# Faciliter les contributions des astronomes amateurs grâce à l'interopérabilité avec les outils de l'Observatoire Virtuel

WIVONA: Un Projet Pro/Am soutenu par l'Observatoire de Paris-PSL

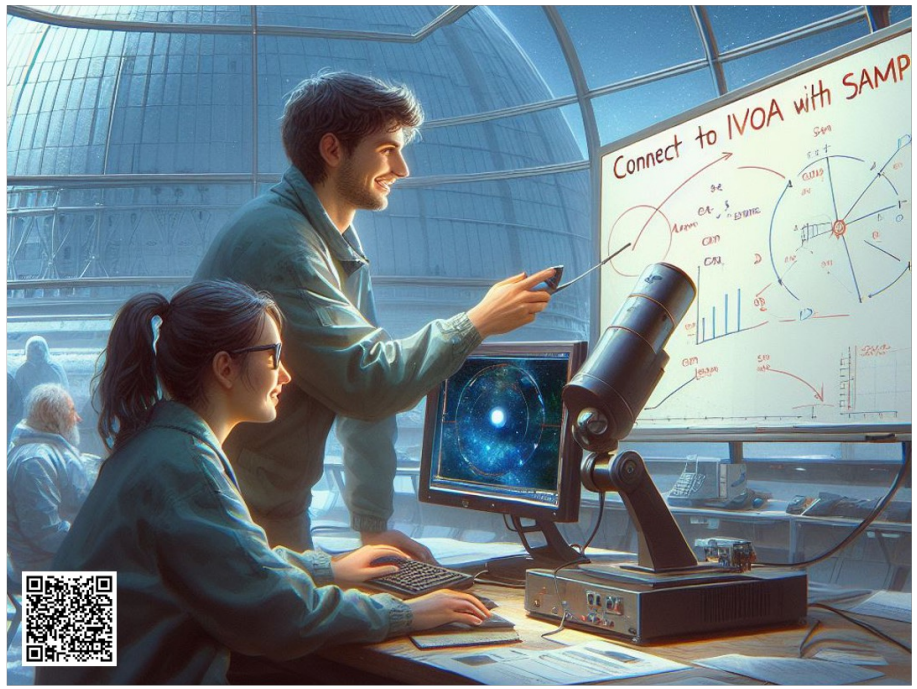
https://gemini.obspm.fr/les-nouveaux-clients-de-lobservatoire-virtuel/
https://www.observatoiredeparis.psl.eu/api-collaborations-proam.html

Renaud Savalle - PADC/Observatoire de Paris-PSL
renaud.savalle@obspm.fr

# WIVONA

We Implement
Virtual Observatory
Needs of Astrams

*A **Pro/Am** collaboration for the Observers Community*




- PI: **Jean-Paul GODARD**, Astronome amateur (Dev PRISM: SAMP, Astro-Colibri)
- **Renaud SAVALLE**, PADC/ Observatoire de Paris, Ingénieur de recherche CNRS, (Dev SharpCap: SAMP, Scripts Python)
- **Cyril CAVADORE**, ALCOR SYSTEM, PhD (Dev PRISM)
- **David VALLS-GABAUD**, LERMA/ Observatoire de Paris, Directeur de Recherche CNRS

# Plan

1. L’Observatoire Virtuel (OV) et l’IVOA
2. L’échange des données entre applications avec SAMP
3. Le suivi des Alertes des Phénomènes Transitoires avec Astro-Colibri et PRISM
4. L’observations des transits d’Exoplanètes avec Exoclock et PRISM
5. L’accès aux données de de l’OV avec Python; cas d’utilisation
6. Etat du projet

# L'Observatoire Virtuel

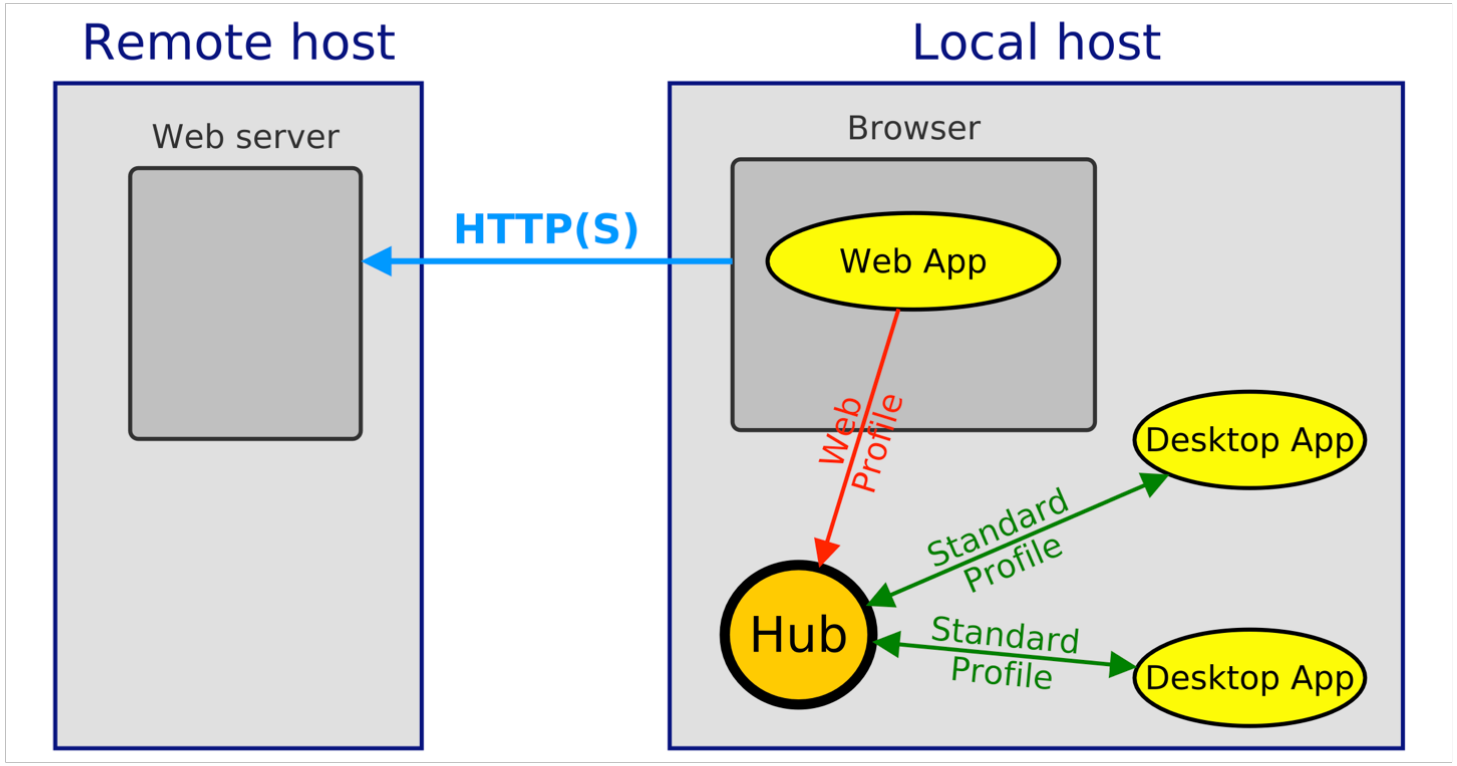
- L'OV n'est pas: un (ensemble de) sites - qui disparaitront - ni un logiciel - qui deviendra obsolète
- L'OV est un effort a long terme
  - porté par operateurs de ~50 centres de données + développeurs,
  - fédéré par **l'International Virtual Observatory Alliance (IVOA)**
- Contient les **données** astronomiques des archives de la plupart des observatoires professionnels
  - Catalogues avec caractéristiques des objets (position, physique) et leur bibliographie
  - Images
  - Spectres
  - Cubes Spectraux
  - Séries temporelles
  - Alertes (Phénomènes transitoires)
- Des **services** (ex: résolution de nom) et des **protocoles/standards** d'accès normalisés aux données
- Des **applications** clientes échangeant entre elles via le protocole standard **SAMP: Simple Application Messaging Protocol** (ex: Aladin, TOPCAT)
- Un annuaire décentralisé contenant les métadonnées des collections et des services: le Registre

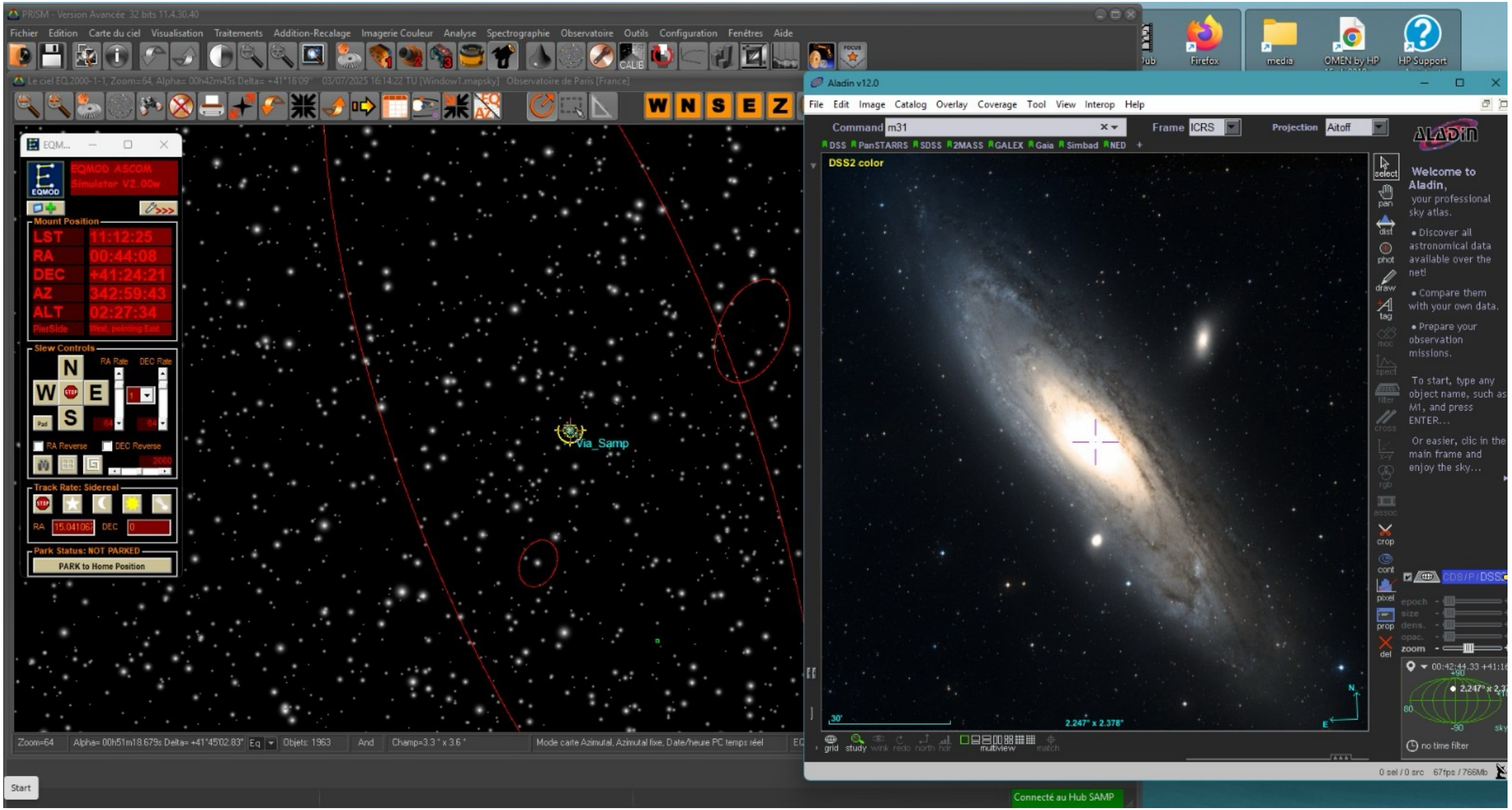
# A few high level IVOA Standards

- Standards for designing **applications**
  - **VOTable** the format for exchanging tabular data including rich metadata (coosys, timesys, ucd, utype, VOunits, datalink…) - used by many other standards
  - **HiPS** (Hierarchical Progressive Survery) - tailored for large image data volumes
- Standards describing *web services* to **discover** and **transport** data*:*
  - **VOEvent** for alerts
  - **Simple Cone Search** - spatial and temporal search for catalogs
  - **Simple Image Access**
  - **Simple Spectral Access**
  - **MOC** Multi-Order Coverage map - spatial and temporal indexing for large data volumes
  - **TAP + ADQL** — Table Access Protocol & astronomical data query language
  - **ObsCore & ObsTAP** — description of observations, and upcoming extensions
- **Registry** Standards – to define how to register and discover resources
- Planning of observations: (under dev.)
  - **ObjVisSAP** — visibility of objects
  - **ObsLocTAP** — facilitate coordination of observations from different facilities

# SAMP

- **Simple Application Messenging Protocol**
- Un autre protocole “simple”:




PRISM - Version Avancée 32 bits 11.4.30.40
Fichier Edition Carte du ciel Visualisation Traitements Addition-Recalage Imagerie Couleur Analyse Spectrographie Observatoire Outils Configuration Fenêtres Aide
Aladin v12.0
File Edit Image Catalog Overlay Coverage Tool View Interop Help
Command m31
Frame ICRS
Projection Aitoff
DSS2 color
Welcome to Aladin,
your professional sky atlas.
Mount Position
LST 11:12:25
RA 00:44:08
DEC +41:24:21
AZ 342:59:43
ALT 02:27:34
Slew Controls
Track Rate: Sidereal
Park Status: NOT PARKED
PARK to Home Position
Via_Samp
Connecté au Hub SAMP

# Observer Les Phénomènes Transitoires

- Des phénomènes fugaces (SNe, GRBs…)
- Détectés par des observatoires automatiques
- Les observatoires génèrent des alertes, les observateurs recherchent des contre-parties optiques.
- Besoin: fusion de données multi-messagers
  - Spectre électromagnétique
  - Ondes Gravitationnelles (OG)
  - Neutrinos

# Un Moniteur d'Alertes

- Extérieur à PRISM
- Ecoute les brokers (H24,7/7) et stocke les alertes
- Fournit des filtres (type, mag) pour l'utilisateur enregistré
- Restitue des "VOevents" complets
- Moniteur retenu: **Astro-Colibri**
- Communication avec PRISM via l'API HTTP/JSON

# Architecture Astro-Colibri + PRISM

Observatoires

Brokers

SWIFT

Alertes

Aggregateur
Astro-Colibri

'Tagueurs':
RAPAS
KNC
BH_TOM

BD

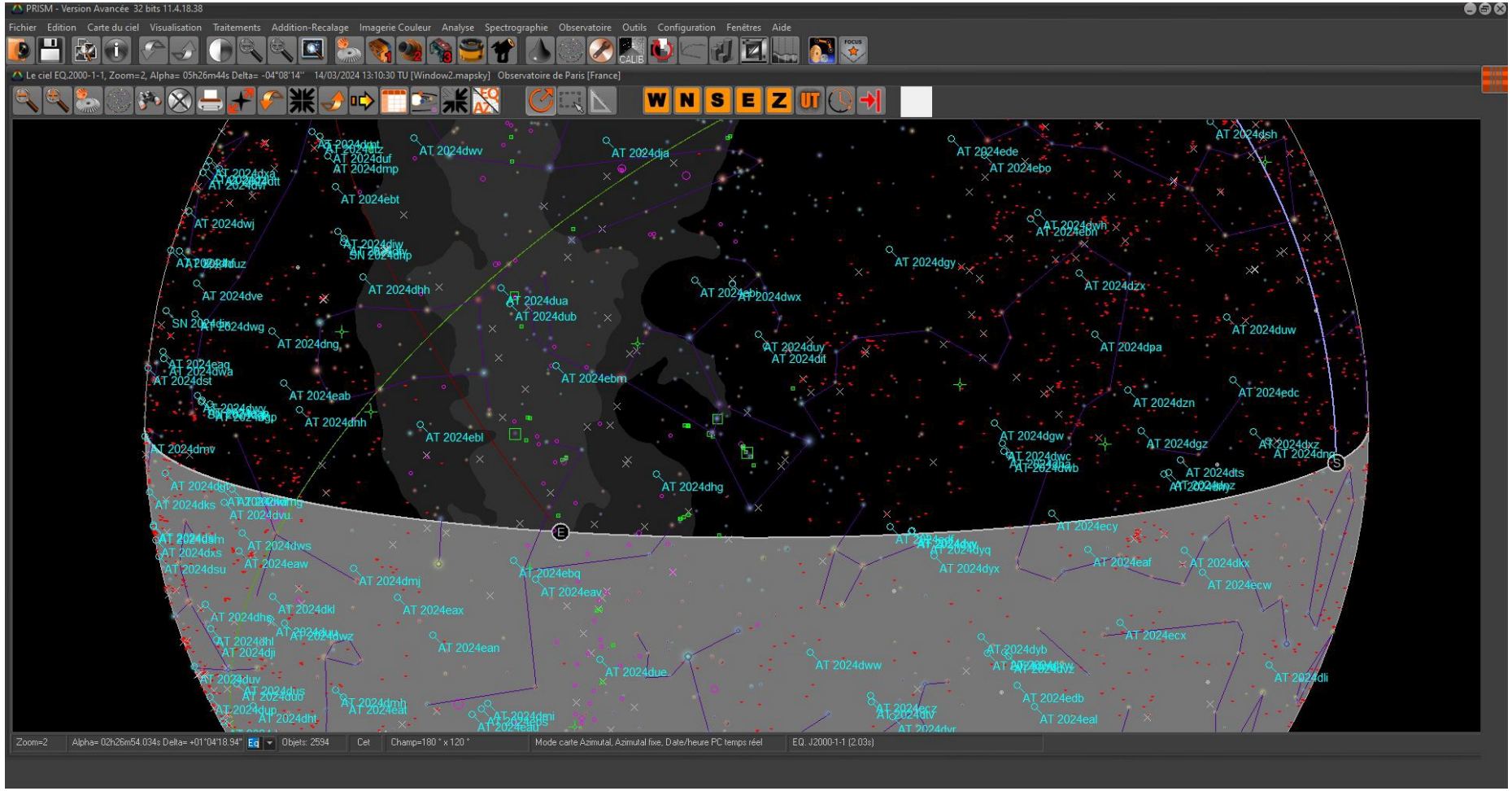
HTTP/JSON
API

VOEvents

Applications
Clientes

PRISM

# Les Phénomènes Transitoires dans PRISM



# La base de données des transits Exoclock

- Les transits de plusieurs centaines d’exoplanètes sont accessibles aux amateurs
- Exoclock est accessible par web service au format JSON
- Le service fournit les instants des transits du jour en BJD
- Nécessité de calculer les éphémerides locaux en UTC

**The ephemeris**

For every transiting planet we have an ephemeris – an equation that is determined by current observations and connects the mid-times of different transit events in the past and in the future. This equation looks like this:

$$T = T_0 + N \times P$$

where:

**$T_0$** is the zero-epoch transit mid-time –
the mid-time of a transit that we use as reference (it is up to us to choose which transit this will be)

**P** is the period –
the difference in time between two transit mid-times

**T** is the transit mid-time –
refers to the mid-time a transit

**N** is the transit epoch –
the number of periods that have elapsed since the reference transit

From the above parameters, $T_0$ and P are determined from existing transit observations. For example, the current ephemeris of WASP-10b is:

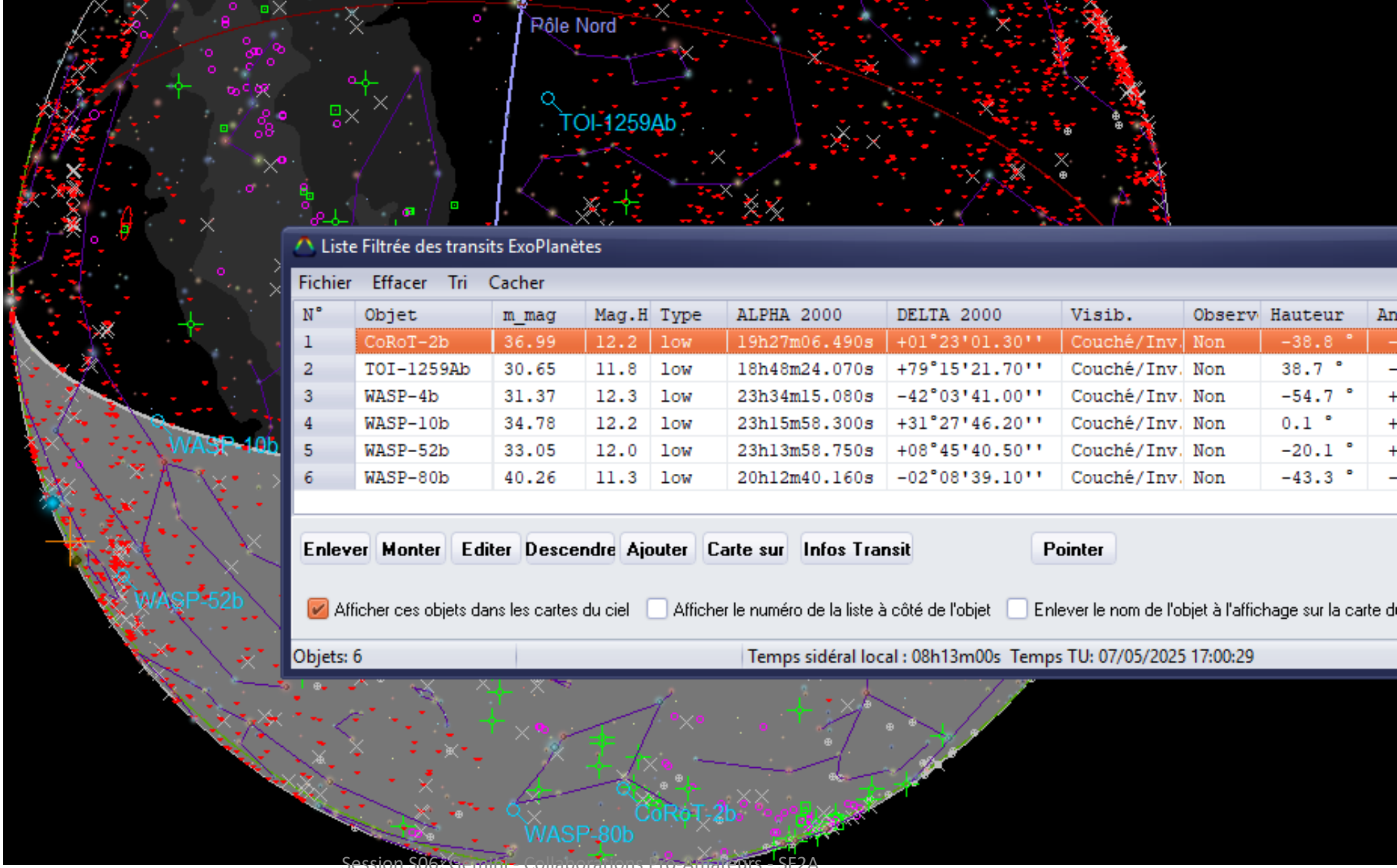
$$T = 2454664.038040(6) + N \times 3.0927295(3)\ BJD_{TDB}$$

This means that one transit – the one that we take as reference – happened at 2454664.03804 $BJD_{TDB}$, corresponding to the 16th of July 2008, 12:51:14.4 UTC, and other transits happen every 3.0927295 days.

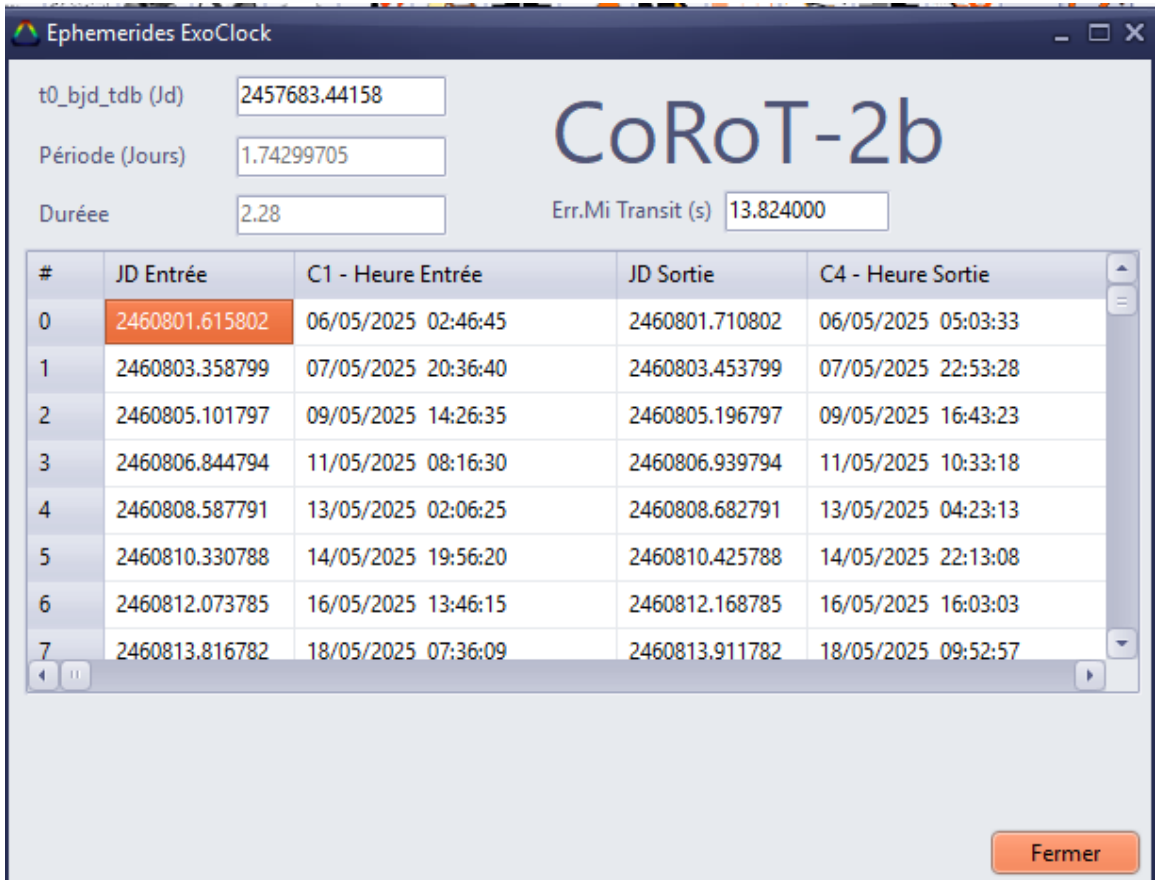
# PRISM et la Base des transit EXOCLOCK

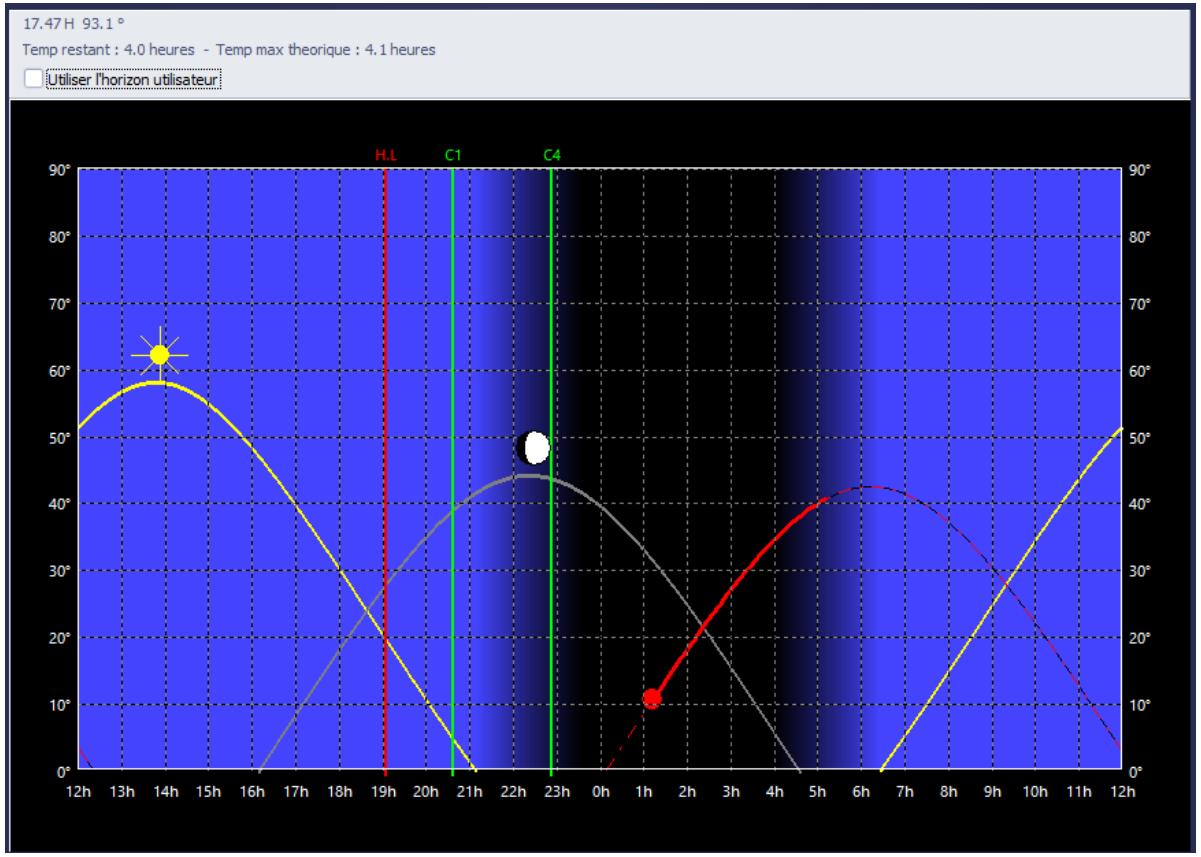
- Prism filtre et charge la base Exoclock
  - En récuperant les événements prévus sur une fenêtre en jours
  - En tenant compte
    - du diamètre du télescope
    - des capacités de l'observateur (Delta Mag Min)

# Etoiles Hôtes



# Transits et visibilité locale



# Accès a l'OV dans PRISM avec Python

- **Console Python pour PRISM**
  - Intégrée au logiciel avec Python4Delphi
  - **Accès aux données PRISM de session (site, instrument, pointage,…)**
- **Développer des scripts adhoc:**
  - Charger, sauvegarder des scripts Python
  - Les échanger, les éditer, les paramétrer et les exécuter
- **Utiliser les protocoles de l'OV via les modules Python**:
  - astropy , pyvo, astroquery
- NB: il sera bientôt possible d'utiliser Python pour **échanger des données avec la liste des objets à observer PRISM, télécharger pendant la session d'observation les étoiles Gaia du champ observé et automatiser les observations.**

# Exemple: Python pour l'accès a l'OV

- **Cas d'utilisation**

  - Localiser les resources de l'OV contenant des collections de données liées aux GRBs en utilisant le terme "gamma-ray-bursts" de l'Unified Astronomy Thesaurus (UAT) de l'**UAI**

  - Interroger un service "Simple Cone Search" d'une de ces ressources pour chercher des objects dans une zone du ciel

  - Sauvegarder les résultats aux formats CSV et VOTable

  - Charger et examiner le résultats dans TOPCAT

- **Démo avec PRISM**

  - Console Python

- **Packages Python utilisés**

  - **pyvo (v1.7): accès OV**
  - astropy: librairie pour l'astronomie
  - astroquery: requetes de services web astronomiques

# Démo

https://www.youtube.com/watch?v=y3lxmds_lX4&t=592s

## Découverte des données avec le Registre

```
# Import VO packages
try:
    import pyvo, astropy, astroquery
except Exception as e:
    print('ERROR Importing VO modules: ', str(e))
else:
    print('import OK')

print('Using pyvo version ', pyvo.version.version)
print('Using astropy version ', astropy.version.version)
print('Using astroquery version ', astroquery.version.version)

import warnings
from astropy.utils.exceptions import AstropyWarning

warnings.simplefilter('ignore', category=AstropyWarning)

from pyvo import registry

# UAT term to search for
term = 'gamma-ray-bursts'

# Search resources about UAT term
res = registry.search(registry.UAT(term))

print(f'{len(res)} resources found about {term}')

# Show resources found
for i, r in enumerate(res, start=0):
    print(f'{i} {r.short_name} {r.res_title}')
```

## Interrogation d'un service

```
# Type of service to look for
type = 'conesearch'

# coordinates for conesearch
coords = (179.1, 65.2)

# Select resources to query
res1 = [res[0]]  # , res[1]]

# Query found resources for services of a certain type
for r in res1:
    try:

        svc = r.get_service(service_type=type)
        print(f'Querying {svc}')

        results = svc.search(pos=coords, sr=10.0)
        print(f'Received {len(results)} rows')
        table = results.to_table()
        print(table)
    except ValueError:
        # print('no matching interface for {svc}')
        pass
```

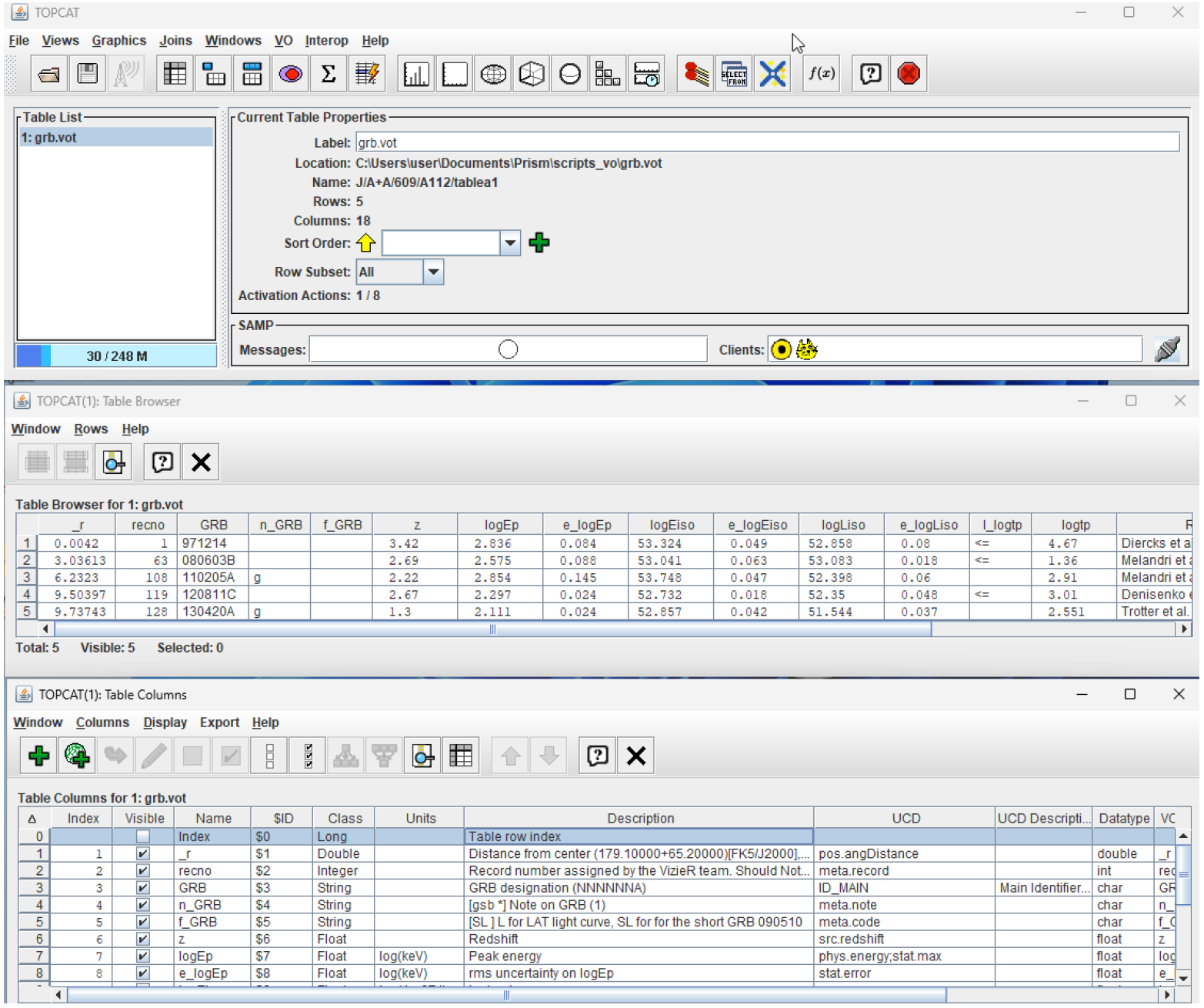
## Sauvegarde du résultat

```
# Save results to CSV file
from astropy.io import ascii

data_path = ''
filename_csv = data_path + 'grb.csv'
table.write(filename_csv, format='csv', overwrite=True)
print(f'Saved CSV file to {filename_csv}')

# Save results to VOTable file
from astropy.io.votable import from_table

filename_vot = data_path + 'grb.vot'
vot = from_table(table)
vot.to_xml(filename_vot)
print(f'Saved VOT file to {filename_vot}')
```




| | _r | recno | GRB | n_GRB | f_GRB | z | logEp | e_logEp | logEiso | e_logEiso | logLiso | e_logLiso | l_logtp | logtp | R |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| 1 | 0.0042 | 1 | 971214 | | | 3.42 | 2.836 | 0.084 | 53.324 | 0.049 | 52.858 | 0.08 | <= | 4.67 | Diercks et al |
| 2 | 3.03613 | 63 | 080603B | | | 2.69 | 2.575 | 0.088 | 53.041 | 0.063 | 53.083 | 0.018 | <= | 1.36 | Melandri et a |
| 3 | 6.2323 | 108 | 110205A | g | | 2.22 | 2.854 | 0.145 | 53.748 | 0.047 | 52.398 | 0.06 | | 2.91 | Melandri et a |
| 4 | 9.50397 | 119 | 120811C | | | 2.67 | 2.297 | 0.024 | 52.732 | 0.018 | 52.35 | 0.048 | <= | 3.01 | Denisenko e |
| 5 | 9.73743 | 128 | 130420A | g | | 1.3 | 2.111 | 0.024 | 52.857 | 0.042 | 51.544 | 0.037 | | 2.551 | Trotter et al. |

| Δ | Index | Visible | Name | $ID | Class | Units | Description | UCD | UCD Descripti... | Datatype | VC |
|---|---|---|---|---|---|---|---|---|---|---|---|
| 0 | | ☐ | Index | $0 | Long | | Table row index | | | | |
| 1 | 1 | ☑ | _r | $1 | Double | | Distance from center (179.10000+65.20000)[FK5/J2000],... | pos.angDistance | | double | _r |
| 2 | 2 | ☑ | recno | $2 | Integer | | Record number assigned by the VizieR team. Should Not... | meta.record | | int | rec |
| 3 | 3 | ☑ | GRB | $3 | String | | GRB designation (NNNNNNA) | ID_MAIN | Main Identifier... | char | GR |
| 4 | 4 | ☑ | n_GRB | $4 | String | | [gsb *] Note on GRB (1) | meta.note | | char | n_ |
| 5 | 5 | ☑ | f_GRB | $5 | String | | [SL ] L for LAT light curve, SL for for the short GRB 090510 | meta.code | | char | f_C |
| 6 | 6 | ☑ | z | $6 | Float | | Redshift | src.redshift | | float | z |
| 7 | 7 | ☑ | logEp | $7 | Float | log(keV) | Peak energy | phys.energy;stat.max | | float | log |
| 8 | 8 | ☑ | e_logEp | $8 | Float | log(keV) | rms uncertainty on logEp | stat.error | | float | e_ |

# Etat du Projet WIVONA

- SAMP, Python, Astro-Colibri, Exoclock sont intégrés à PRISM v11
- Les sources Delphi seront accessibles sur GitHub
- Aide, démos et tutoriels disponibles: Tutoriel Python: https://t.ly/ugxOR
- Présentations
  - Ecoles de Photométrie (Marseille 2024, Toulouse 2025)
  - Colloque Astro-Colibri (Orsay, Oct. 2024)
  - Rencontres du Ciel et de l'Espace (Paris, La Villette Nov. 2024)
  - Atelier RAPAS (Paris, Déc. 2024)
  - Colloque ACME (Juin 2025)

# BACKUP SLIDES

# P106

# Discover your astronomical data from Python – simply!

**Renaud Savalle[1], Markus Demleitner[2], Hendrik Heinl[2]**

[1] *DIO/Paris Astronomical Data Centre (PADC), Paris Observatory, UAR2201-CNRS, PSL Research Univ., 5, place Jules Janssen, 92195 Meudon, France*
[2] *Universität Heidelberg, Zentrum für Astronomie, Mönchhofstraße 12-14, 69120 Heidelberg, Germany*

***Contact: renaud.savalle@obspm.fr***

The VO Registry is a set of about 30,000 metadata records of astronomical resources. It is queryable using the powerful RegTAP protocol requiring users to write ADQL. A friendlier interface using that protocol has recently been written a part of the PyVO astropy affiliated package. In this poster, we briefly introduce the standards this is implemented against. The new API can be used by astronomers to discover data based on various constraints, ranging from free text to physical concepts and areas in space, time, and spectrum.

## The Virtual Observatory Registry

The Virtual Observatory Registry [1] implements a set of IVOA standards to query the metadata for VO data collections and services, thus making them findable. Full Searchable Registries are RegTAP [2] instances which expose that metadata in a set of tables queryable with TAP [3] and ADQL [4].

## How to discover astronomical data

The standard programmatic way to discover data in the Registry is to send an ADQL query to a RegTAP instance, using the TAP protocol. This goes far beyond keyword searches; for instance, you can precisely constrain authors, subjects, physics represented in table columns, relations to other resources (such as “Continues” or “IsServedBy”), coverage in space, time, or spectrum, etc. It is also straightforward to restrict the metadata retrieved to what you actually need for the task at hand, which is relevant when a full metadata record can easily be as large as a Megabyte. To keep up with the pace of the growing data in the field, the structure of the Registry schema is evolving. On the one hand, that enables more elaborate and detailed queries, but on the other hand it makes the metadata model more complex, which also impacts the queries.

The downside of this architecture is that common free-text queries (which, of course, this architecture tries to improve upon to some degree) become rather verbose, such as this query for "Gaia lightcurves" (which is not the preferred way to discover this kind of data):

```
SELECT ivoid, res_type, short_name, res_title, res_description, reference_url
FROM rr.resource
NATURAL LEFT OUTER JOIN rr.capability
NATURAL LEFT OUTER JOIN rr.interface
WHERE (ivoid IN (SELECT ivoid FROM rr.resource WHERE 1=ivo_hasword(res_description, 'Gaia
Lightcurves') UNION SELECT ivoid FROM rr.resource WHERE 1=ivo_hasword(res_title, 'Gaia
Lightcurves') UNION SELECT ivoid FROM rr.res_subject WHERE res_subject ILIKE '%Gaia
Lightcurves%'))
GROUP BY ivoid, res_type, short_name, res_title, res_description, reference_url
```

The above query already looks surprisingly complex compared to the idea of "just" searching for a short string. Adding constraints to the query demands even more skills in TAP/ADQL and is far from a reasonable effort the typical VO end-user would want to do, especially compared to how easy all this looks from client applications like Aladin and TOCAT.

## The pyvo.registry API

PyVO [5] is a an affiliated package for the astropy package which helps find and retrieve astronomical data available from archives that support standard IVOA Virtual Observatory service protocols. Since PyVO version 1.4, the module `pyvo.registry` provides a new API to Full Searchable Registries. The principle of that new API is to build discovery queries by stacking constraints such as `Freetext(keywords)`. Using PyVO 1.5, the above query looks like this:

```
from pyvo import registry
resources = registry.search(
    registry.Freetext("Gaia Lightcurves"))
```

Under the hood, an ADQL query is generated from the given set of constraints. That query can be displayed by calling `registry.get_RegTAP_query()` instead of `registry.search()`. This is also a useful way to learn about the RegTAP relational schema.

## Search Constraints

The available constraints are listed below [6]. When they accept several arguments, those are combined disjunctively (“OR”). Multiple constraints (possibly of the same type) are combined conjunctively (”AND”).

- **`pyvo.registry.Freetext`** ( `keywords` ): one or more freetext words, mached in the title, description or subject of the resource.
- **`pyvo.registry.Servicetype`** ( `servicetype` ): constrain to one of tap, ssa, sia, conesearch (or full ivoids for other service types). This is the constraint you want to use for service discovery.
- **`pyvo.registry.UCD`** ( `ucd` ): constrain by one or more UCD patterns; resources match when they serve columns having a matching UCD (e.g., `phot.mag;em.ir.%` for “any infrared magnitude”).
- **`pyvo.registry.Waveband`** ( `waveband` ): one or more terms from the vocabulary at http://www.ivoa.net/rdf/messenger giving the rough spectral location of the resource.
- **`pyvo.registry.Author`** ( `author` ): an author (“creator”). This is a single SQL pattern, and given the sloppy practices in the VO for how to write author names, you should probably generously use wildcards.
- **`pyvo.registry.Datamodel`** ( `datamodel` ): one of obscore, epntap, or regtap: only return TAP services having tables of this kind.
- **`pyvo.registry.Ivoid`** ( `ivoid` ): exactly match a single IVOA identifier (that is, in effect, the primary key in the VO). *RegTAP 1.2 Extension*
- **`pyvo.registry.Spatial`** ( `spatial` ): match resources covering a certain geometry (point, circle, polygon, or MOC). *RegTAP 1.2 Extension*
- **`pyvo.registry.Spectral`** ( `spectral` ): match resources covering a certain part of the spectrum (usually, but not limited to, the electromagnetic spectrum). *RegTAP 1.2 Extension*
- **`pyvo.registry.Temporal`** ( `temporal` ): match resources covering a some point or interval in time. *RegTAP 1.2 Extension*

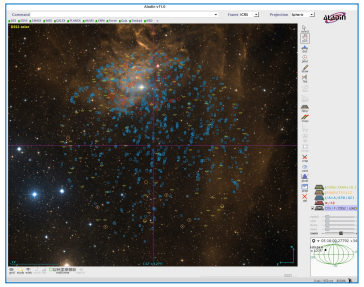
## Example

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

resources = registry.search(
    registry.Freetext("Emission line stars"),
    registry.UCD('phot.mag;em.ir*%'), # smart use of wildcard to match all IR bands
    registry.Spatial(SkyCoord('79d 34d')))

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)
```

In this example, we use `registry.search()` to look for VO resources that publish infrared magnitudes (the UCD [7] 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 [8], we loop over the resources we have discovered. Those that have an SCS [9] endpoint we query and send whatever we get back to Aladin. It would be relatively straightforward to support other data access protocols (e.g., TAP) in this way, but of course the calling sequence would be a bit more complicated in these cases.



## References

**[1] M. Demleitner et al - The virtual observatory registry, Astronomy and Computing, Volumes 7–8, 2014** https://doi.org/10.1016/j.ascom.2014.07.001
**[2] IVOA Registry Relational Schema standard -** https://www.ivoa.net/documents/RegTAP
**[3] IVOA Table Access Protocol standard -** https://www.ivoa.net/documents/TAP/
**[4] IVOA Astronomical Data Query Language standard -** http://www.ivoa.net/documents/ADQL
**[5] PyVO documentation -** https://pyvo.readthedocs.io/en/latest/
**[6] PyVO documentation for pyvo.registry -** https://pyvo.readthedocs.io/en/latest/registry/index.html
**[7] IVOA Unified Content Descriptors standard** https://ivoa.net/documents/latest/UCD.html
**[8] IVOA Simple Application Messaging Protocol standard -** https://ivoa.net/documents/latest/SAMP.html
**[9] IVOA Simple Cone Search standard -** https://ivoa.net/documents/latest/ConeSearch.html

This research has made use of data obtained from or tools provided by Paris Astronomical Data Centre

V1.2