Scope definition

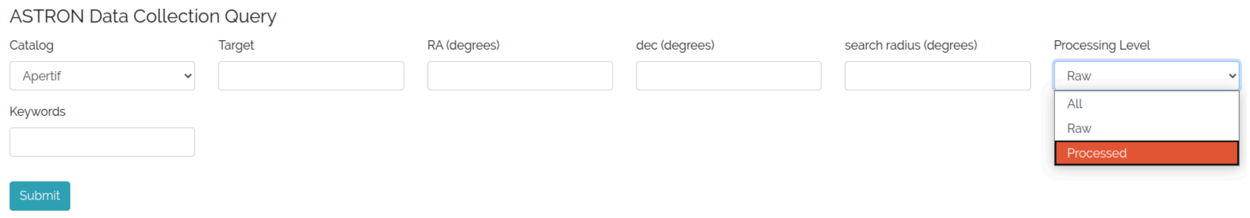
- Query schema of VO registry
- Steps of VO registry query

Example Query Schema

Please suggest editing the example query schema below:

```
# definition of the query fields and values
query_schema = {
        "name": "adex",
        "title": "ASTRON Data Collection Query",
        "type": "object",
        "properties": {
                "catalog": {
                "type": "string",
                "title": "Catalog",
                "default": "apertif",
                "enum": ["all","apertif", "astron_vo"],
                "enumNames": ["All","Apertif", "ASTRON_VO"]
                },
                "target": {
                "type": "string",
                "title": "Target"
                },
                "ra": {
                "type": "number",
                "title": "RA (degrees)",
                },
                "dec": {
                "type": "number",
                "title": "dec (degrees)",
                },
                "fov": {
                "type": "number",
                "title": "search radius (degrees)",
                },
                "level": {
                "type": "string",
                "title": "DataProduct Level",
                "default": "raw",
                "enum": ["all","raw","processed"],
                "enumNames": ["All","Raw","Processed"]
                },
                "category": {
                "type": "string",
                "title": "Keywords",
                },
        }
}
```

Rendering this query schema gives the query form that looks as follows:



ASTRON Data Collection Query

Catalog: Apertif

Target

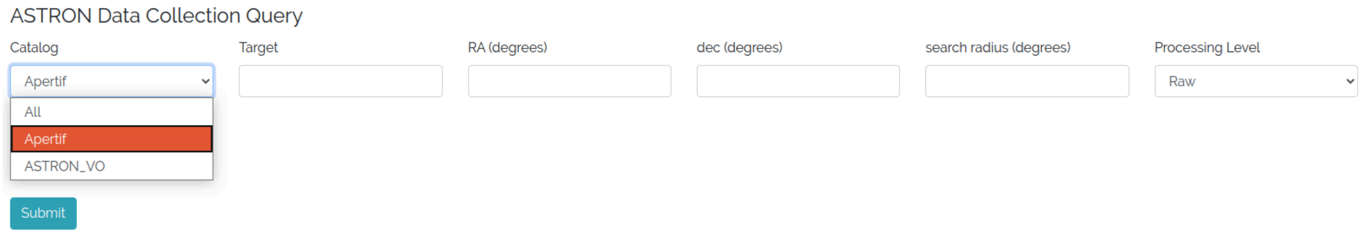
RA (degrees)

dec (degrees)

search radius (degrees)

Processing Level: Raw

Keywords



ASTRON Data Collection Query

Catalog: Apertif

Target

RA (degrees)

dec (degrees)

search radius (degrees)

Processing Level: Raw

VO Registry Query steps

The following describes some use cases that would use the VO Registry to discover data and data services.

The VO Registry allows you to query for certain for certain service types, including but not limited to: *sia, ssa, scs, tap*

- **TAP**: Tables
- **SCS**: Cone Search
- **SSA**: Spectra
- **SIA**: Images

Use Case Scenarios:

Scenario 1:

User arrives at the search page. They are given a list of service types to select from, including catalog data, images, spectra, cone search services, all of which correspond to equivalent VO Protocols (listed above). Along the service types there is a textfield/textarea for searching based on a keyword.

Based on the type of service they have selected, there is an optional set of parameters displayed, which are checkbox options corresponding to the keyword search.

For example:
If the catalog data option is selected (TAP), users can select whether the keyword should match: *Table Names, Table Descriptions, Service Name.*

There are also a number of checkbox options that are default and shown for all service types: *Short Name, Title, Subjects, ID, Publisher & Description*

Similarly, all these options define whether the keyword search should return results that match these parameters to the keyword.
An optional parameter that can be used in this search is a *waveband* (i.e. waveband = "x-ray").

Aside from the above, an option for a user to manually enter a service URL can be provided, for users who are already aware of the service they want to access.

Once the user has selected the options and entered a keyword to search for, they can hit the "Search" option, which will return a list of services that correspond to that search.
The result list should at the very least contain a *short name,* a *title,* the *subjects* (short description of service), and an *access URL*.

The user can select the service they want to access, and launch a second (query) page, where they can run a query to access data from that service.

The query page would most likely depend on the service type defined above.

For a TAP service a user can:

- Display list of schemas/tables & columns for the selected service
- Create SQL query manually

- Generate SQL query through form (i.e. select [cols as checkboxes] from [table name as checkboxes] where [where criteria as text?] and ...)

For a Cone Search access service:

- Provide text field options for defining *RA, DEC, Object name & Radius*

For an Image access service:

- Provide text field options for defining *RA, DEC, Object name, Angular Size & Image format*
- Provide Image format dropdown with the following options: *image/fits, GRAPHIC, ALL*

For either of these options, a query should probably display a new page (stateless, i.e. can copy paste URL and get the same results) with a list of rows in the case of tables, or a list of images in the case of an image search. From here on they can either be presented with a data-cart functionality for exporting the data, or an interactive way for navigating through the results, and saving the results locally etc..

Scenario 2:

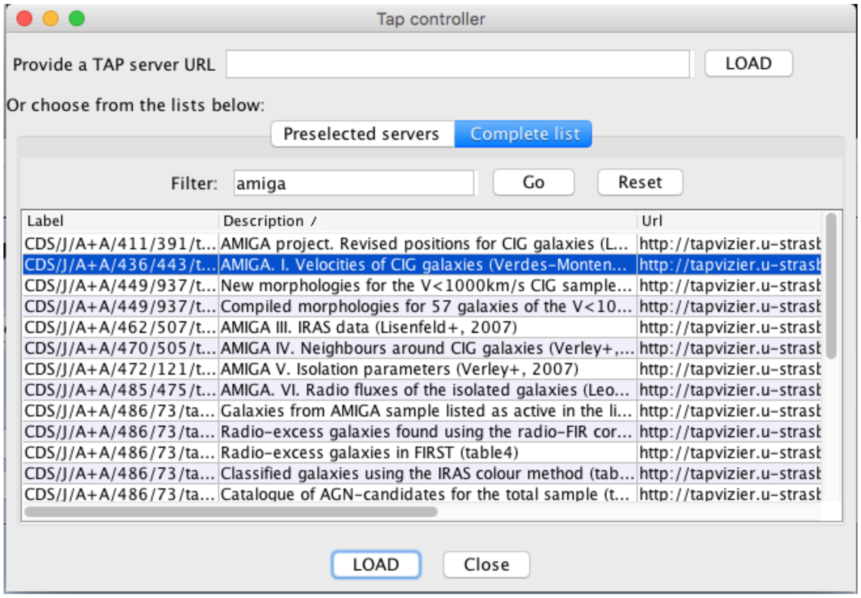
User arrives at the search page. Using e.g. a TAP query they discover a set of objects of interest. It is likely that they would then like to search for and retrieve any imaging and spectral data that are available for the objects that they are interested in. If it does not “come for free” as part of the VO protocol, then it might be a nice value-added feature of the ESAP VO query interface to enable query chaining or joins between related queries for different data related to the same objects.

Concretely this might involve extracting RA/Dec coordinates and object angular extensions from the results of a TAP query and using these to execute image access queries for the corresponding objects.

Scenario 3: connecting ESAP to other VO tools through SAMP

SAMP Use Case 1: TAP services and Aladin

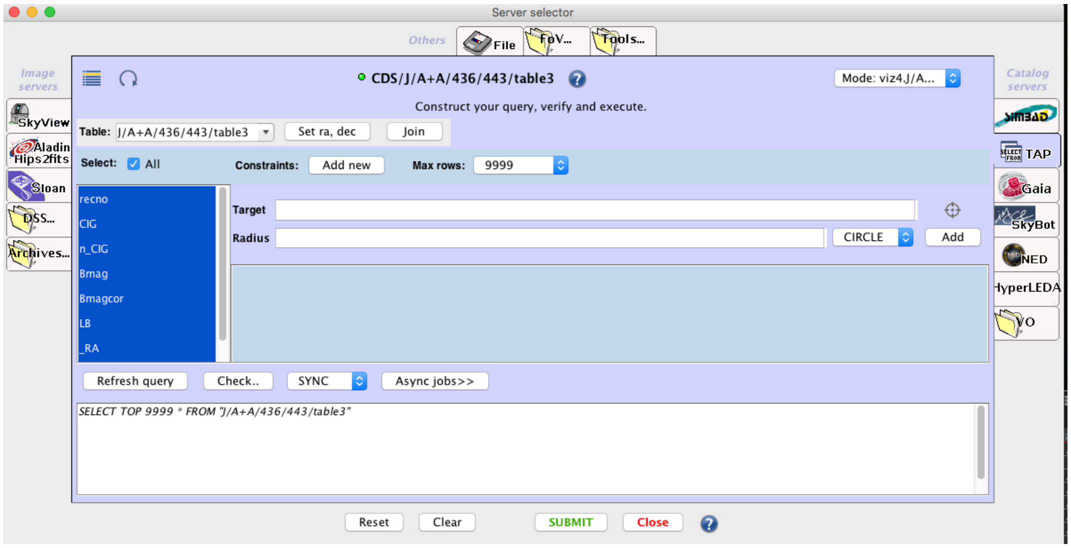
- An user opens the server selector window in Aladin, and using this GUI, he/she filters the complete list of TAP services using some keywords. Then select (LOAD button) the TAP service that he/she needs.



Filter: amiga

Label	Description /	Url
CDS/J/A+A/411/391/t...	AMIGA project. Revised positions for CIG galaxies (L...	http://tapvizier.u-strast
CDS/J/A+A/436/443/t...	AMIGA. I. Velocities of CIG galaxies (Verdes-Monten...	http://tapvizier.u-strast
CDS/J/A+A/449/937/t...	New morphologies for the V<1000km/s CIG sample...	http://tapvizier.u-strast
CDS/J/A+A/449/937/t...	Compiled morphologies for 57 galaxies of the V<10...	http://tapvizier.u-strast
CDS/J/A+A/462/507/t...	AMIGA III. IRAS data (Lisenfeld+, 2007)	http://tapvizier.u-strast
CDS/J/A+A/470/505/t...	AMIGA IV. Neighbours around CIG galaxies (Verley+,...	http://tapvizier.u-strast
CDS/J/A+A/472/121/t...	AMIGA V. Isolation parameters (Verley+, 2007)	http://tapvizier.u-strast
CDS/J/A+A/485/475/t...	AMIGA. VI. Radio fluxes of the isolated galaxies (Leo...	http://tapvizier.u-strast
CDS/J/A+A/486/73/ta...	Galaxies from AMIGA sample listed as active in the li...	http://tapvizier.u-strast
CDS/J/A+A/486/73/ta...	Radio-excess galaxies found using the radio-FIR cor...	http://tapvizier.u-strast
CDS/J/A+A/486/73/ta...	Radio-excess galaxies in FIRST (table4)	http://tapvizier.u-strast
CDS/J/A+A/486/73/ta...	Classified galaxies using the IRAS colour method (tab...	http://tapvizier.u-strast
CDS/J/A+A/486/73/ta...	Catalogue of AGN-candidates for the total sample (t...	http://tapvizier.u-strast

- Then the user creates a TAP query using the ALADIN GUI.



CDS/J/A+A/436/443/table3

Mode: viz4.J/A...

Construct your query, verify and execute.

Table: J/A+A/436/443/table3

Select: ☑ All

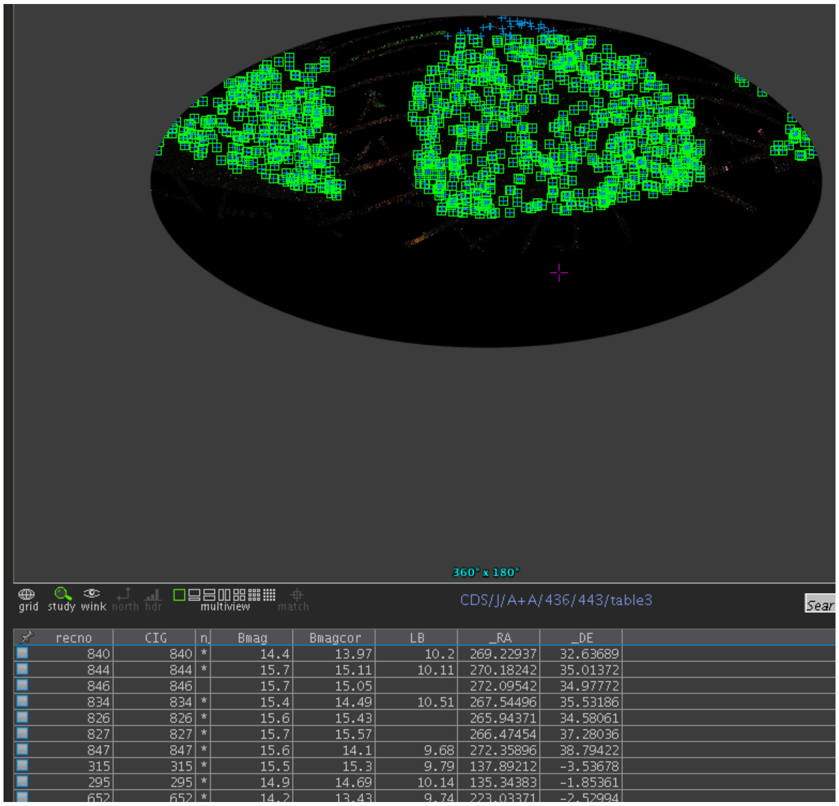
Constraints:

Max rows: 9999

recno
CIG
n_CIG
Bmag
Bmagcor
LB
_RA

```
SELECT TOP 9999 * FROM "J/A+A/436/443/table3"
```

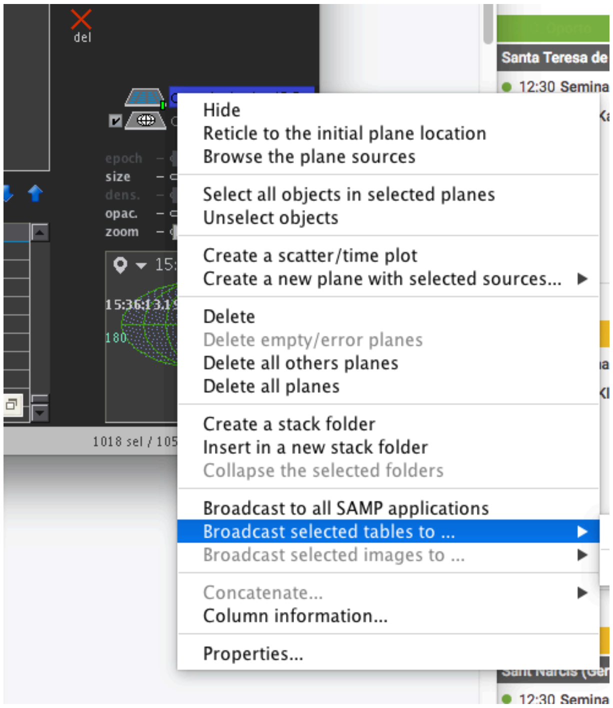
- TAP query results are loaded in ALADIN



CDS/J/A+A/436/443/table3

	recno	CIG	n_	Bmag	Bmagcor	LB	_RA	_DE
▢	840	840	*	14.4	13.97	10.2	269.22937	32.63689
▢	844	844	*	15.7	15.11	10.11	270.18242	35.01372
▢	846	846		15.7	15.05		272.09542	34.97772
▢	834	834	*	15.4	14.49	10.51	267.54496	35.53186
▢	826	826	*	15.6	15.43		265.94371	34.58061
▢	827	827	*	15.7	15.57		266.47454	37.28036
▢	847	847	*	15.6	14.1	9.68	272.35896	38.79422
▢	315	315	*	15.5	15.3	9.79	137.89212	-3.53678
▢	295	295	*	14.9	14.69	10.14	135.34383	-1.85361
▢	652	652	*	14.2	13.43	9.74	223.03371	-2.52994

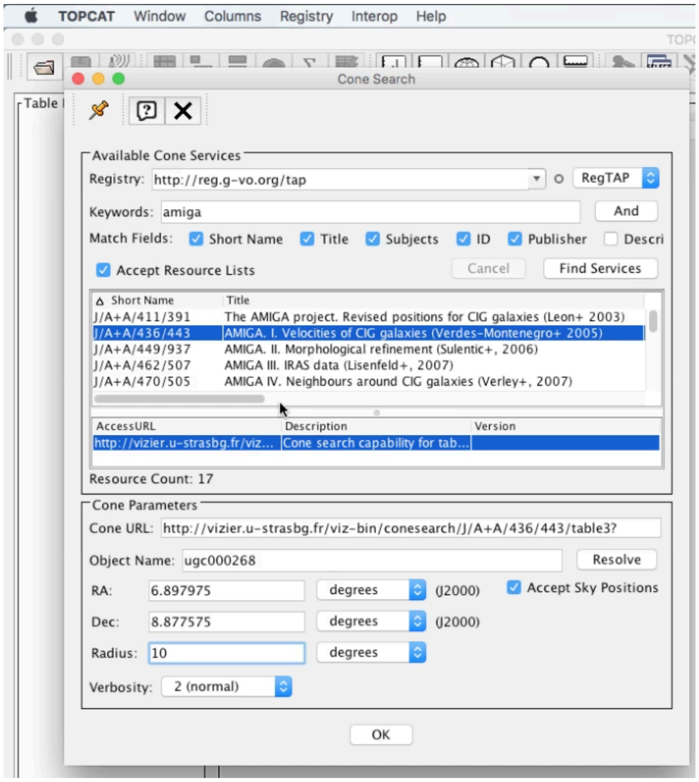
- Once the data results is loaded in ALADIN the user can e.g. load the SLOAN DR8 to better identify the data he/she needs
- Once she/he has selected the required data, select the ALADIN option "Broadcast selected tables to" > ESAP platform.


del
epoch
size
dens.
opac.
zoom
15:36:13.1
180
1018 sel / 105
Hide
Reticle to the initial plane location
Browse the plane sources
Select all objects in selected planes
Unselect objects
Create a scatter/time plot
Create a new plane with selected sources...
Delete
Delete empty/error planes
Delete all others planes
Delete all planes
Create a stack folder
Insert in a new stack folder
Collapse the selected folders
Broadcast to all SAMP applications
Broadcast selected tables to ...
Broadcast selected images to ...
Concatenate...
Column information...
Properties...
Santa Teresa de
12:30 Semina
12:30 Semina

- The VO Table resulting from the TAP query is sent to ESAP via SAMP protocol
- The VO Table is shown in the ESAP Front End and then the user presses a button to start the staging process to transfer the data from the catalogs to the data lake.

SAMP Use Case 2: Cone Search services and Topcat

- The user opens TOPCAT and from the ConeSearch GUI search for a coneSearch service in the VO Registry. Then he/she provides the input values for RA, DEC and Radius



Available Cone Services

Registry: http://reg.g-vo.org/tap ○ RegTAP

Keywords: amiga   And

Match Fields: ☑ Short Name ☑ Title ☑ Subjects ☑ ID ☑ Publisher □ Descri

☑ Accept Resource Lists   Cancel   Find Services

Short Name	Title
J/A+A/411/391	The AMIGA project. Revised positions for CIG galaxies (Leon+ 2003)
J/A+A/436/443	AMIGA. I. Velocities of CIG galaxies (Verdes-Montenegro+ 2005)
J/A+A/449/937	AMIGA. II. Morphological refinement (Sulentic+, 2006)
J/A+A/462/507	AMIGA III. IRAS data (Lisenfeld+, 2007)
J/A+A/470/505	AMIGA IV. Neighbours around CIG galaxies (Verley+, 2007)

AccessURL	Description	Version
http://vizier.u-strasbg.fr/viz...	Cone search capability for tab...	

Resource Count: 17

Cone Parameters

Cone URL: http://vizier.u-strasbg.fr/viz-bin/conesearch/J/A+A/436/443/table3?

Object Name: ugc000268   Resolve

RA: 6.897975 degrees (J2000)   ☑ Accept Sky Positions

Dec: 8.877575 degrees (J2000)

Radius: 10 degrees

Verbosity: 2 (normal)

- The VOTable resulting from the ConeSearch query is loaded in TOPCAT

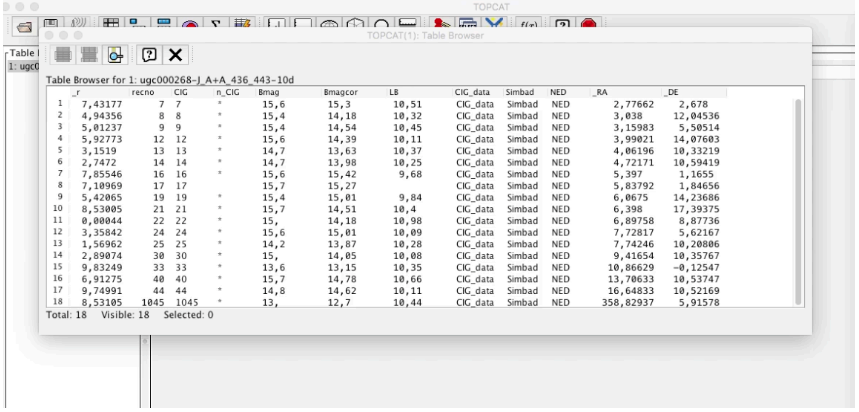


Table Browser for 1: ugc000268-J_A+A_436_443-10d

	_r	recno	CIG	n_CIG	Bmag	Bmagcor	LB	CIG_data	Simbad	NED	_RA	_DE
1	7,43177	7	7	*	15,6	15,3	10,51	CIG_data	Simbad	NED	2,77662	2,678
2	4,94356	8	8	*	15,4	14,18	10,32	CIG_data	Simbad	NED	3,038	12,04536
3	5,01237	9	9	*	15,4	14,54	10,45	CIG_data	Simbad	NED	3,15983	5,50514
4	5,92773	12	12	*	15,6	14,39	10,11	CIG_data	Simbad	NED	3,99021	14,07603
5	3,1519	13	13	*	14,7	13,63	10,37	CIG_data	Simbad	NED	4,06196	10,33219
6	2,7472	14	14	*	14,7	13,98	10,25	CIG_data	Simbad	NED	4,72171	10,59419
7	7,85546	16	16	*	15,6	15,42	9,68	CIG_data	Simbad	NED	5,397	1,1655
8	7,10969	17	17		15,7	15,27		CIG_data	Simbad	NED	5,83792	1,84656
9	5,42065	19	19	*	15,4	15,01	9,84	CIG_data	Simbad	NED	6,0675	14,23686
10	8,53005	21	21	*	15,7	14,51	10,4	CIG_data	Simbad	NED	6,398	17,39375
11	0,00044	22	22	*	15,	14,18	10,98	CIG_data	Simbad	NED	6,89758	8,87736
12	3,35842	24	24	*	15,6	15,01	10,09	CIG_data	Simbad	NED	7,72817	5,62167
13	1,56962	25	25	*	14,2	13,87	10,28	CIG_data	Simbad	NED	7,74246	10,20806
14	2,89074	30	30	*	15,	14,05	10,08	CIG_data	Simbad	NED	9,41654	10,35767
15	9,83249	33	33	*	13,6	13,15	10,35	CIG_data	Simbad	NED	10,86629	-0,12547
16	6,91275	40	40	*	15,7	14,78	10,66	CIG_data	Simbad	NED	13,70633	10,53747
17	9,74991	44	44	*	14,8	14,62	10,11	CIG_data	Simbad	NED	16,64833	10,52169
18	8,53105	1045	1045	*	13,	12,7	10,44	CIG_data	Simbad	NED	358,82937	5,91578

Total: 18   Visible: 18   Selected: 0

- Then, the user sends this table to the ESAP Front End, by the TOCAT functionality "Broadcast table" (or "Send table to")

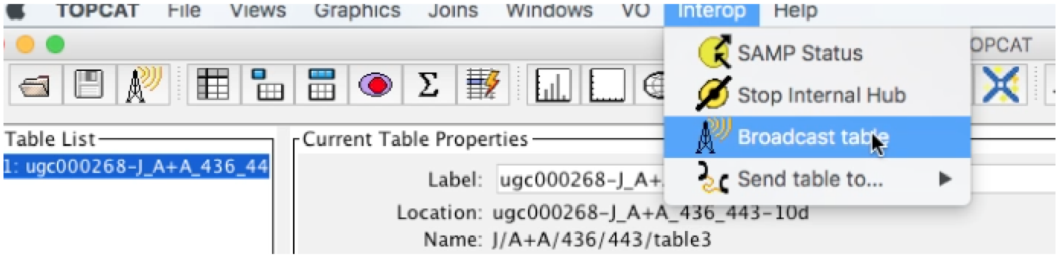


Table List

1: ugc000268-J_A+A_436_44

Current Table Properties

Label: ugc000268-J_A+

Location: ugc000268-J_A+A_436_443-10d

Name: J/A+A/436/443/table3

-

About SAMP and HTTPS

Although SAMP’s Web Profile works well with HTTP-based web applications, it cannot be made to work for web applications hosted on servers using the HTTPS protocol.

This problem is fully explained here:

- https://wiki.ivoa.net/twiki/bin/view/IVOA/WebSampHttps
- https://arxiv.org/pdf/1912.00917.pdf
- https://wiki.ivoa.net/internal/IVOA/InterOpOct2019Apps/tlsamp.pdf
- https://github.com/astrojs/sampjs

Obscore Search

Several libraries including PyVO allow you to find TAP services that support the Obscore data model.

Obscore lets people publish observational datasets through TAP tables.
For example with PyVO this can be done using:

```
  vo.regsearch(datamodel="obscore")
```

Obscore uses generic metadata so that any Obscore service can be used in the same way.
Here is an example use case that uses the Obscore protocol:

Show me a list of all data which satisfies:

I. DataType=TimeSeries
II. RA includes 16.00 hours
III. DEC includes +41.00
IV. Time resolution better than 1 minute
V. Time interval (start of series to end of series) > 1 week
VI. Observation data before June 10, 2008
VII. Observation data after June 10, 2007

Query

```
SELECT  TOP 100  ivoa.ObsCore.access_url,ivoa.ObsCore.obs_id
FROM ivoa.ObsCore
WHERE          (CONTAINS(POINT('ICRS', ivoa.ObsCore.s_ra, ivoa.ObsCore.s_dec),
CIRCLE('ICRS', 10.684667, +41.268750, 0.016666666666666666)) = 1)
          AND ivoa.ObsCore.dataproduct_type = 'timeseries'
          AND ivoa.ObsCore.t_resolution > 60
```

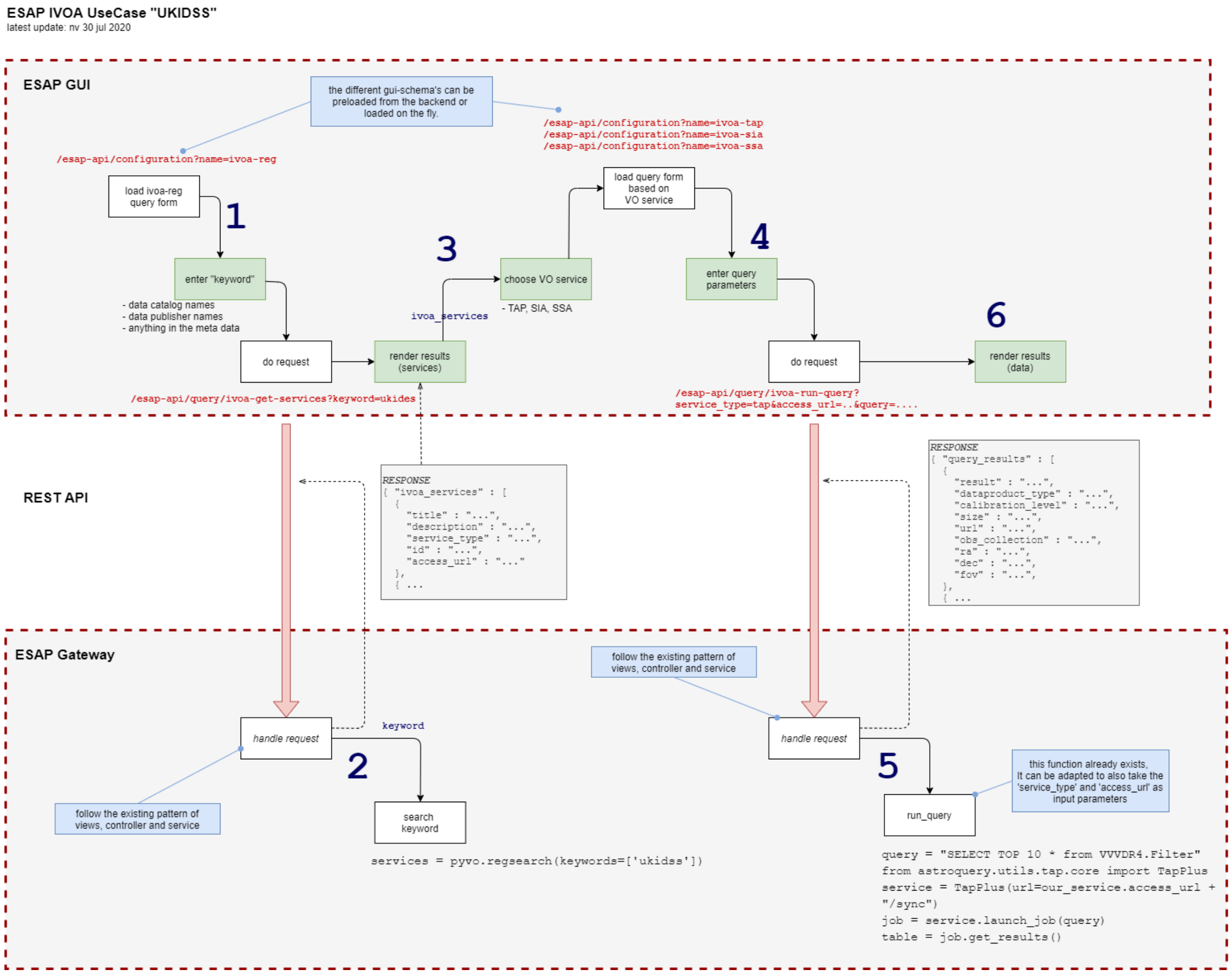
```
        AND ivoa.ObsCore.t_min BETWEEN 54261 AND 54627
```

An additional user data flow diagram could be used for the Obsore protocol, where a search returns a list of services that support the Obscore protocol, and then the user provides a set of parameters which generate a query on every service in the list. The user can then select from the result what they want to checkout.

more...

User Data Flow & UKIDSS VO Example

NV: This is a model and design for the current implementation of ESAP based on what Stelios wrote. There is one main difference; the description had not taken into account that the workflow is handled by the GUI, the business logic by the ESAP Gateway and that all communication is through a REST API.


ESAP IVOA UseCase "UKIDSS"
latest update: nv 30 jul 2020
ESAP GUI
the different gui-schema's can be preloaded from the backend or loaded on the fly.
/esap-api/configuration?name=ivoa-reg
/esap-api/configuration?name=ivoa-tap
/esap-api/configuration?name=ivoa-sia
/esap-api/configuration?name=ivoa-ssa
load ivoa-reg query form
1
enter "keyword"
- data catalog names
- data publisher names
- anything in the meta data
do request
/esap-api/query/ivoa-get-services?keyword=ukides
render results (services)
ivoa_services
3
choose VO service
- TAP, SIA, SSA
load query form based on VO service
4
enter query parameters
do request
/esap-api/query/ivoa-run-query?service_type=tap&access_url=..&query=....
6
render results (data)
REST API
ESAP Gateway
handle request
keyword
2
search keyword
follow the existing pattern of views, controller and service
follow the existing pattern of views, controller and service
handle request
5
run_query
this function already exists, It can be adapted to also take the 'service_type' and 'access_url' as input parameters

```
RESPONSE
{ "ivoa_services" : [
  {
    "title" : "...",
    "description" : "...",
    "service_type" : "...",
    "id" : "...",
    "access_url" : "..."
  },
  { ...
```

```
RESPONSE
{ "query_results" : [
  {
    "result" : "...",
    "dataproduct_type" : "...",
    "calibration_level" : "...",
    "size" : "...",
    "url" : "...",
    "obs_collection" : "...",
    "ra" : "...",
    "dec" : "...",
    "fov" : "...",
  },
  { ...
```

```
services = pyvo.regsearch(keywords=['ukidss'])
```

```
query = "SELECT TOP 10 * from VVVDR4.Filter"
from astroquery.utils.tap.core import TapPlus
service = TapPlus(url=our_service.access_url +
"/sync")
job = service.launch_job(query)
table = job.get_results()
```

GUI page 1: Enter Keyword (input)

Users arrive at the Search page, where they are given a "keyword" textfield (Google-like), where they type a keyword to search for. This keyword may refer to Data Catalog names, Data Publisher names, or anything else in the registered description or metadata of the service.

Once they click the search button, the ESAP platform would call a service which could use PyVO to search for the given keyword:

```
    services = pyvo.regsearch(keywords=['ukidss'])
```

There is an alternative to this simple keyword based search, which is to allow users to create a more advanced search, where they are also able to define the type of the service that they want to search for. This could be in the form of a dropdown list populated with the most common service types (TAP (tables), SIA (images), (SCS) Cone search, SSA (Spectra)). If this is provided the registry search would add an extra parameter as follows:

```
    services = pyvo.regsearch(keywords=['ukidss'], servicetype='tap')
```

Once this search is done the user is provided (probably in a new page) with the list of results from this search.

The info for each result that show for each user can be as detailed as we want, but the results from the registry search above would return the following fields:

ivoid
cap_index
intf_index
intf_type
intf_role
std_version
query_type
result_type
wsdl_url
url_use
access_url
mirror_url
authenticated_only
security_method_id
*ivoid_*
*cap_index_*
cap_type
cap_description
standard_id
ivoid__
res_type
created
short_name
res_title
updated
content_level
res_description
reference_url
creator_seq
content_type
source_format
source_value
res_version
region_of_regard
waveband
rights
rights_uri
harvested_from

Out of this list *res_title, res_description, content_type, standard_id, access_url* are probably the most important for the user to decide which to use, as well as for us to gather the necessary metadata to find out how to get data from that service.

NV: *"The fields (in the json response) will have to have 'standard esap names' that the GUI will recognize. Like "title, description, type, id, url". These 'standard' names can be translated by specific 'parameter_mapping' for this catalog".*
See:
https://git.astron.nl/astron-sdc/esap-api-gateway/-/wikis/Service-Category:-Query

As an example, we could grab the relevant information, and show it as a list to the user as such:

NV: *"This cannot really work this way. Because communication with the user is through the frontend (GUI). Which is ReactJS, not Python. All communication between frontend and (Django) backend is through http requests (a url with parameters) and responses (a json structure containing the desired information).*
So the pyvo functionality should be implemented in the backend, but workflow is implemented in the frontend"

```
import collections
class RegResults(object):

    SERVICE_IDS = collections.defaultdict(str,
{"ivo://ivoa.net/std/tap" : "tap", "ivo://ivoa.net/std/sia" : "sia"
})

    def __init__(self, results):
        self.title = results["res_title"]
        self.description = results["res_description"]
        self.content_type = results["content_type"].decode("utf-8")
        self.servicetype =
self.SERVICE_IDS[results["standard_id"].decode("utf-8")]
        self.access_url = results["access_url"].decode("utf-8")

results = [RegResults(svc) for i in results]
```

GUI page 1: Enter Keyword (input) + Results (output) + Select VO Service

```
for i in results:

    # Some formatting here..
    print (i.title)
    print (i.description)
    print (i.content_type)
    print (i.servicetype)
    print (i.access_url)
    print ("")
```

NV: *"The results should be returned to the frontend in the http response in json format"*

A user is then able to select the service they want to use with a click which should then direct them to the next ("Query") page.

From the above list in Python, we could just select the one that they've chosen like:

```
 our_service = results[0]
```

GUI page 2: Enter query parameters for selected VO Service (input)

In the next page, the query parameters will depend on the service type.
Ignoring SIA and SSA for now, let's focus on TAP & Cone Search (SSA)

For the TAP Query UI, at the very basic form, we'd provide the user with a textarea element, which is labeled "query".
Here the user would populate an ADQL query that they want to send to the selected service.

NV: *"This is very VO specific. In ESAP the esap query parameters are now mapped to VO query parameters and combined into such a ADQL query. Manually typing in a ADQL query by the user seems a bit like a step back from that?"*

Here is an example of how to query a TAP service, assuming the user has input a query:

```
query = "SELECT TOP 10 * from VVVDR4.Filter"
from astroquery.utils.tap.core import TapPlus
service = TapPlus(url=our_service.access_url + "/sync")
job = service.launch_job(query)
table = job.get_results()
```

In a more advanced UI we could also provide a way for the user to navigate through the list of schemas, tables & columns, using the TAP_SCHEMA tables, which are available for every TAP service.

This could be done by sending TAP queries (see above) that select from the TAP_SCHEMA Schema.

e.g.

```
query = "select+*+from+TAP_SCHEMA.schemas"
query = "select+*+from+TAP_SCHEMA.tables"
query =
"select+*+from+TAP_SCHEMA.columns+where+table_name='II/336/apass9'"
```

In the case of Cone search the UI presents the user with options for ra, dec and radius.

NV: *“Currently a TAP based cone search is already onboard. Does the specific pyvo.conesearch functionality offer something extra?”*

We can then either use PyVO to execute the query:

```
import pyvo
url = "https://irsa.ipac.caltech.edu/SCS?table=fp_psc"
objects = pyvo.conesearch(url, pos=(180, 0), radius = 0.05)
objects.table
objects.fieldnames
objects.getdesc('k_m').description
```

NV: *“The results will need to be written into the json body of the response in the structure that the GUI can understand”*

GUI page 3: Query parameters (input) + Results (output)

Or we can do it manually by generating a Cone Search URL:

NV: *“This would probably not fit in the current model. The query parameters are translated per ‘catalog’ from ESAP specific parameters to catalog specific parameters.*
VO_REG would be 1 (ESAP) catalog), with 1 translation from ESAP to catalog.
ESAP knows ‘ra, dec, fov’ and this is currently translated for the VO (ESAP) catalog to ‘s_ra’, ‘s_dec’, ‘s_fov’ as per obscore standard.

This can be a different translation for the VO_REG catalog, this could indeed be 'RA, DEC, SR' as in the example. But if it would have to be different for every underlying service then that would be hard to model."

```
curl -o out.tbl "https://irsa.ipac.caltech.edu/SCS?table=fp_psc&RA=180.0&DEC=0.0&SR=0.05&format=ipac_table"
```

NOTE: The above does not describe the ObsCore use case, where we may have a different flow, for example generating a query around a certain ra, dec & sending it to many services.

Tools:

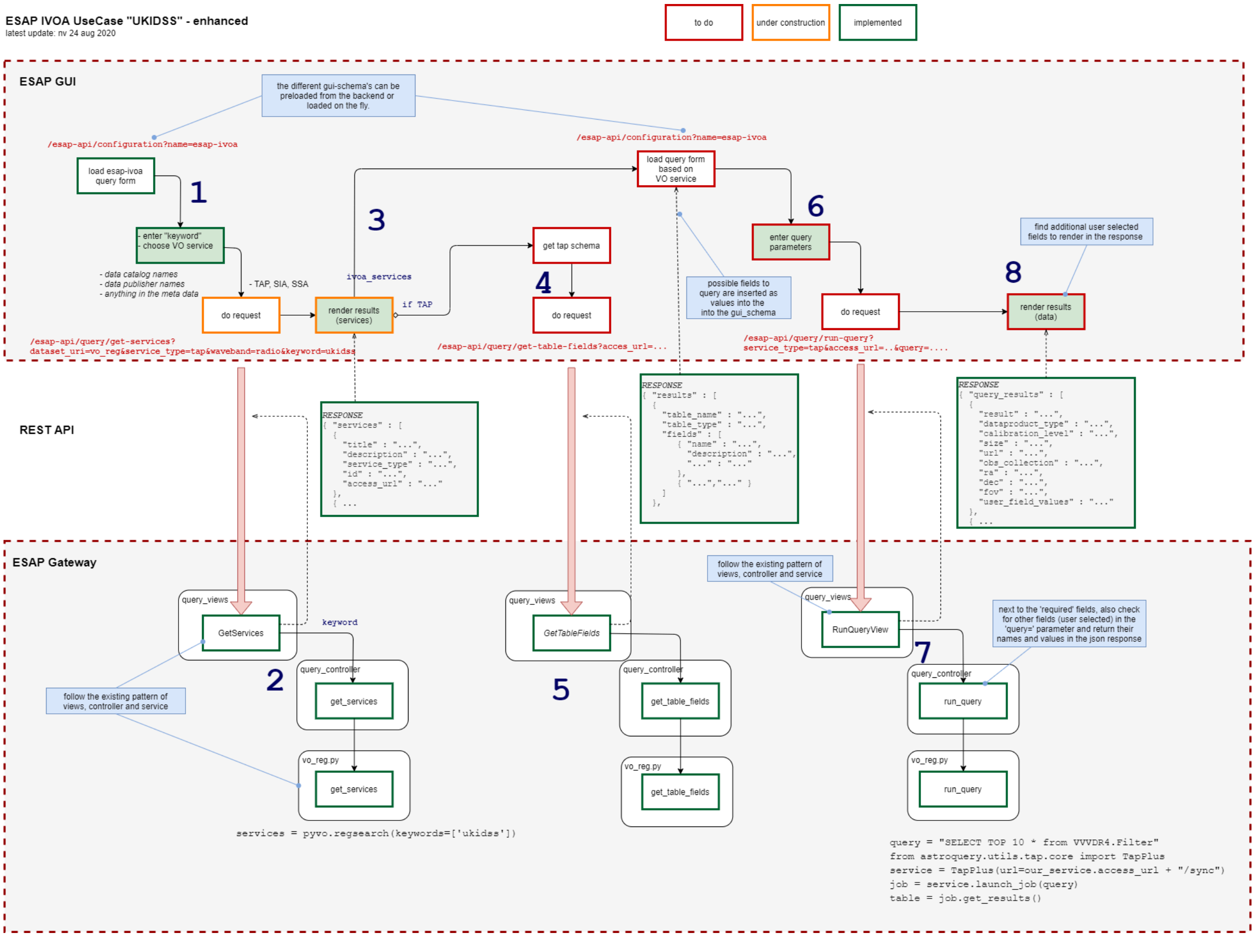
http://saada.unistra.fr/taphandle/?url=http://voparis-tap-astro.obspm.fr:80/tap#

NV: 'Enhanced' version of the UKIDSS use case, with interactive field selection based on the service chosen by the user:

https://app.diagrams.net/#G1zMQADmY3UgI_oJVvAjTwdQIEbyrU1k9d

ESAP IVOA UseCase "UKIDSS" - enhanced

latest update: nv 24 aug 2020


to do
under construction
implemented
ESAP GUI
the different gui-schema's can be preloaded from the backend or loaded on the fly.
load esap-ivoa query form
1
- enter "keyword"
- choose VO service
- data catalog names
- data publisher names
- anything in the meta data
- TAP, SIA, SSA
do request
render results (services)
ivoa_services
3
if TAP
get tap schema
4
do request
load query form based on VO service
possible fields to query are inserted as values into the into the gui_schema
6
enter query parameters
do request
8
render results (data)
find additional user selected fields to render in the response
REST API
ESAP Gateway
query_views
GetServices
keyword
2
query_controller
get_services
vo_reg.py
get_services
follow the existing pattern of views, controller and service
query_views
GetTableFields
5
query_controller
get_table_fields
vo_reg.py
get_table_fields
follow the existing pattern of views, controller and service
query_views
RunQueryView
7
next to the 'required' fields, also check for other fields (user selected) in the 'query=' parameter and return their names and values in the json response
query_controller
run_query
vo_reg.py
run_query

```
/esap-api/configuration?name=esap-ivoa
```

```
/esap-api/query/get-services?
dataset_uri=vo_reg&service_type=tap&waveband=radio&keyword=ukidss
```

```
/esap-api/query/get-table-fields?acces_url=...
```

```
/esap-api/query/run-query?
service_type=tap&access_url=..&query=....
```

```
RESPONSE
{ "services" : [
  {
    "title" : "...",
    "description" : "...",
    "service_type" : "...",
    "id" : "...",
    "access_url" : "..."
  },
  { ...
```

```
RESPONSE
{ "results" : [
  {
    "table_name" : "...",
    "table_type" : "...",
    "fields" : [
       { "name" : "...",
         "description" : "...",
         "..." : "..."
       },
       { "...","..." }
    ]
  },
```

```
RESPONSE
{ "query_results" : [
  {
    "result" : "...",
    "dataproduct_type" : "...",
    "calibration_level" : "...",
    "size" : "...",
    "url" : "...",
    "obs_collection" : "...",
    "ra" : "...",
    "dec" : "...",
    "fov" : "...",
    "user_field_values" : "..."
  },
  { ...
```

```
services = pyvo.regsearch(keywords=['ukidss'])
```

```
query = "SELECT TOP 10 * from VVVDR4.Filter"
from astroquery.utils.tap.core import TapPlus
service = TapPlus(url=our_service.access_url + "/sync")
job = service.launch_job(query)
table = job.get_results()
```