net, and much like your web browser talks to web servers using HTTP and HTML, Aladin communicates with the services in this example using HTTP, SIAP, and VOTable.

Aladin will then load a background image from some survey (probably DSS2) and take you to the object. Zoom in (use the mouse wheel or the slider in the lower right of the Aladin window). You can also try some other backgrounds (remove planes by right-clicking on them in the stack and selecting delete from the pop-up menu).

When you've seen enough, zoom in to a field of view (FoV) around WDS 22468+4420 of about half a degree or less; you don't want to transmit too much data for the object's surroundings when you're really only interested in the binary. The FoV is given center bottom in the image pane.

Figure 1: A portion of Aladin's discovery tree after looking for "Heidelberg" and clicking on the HDAP entry, ready to query the remote service.

To find images, you will have to query one or more services for them. The VO is built from many of them, so there's not one single point that you can ask "are there images matching X and Y?" However, there *is* a place to ask "are there services matching X and Y?": The VO Registry. X and Y in this question could constrain, for instance, names, coverage, descriptions, and service types. From such a query, you will not directly get back images, but only services you can query for images[3].

In Aladin, this Registry is exposed in a tree on the left of the window. If you don't see it there, look for three small arrows on the left edge of the window and click those to show the discovery tree. By our simplifying assumption above we will initially restrict ourselves to the archives of Landessternwarte Heidelberg-Königstuhl as an example. So, enter "Heidelberg" in the select field near the bottom of the left-hand pane. You'll see that the discovery tree updates itself, and you should be seeing something called "HDAP" in a branch called "SIA (Image)".

To query the service you've found in this way, left-click the HDAP entry in the discovery tree. You will see a dialog pop up. In it, make sure "in view" is checked. When you hit Load, after a few seconds you'll see a new plane (with a label like "org.gavo.dc/lswscans" or so) in the plane stack on the right, and a table with below the main image display (double click on the plane label if it's not there).

[3] In case you're wondering why the VO works like this: Very briefly, building a common metadata repository for the services works reasonably well. Building a common metadata repository for all the datasets, on the other hand, turns out to be a largely hopeless effort for many reasons. The best we can do is make all the archives offer the same interface. If you like fancy words: This is called a "distributed information system".

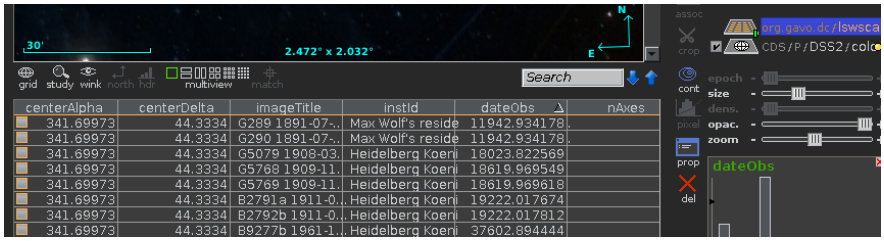


Figure 2: The SIAP matches from HDAP, sorted by dateObs. Mouse over the table rows to see the footprint of the corresponding image.

If you scroll the table a bit to the right, you'll see a dateObs column, which has the observation dates for the various plates as MJD. Sort by that column by clicking the column head for dateObs (cf. Fig. 2 to see how this should look like). If you mouse over the table rows, you'll see footprints of the results. For this particular service (which produces cutouts of large plate scans on the fly), it's usually a bad sign if they're not basically covering the FoV, since you're then either at the edge of a scanned plate, or the astrometric calibration went haywire.

By this criterion, at the time of writing all of the observations made with Wolf's Doppel-Astrograph (image titles starting with G) won't work, which is a bit of a pity since they'd expand the time axis a good deal into the 19th century. However, you'll see there are at least images from 1911 and 1961. So, load B2791a, and B9277 by scrolling the table display left again and clicking the corresponding buttons in the accref column.

Once you have multiple observations of the same field, you would clearly like to compare them. For starters, let's see the old exposures together with the DSS2 background. So, click the checkbox on the the DSS2 plane[4]. Then you can blend in any of the plates by grabbing the little slider under their icon and sliding it right[5]. Figure 3 shows how this could look like. Notice how the astrometric calibration by and large matches, but some stars have moved quite a bit between 1911 (when the plate was exposed) and the late 20th century, the epoch of DSS2; note that right-clicking on a plane and selecting Properties lets you inspect the FITS header (or similar metadata) of a plane. Blend in both the 1911 and the 1961 plate to get a feeling for what's moving how.

At this point you should further explore Aladin. Recommended things to try:

- Use the Dist button to the right of the image area to measure the distance and position angle of the connection between the two stars having an encounter here. Note that dragging with the right mouse

[4]If your Aladin still jumps away from your target when you do that (a little bug), simply use the arrow in the Command selector and select your input "WDS 22468+4420" again to jump back.

[5]Alternatively, you can use the opacity slider when the plane is selected.

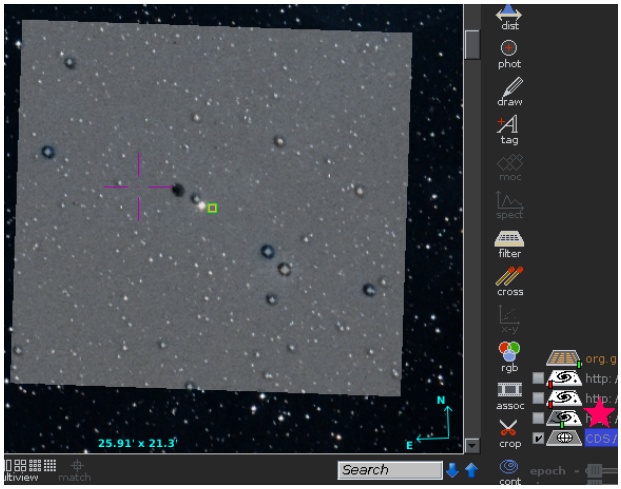


Figure 3: Blending a plate into a HiPS background. The red star marks the icon with the opacity slider.

button pressed (or using the Pixel tool button) you can change the lookup table, which helps when finding plausible centers in these fairly saturated objects.

- Blink the 1911 and the 1961 image (Image menu, Blink/Movie generator entry to see our target star move.

- Select View → Panels → 4 Panels to see four panels at once, drag each of the three image planes you should have by now into one panel and hit Match just below the image area so the panels all share one WCS (try it by zooming in and out). You'll also see your measurements mapped in all planes.

The upshot is that around 1961 the brighter of the two stars around the position of WDS 22468+4420 overtook the fainter one.

Let's use Gaia to gain some more insight as to what the double star components from WDS might be. For this simple application, you could just use Aladin's built-in Gaia HiPS (above the image area). But let's use proper data discovery and enter "Gaia" in the Select field below the discovery tree.

After switching back to the 1-panel view and zooming to a FoV of perhaps 10 arcmins, click on "GaiaSource DR2" in the discovery tree (if you read this in the more distant future, later releases should of course also work), and you'll see another query dialog of the type we've used to query the image server above. Again, "in view" should do what we want, so hit Load. On top of your image display, you will now see lots of little crosses for the stars Gaia has identified in this area. Sure enough, nothing really matches the fast star; Gaia's epoch is far beyond either DSS2 or our plates.

To get a grasp of what is going on, we'd like to have some idea how bright the stars corresponding to the little crosses are, and how they move. In Aladin, you do this with filters; use the tool button right of the image pane,

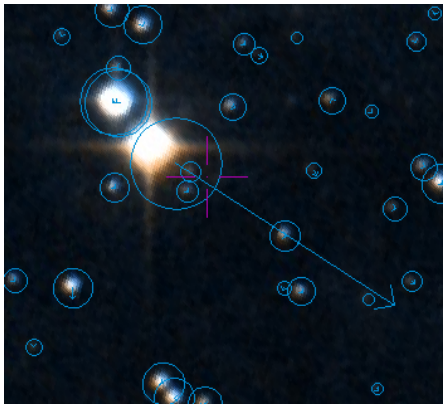


Figure 4: The halfway modern (DSS2) situation with Gaia data overplotted, showing both magnitude and motion of the star.

not the filter below the discovery tree. You can get really fancy here, but the simple "Draw circles proportional to the object luminosity" or "Draw proper motions" will get you a long way here. Fig. 4 suggests strongly that while the fainter star is indeed a binary (resolved by Gaia but not our ground-based, pre-adaptive optics plates), the wide-separation, fast-moving star that WDS (with half an arcmin separation) probably assumed to belong to the system is an artefact of the high proper motion of the object[6].

By the way: For catalogs in which Aladin can find proper motions and the necessary metadata, it will have an epoch slider (select the Gaia plane to see it). Playing with it also gives nice insights into the kinetics in our field.

4 Going All-VO, and: Datalink!

The great thing about the VO is that everything you've done so far works pretty much the same for all VO-enabled services.

Let's have a slightly different use case: Let's investigate the time axis in the Orion Nebula (M 42); we still have old observations in mind and thus will limit our query to optical image services.

To start with a clean slate, clear the planes you have accumulated so far by selecting your DSS2 plane and selecting Delete all other Planes from the context menu.

Also, clear the Select field under the discovery tree.

To restrict the queries we're going to run to only optical image services, we have to create a discovery tree filter in Aladin. This happens by hitting the icon next to the from selector (see the red star in Fig. 5). Our constraints translate to a check at Optical (as set in Fig. 5) and one at SIA(1&2) – that's the image service part – in the Technical tab. This should leave of order

[6] Unsurprisingly for a nearby dwarf, this fast star is also known as a flare star, EV Lac. How did I find this out? Try the Simbad pointer in Aladin – the Study button below the image pane lets you switch it on.

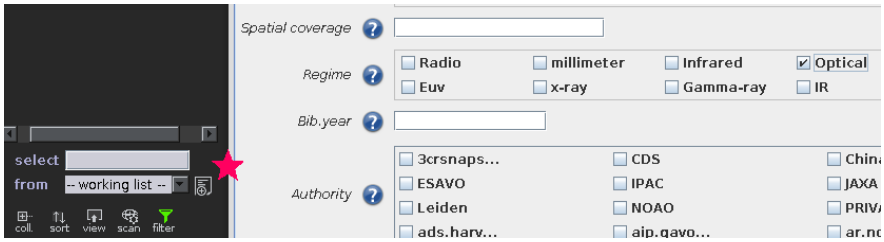


Figure 5: Building a filter for optical resources in Aladin. The little red star indicates the button that creates a new discovery filter.

100 resources to query.

You *could* query all of these by clicking on the head of the SIA branch and hitting Load. For about a hundred resources that's still realistic (you should avoid querying more than, say, 500 resources at a time as part of a regular workflow; there are probably better ways to do what you're trying).

Now that we have the subset of services we might want to query, go to our target object (use the M42 we've suggested above for starters so you'll see the nice data we'll be discussing below). As above use the Command input box. To both find interesting data and avoid huge result sets, zoom to a FoV of about a degree.

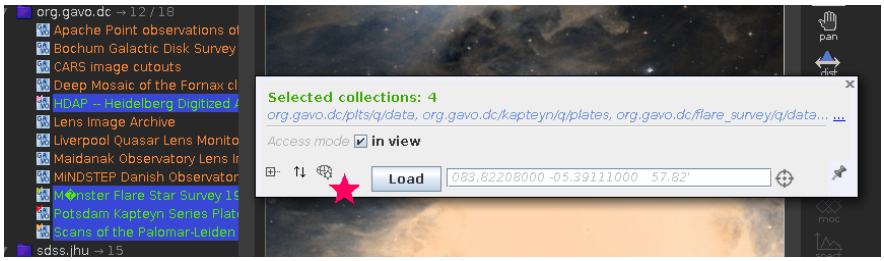


Figure 6: Querying four data collections at once in Aladin. Here, the resources have already been filtered for having data in the selected field using the button marked with the red star.

You can now also hit any individual service in your discovery tree that raises your curiosity see what images they have, but many of them will return empty. It's much less frustrating to instead use the little map logo in the discovery dialog (marked with a little red star in Fig. 6); this will run queries against all the selected archives and paint the ones that have data green.

Try it with the org.gavo.dc tree by clicking on the head of this branch and clicking the little map icon in the load dialog. You'll see that several services will end up orange, others green. These latter will return data. Now, either select a subset of them using control-clicks and use Load in the search dialog. Or simply click the view button at the bottom of the discovery tree to hide all services without data and then click on the head of the branch

(in this case, `org.gavo.dc` to query that subset. You will get one plane per service.

If you inspect the accsize column in the result table – that's the size of the file in bytes – , you will notice that some of the scans are really large and therefore load slowly. This hasn't been much of a problem above because the HDAP service automatically does cutouts to the size of your search regions, but not many SIAP services do that.

To have a flexible and discovery protocol-independent solution to this problem, the Datalink protocol was developed. It lets you, for instance, specify arbitrary cutouts and also link additional data (for plate scans, that could be anything from grayscale wedges to lists of extracted objects).

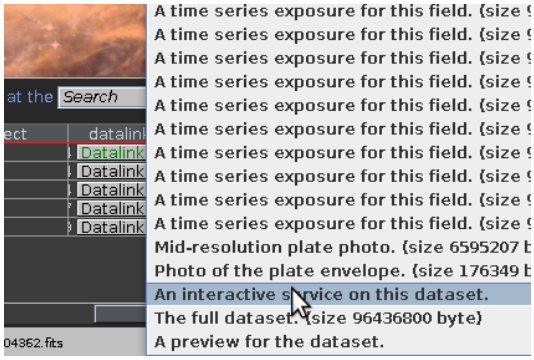


Figure 7: Aladin's representation of a datalink document for a master plate of the Münster flare survey.

To see how that works, have a look at the plates from the flare survey. At the right end of the result table, there's a column "Datalink"[7]. Click on it. You should see a popup somewhat resembling Fig 7 listing lots of "related" artefacts

In this case, this includes photos of the plate envelope and the plate itself. You can view the photos Aladin, but since they don't have an astrometric calibration, there's not much more you can do with them. There's also the full dataset and the cutout service (which you cannot yet operate from Aladin stable at the time of writing). And, as a special feature of this particular data collection, links to datalinks for plates with timed exposures for which this selected frame is the astrometric master frame.

Try one of "time series exposures": Follow the datalink, perhaps have a look at the plate photo, an then load the full dataset (caution: it's about 100 MB). You will see that each star on the frame has multiple images distributed roughly equidistantly along a line. Well, the plan back when the plates were taken has been to detect young flare stars, and to find them sticking out on the plates, each plate was exposed multiple times, where, an between each exposure, the camera was moved a bit. Hence, you can generate a lot of simple photometric time series such plates (if with a bit of effort).

[7]Other services, including for instance CADC, may deliver datalinks as the actual product; in that case, you will see the datalink selector when clicking on the product button.

This kind of data lends itself to try the simple photometry built into Aladin (which isn't intended to compete with Sextractor – on the contrary, there's a Sextractor interface built into Aladin). There is an additional complication here, though: you will notice that on many frames, even constant sources seem to vary, perhaps because of passing cirrus clouds, perhaps because the exposure times weren't quite equal. So, to detect intrinsic variability, we need to divide fluxes we get by those of a constant source (also known as: about any star on these time scales). Because of the nonlinear response of plate emulsions, comparison star need to have comparable brightness.

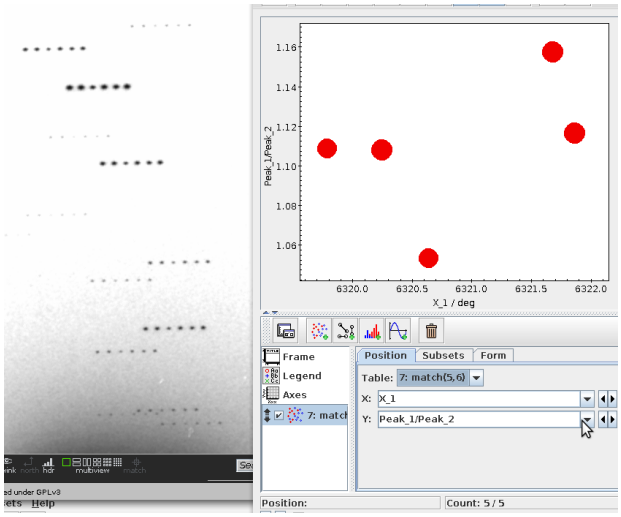


Figure 8: Our ad-hoc photometric timeseries, mixing Aladin and TOPCAT

As an example for a nontrivial, cross-tool workflow, let's try an ad-hoc relative photometry (see Fig. 8 for an illustration of what we're after). The plan is: Measure the pixels for two stars, join the two time series, and compute the quotients. Because the latter workflow is smoother in TOPCAT, we'll use that to do it.

So: Start TOPCAT and...

▷ **1** *Generate the first time series* – In Aladin, look for a pair of dot trail with roughly comparable brightness. Select the phot tool and click on each dot in the first line once, left to right. You'll get a new plane (called "Drawing"), and you'll see the extraction result in the table area.

▷ **2** *Send the first time series to TOPCAT* – Right-click on "Drawing" and select Broadcast to all SAMP applications. You'll notice that your table pops up in TOPCAT.

▷ **3** *Generate and send the second time series* – Delete the "Drawing" plane and repeat the clicking and sending you just did for the second dot trail.

▷ **4** *Combine the two time series* – In order to compute quotients, you will have to have your two measurements in one table. So, we'll have to join the two extraction results. To do that, in TOPCAT, click Joins → Pair Match. You'll want to match by row; you get that setting Algorithm to "Exact Value"

and both Matched Value columns to "Index" (you'll have to manually enter that, because TOPCAT doesn't expect this kind of slight abuse of a facility originally intended to do crossmatches). Then hit Go. You'll see a new table called something like "match(n,m)".

▷ **5** *Plot the thing* – As a surrogate for time, use the x pixel coordinate; the dependent variable simply is the quotient of the photometric measurements. So, hit Graphics → Plane Plot in TOPCAT, and in the Position Tab enter `X_1` for X and, perhaps, `Peak_1/Peak_2` as Y. To make it prettier, in the Forms tab use + Forms → Add Size and configure the size as, say, `Peak_1`[8].

You should see a time series; with a bit of experimentation, you should be able to figure out what's a significant effect in these quotients.

Of course, you wouldn't want to do something like this a thousand times. So, if some exploratory analysis indicates this kind of investigation could yield interesting results if applied at a larger scale, it's time for programmatic access.

5 Programmatic Access

Using standard interfaces to publish data not only lets you operate all such services in a uniform way as we've done so far. It also lets you swap in other clients when you have a different use case. For instance, you can also use the pyVO python package (working on top of astropy) to operate services, which in particular greatly facilitates larger scale tasks or scripts integrating VO workflows with further analysis.

As an example, assume you are investigating some lensed quasars and you are looking for historical exposures of them. To keep this simple, we'll be content just showing some statistics on what's available.

Whenever you have multiple objects to look at, you should seriously consider using the Table Access Protocol TAP. This lets you upload tables (like, in this case, object lists) to servers and operate on all of them at once. This is marginally possible with SIAv2, but TAP's facilities are far more flexible. The downside is that to fully use TAP services, you really should learn the query language ADQL (a dialect of SQL) – but that's an excellent idea anyway.

For uniformly dealing with observations within TAP services, the VO defines a "data model" called ObsCore. In essence, a "data model" here is just a set of columns in a database table. To get an idea what they are, start TOPCAT, enter its TAP client, look for tables called "ivoa.obscore" (that name is also part of the ObsCore standard), choose any of the many services TOPCAT will show you and inspect the columns of that table in TOPCAT's metadata browser.

[8] Yes, the peak isn't an ideal measure, and neither is, probably, the FWHM that you could use as well. If you'd like to obtain robust photometry in a workflow like that, the Aladin team certainly welcomes your suggestions. Incidentally, improvised aperture photometry is available in Aladin when you click and drag in the with the phot tool.

To address our use case, we will first have to find TAP services with obscore tables. The equivalent of Aladin's discovery tree is contained in pyVO's `registry` module. The concrete incantation for obscore is

```
for svc_rec in pyvo.registry.search(datamodel="obscore"):
```

This will execute the loop body once for every service that claims to have an ObsCore-compliant table, and in the loop body we can see various metadata on the service (like its access URL) within `svc_rec`.

To run the same query with the same upload on each of these, within the loop run

```
svc = pyvo.dal.TAPService(svc_rec.access_url)
result = svc.run_sync(QUERY, upload={"myobjs": targets_table})
```

`result` then is a somewhat magic object that (on successful execution on the remote side) will have an astropy table in its `table` attribute (newer pyVOs want you to call `to_table()` to get it).

What remains is to construct the query. It is strongly recommended to use TOPCAT's nice TAP interface interactively to do that. Note in particular the Examples button in the dialog in which you will find pre-written queries that quite often will help you get started (load a table with positions in it into TOPCAT to enjoy the full power of that feature).

In this particular case, the query is reasonably simple (at least when you start from the "upload join" example:

```
SELECT TOP 1000
  i.t_exptime,
  (i.em_max+i.em_min)/2 AS em_mean,
  (i.t_max+i.t_min)/2 AS t_mean,
  i.instrument_name,
  access_url
FROM ivoa.obscore AS i
JOIN TAP_UPLOAD.myobjs AS m
ON 1=CONTAINS(
  POINT('ICRS', m.ra, m.dec),
  CIRCLE('ICRS', i.s_ra, i.s_dec, i.s_fov))
WHERE dataproduct_type='image'
```

(In live use you may want to remove the `TOP 1000` here). As usual when you're programming and want to talk to the world, there's a bit of plumbing to be done to make it all work. You should find an assembled program in this PDF's attachment[9]. Running it will produce (quite a few complaints about broken services and) eventually an output somewhat like this:

[9] Or at http://svn.ari.uni-heidelberg.de/svn/edu/trunk/gavo_plates/res/all_obscore.py.

```
Column t_exptime: 2780 values
  Min   12, Max 15300, Mean 370.722
---
Column em_mean: 3801 values
  Min 1.8081e-09, Max 9.3e-07, Mean 6.40804e-07
---
Column t_mean: 4067 values
  Min 12564.5, Max 58126.3, Mean 51909.1
---
Column instrument_name: 4067 values
  Matches from ROSAT HRI, ROSAT PSPCB, 3.5m APO, ROSAT PSPCC,
    DFOSC_FASU, Calar Alto (493), Schmidt, Heidelberg Koenigstuhl
    (24), Bruce Astrograph, Heidelberg Koenigstuhl (24), Wolf's
    Doppelastrograph, AZT 22, Max Wolf's residence in Heidelberg,
    Maerzgasse, Wolf's Doppelastrograph, DK-1.54, Zeiss Triplet 15 cm
    Potsdam-Telegrafenberg
---
Column access_url: 4067 values
---
```

It will also leave the raw results in a file `allmatches.vot`, which you can further explore with, for instance, TOPCAT.

To actually work with pyVO, you should probably have a longer look at GAVO's pyVO course[10], which also include lots of other cool stuff you can do with it and the VO.

References

Centre de Données Astronomiques de Strasbourg (CDS) (2011), 'Aladin: Interactive Sky Atlas', Astrophysics Source Code Library, ascl:1112.019. http://ascl.net/1112.019

Graham, M., Plante, R., Tody, D. and Fitzpatrick, M. (2014), 'PyVO: Python access to the Virtual Observatory', Astrophysics Source Code Library, ascl:1402.004. http://ascl.net/1402.004

Taylor, M. (2011), 'TOPCAT: Tool for OPerations on Catalogues And Tables', Astrophysics Source Code Library, ascl:1101.010. http://ascl.net/1101.010

Find me on VO Text Treasures http://dc.g-vo.org/VOTT

[10] http://docs.g-vo.org/pyvo