ASTERICS

European Data Provider Forum

and Training Event 2016

Saada Tutorial

Introduction

Saada has been designed to help astronomers to build local archives by the simplest way. A Database created by Saada, a *SaadaDB*, can host multiple collections of heterogeneous data (spectra, catalogs images, spectra, flatfiles, miscellaneous) .
Meta-data extracted from input files (FITS or VOTables) are first stored into the database and then mapped to a common interface allowing consistent queries covering these heterogeneous datasets.
Data collections can be linked to each other with persistent relationships.
A *SaadaDB* comes with a rich Web interface automatically generated and managed.
Data collections can also be exposed through VO services (SCS, SIA, SSA or TAP).
An API is available for those who want to build their own Web interface or to feed another piece of software with data retrieved from the database.
A *SaadaDB* can be managed with a graphical tool or by script.
Although being enable to be operated automatically, the data loader can be configured by hand to extract the proper meta-data.

Software Requirements

- **Saada** works on either **MacOS**, **Windows** or **Linux**, all in 64 bits.
- **Oracle Java JDK** (1.7+) is required.
- **Saada** can run with **PostgresQL, MySQL** or **SQLite**. This last option do not require any software or service installation (recommended for the session).
- The Web interface works with **Apache Tomcat 6+**. Tomcat must have R/W permission in the Saada repository.

- **Windows**: Make both following environment variables pointing on your JDk install (see *Advanced Settings*)
    - PATH
    - JAVA_HOME

Tutorial Steps

This tutorial proposes 11 steps. Some are necessary and some others are optional:

1. Saada installation
2. Database creation
3. Loading catalogues and READMEs
4. Testing the Web interface
5. Loading images
6. Loading spectra
7. Linking catalogs and READMEs
8. Linking images, spectra and sources together
9. Publishing a simple service
10. Publishing a TAP service
11. Accessing the TAP service

The table below shows the dependencies between the different tasks.
To be achieved, each task (green cell) requires all task tagged in yellow on the same row to be completed in the right order.

	1	2	3	4	5	6	7	8	9	10	11
1 Saada installation											
2 Database creation											
3 Loading catalogues and READMEs											
4 Testing the Web interface											
5 Loading images											
6 Loading spectra											
7 Linking catalogues and READMEs											
8 Linking images, spectra and sources together											
9 Publishing simple services											
10 Publishing a TAP service											
11 Accessing the TAP service											

Saada Installation

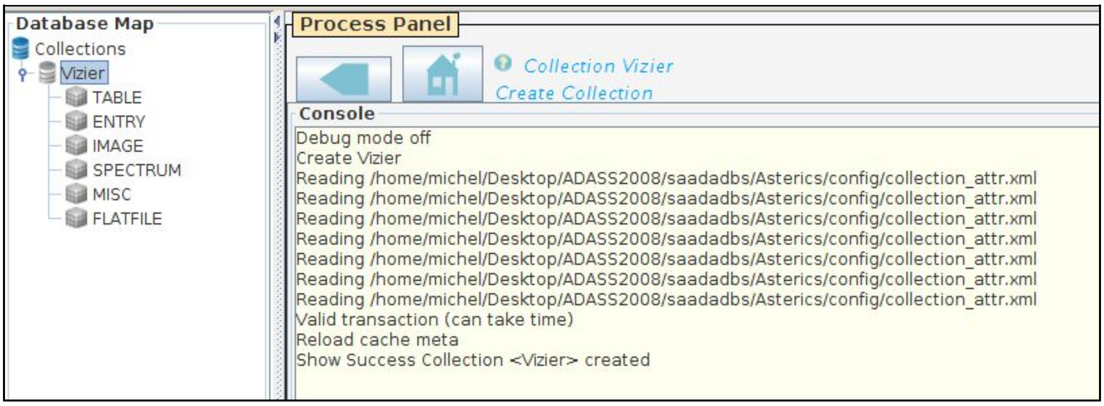
- Download **Saada1.9.build1** from the Saada Web pages
- Download the data sample from *Saada Overview>Tutos and Links*
- Follow the online doc (*tutorial>Getting started>install Saada[Tomcat]*)

Database creation

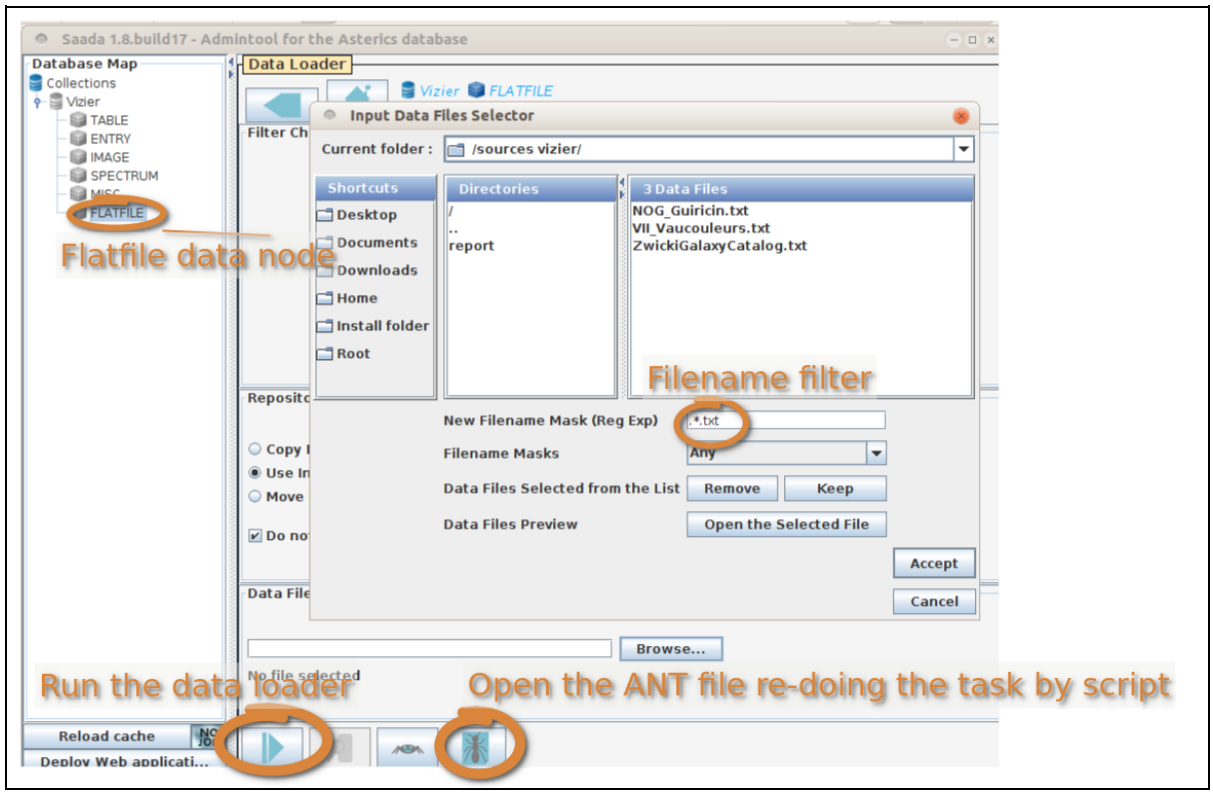
- Follow the online doc (*tutorial>Getting started>create the SaadaDB*)
- Give *Asterics* as *SaadaDB* name
- Choose *SQLite* as DBMS as preference
- Take *Energy/KeV* as spectral unit

Loading catalogues and READMEs

- Most of the coming tasks are run from the graphical administration tool.
    - Go in `./saadadb/Asterics/bin`
    - Run `./admintool`
- **Creating a data collection**:
    - As the 3 catalog extractions are taken out from Vizier, let's call that collection *Vizier. (Admintool > Create collection)*

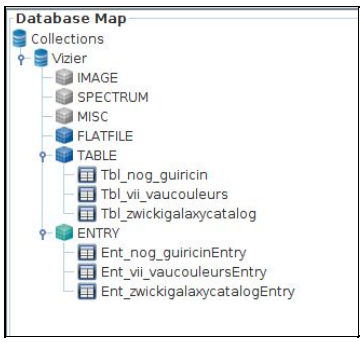


- **Loading the README files**
    - Click on the *FLATFILE* data node
    - Open the *Load Data* panel
    - Open the file browser and go in the *data_sample/Vizier* directory, filter the *.txt file and accept.
    - Run the data loader



- **Display ingested data** with a double click on the *FLATFILE* data node
- **Loading the catalogue extractions**
  - Click on the *TABLE* node
  - Open the data loader panel and take *FITS or VOTable* as filemask in the file explorer
  - Run the loader.

Header metadata are stored the *TABLE* node meanwhile table rows are stored in the *ENTRY* node. As the metadata of these tables are very different, Saada has created 3 different data classes with a name derived from the input file names.



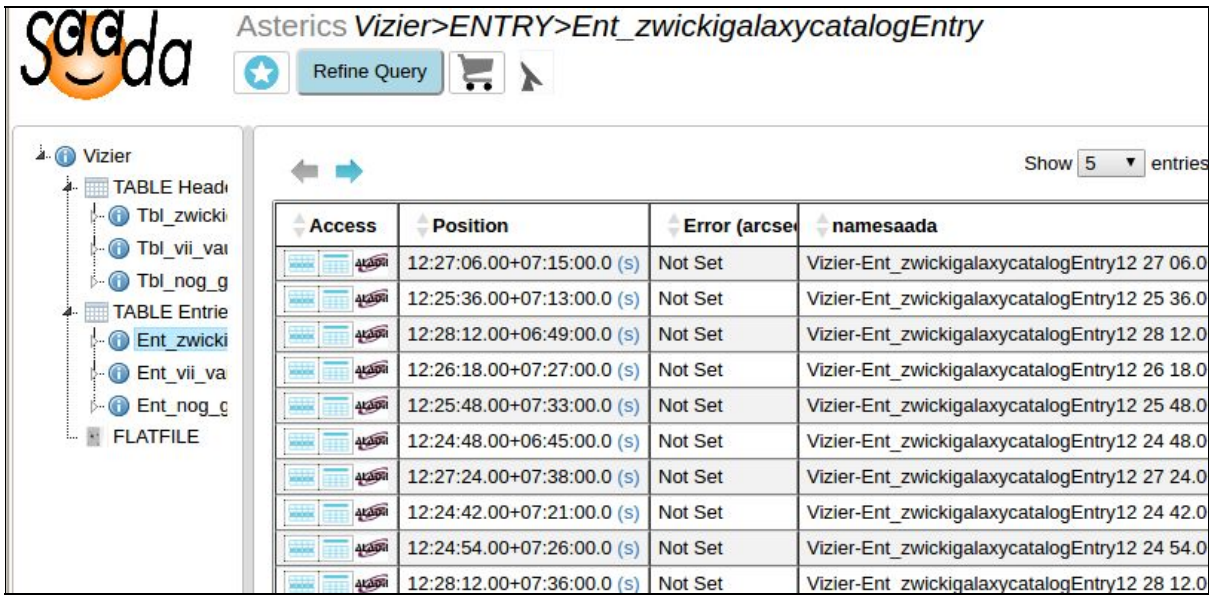
Double click on any node to display the content. There is no data class for the *FLATFILE* node because no data are extracted from flat files.

Testing the Web interface

- We consider that *Tomcat* is running on port 8080 on *localhost*.
- **Click** on the *Deploy Web App* admin tool button.
- **Connect** http://localhost:8080/Asterics. You should retrieve the admintool data tree. Expand it and click on any node to display top data.
  - If you click on a collection level node (*TABLE_Header* e.g.), you get the data computed by Saada. These data are standardized for all ingested datasets.

This interface allows users to select (by position e.g.) data over a set of very different input files.

- If you click on a class level node, you get both collection and class level data.

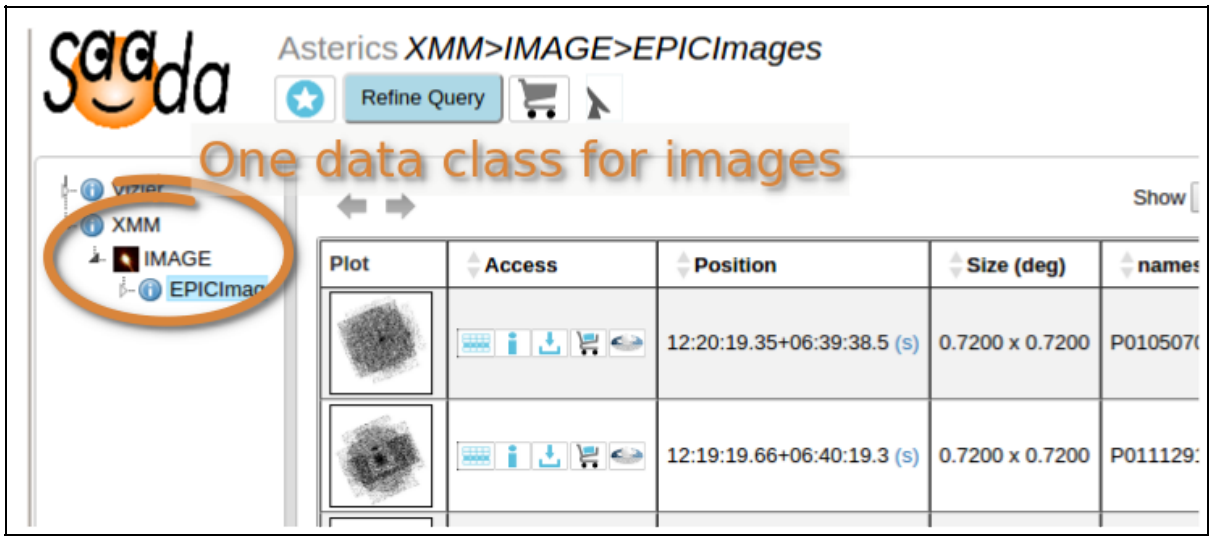


You can refer to the doc online to get an overview of the Web interface features.

Loading images

Our goal now is to load a set of 5 EPIC images taken by XMM-Newton. These images have been measured at different periods. Thus they have been processed by different versions of the pipeline and their metadata (keyword set) can vary from a file to another. But we know that these differences are not significant (just a few KWs added or removed) and that all these images can be stored in the same table (or the same *class* with the *Saada* terminology). That can be achieved by using the **ClassFusion** mode.

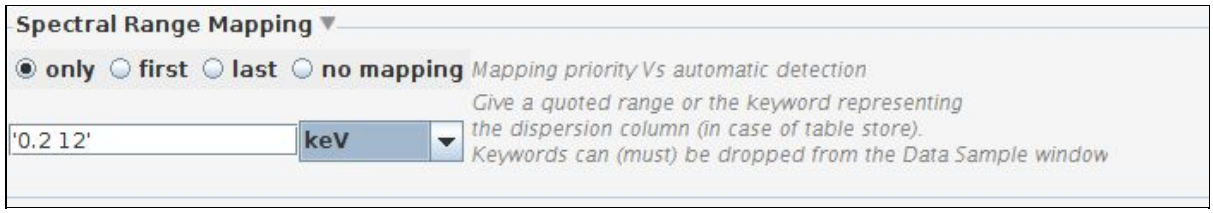
- To make our storage consistent, create first a new collection named *XMM*
- Select (click on) the *IMAGE* node of the *XMM* collection, open the data loader panel and select the content of the *ImageXMM* directory.
- We have no to tell the data loader to gather all these images in one class
  - Click on *New Loader Filter*
  - Tick the *ClassFusion* radio button
  - Set *EPICImage* as class name and save.
- Run the data loader
- Deploy the Web application and reload the Web page.



Loading spectra

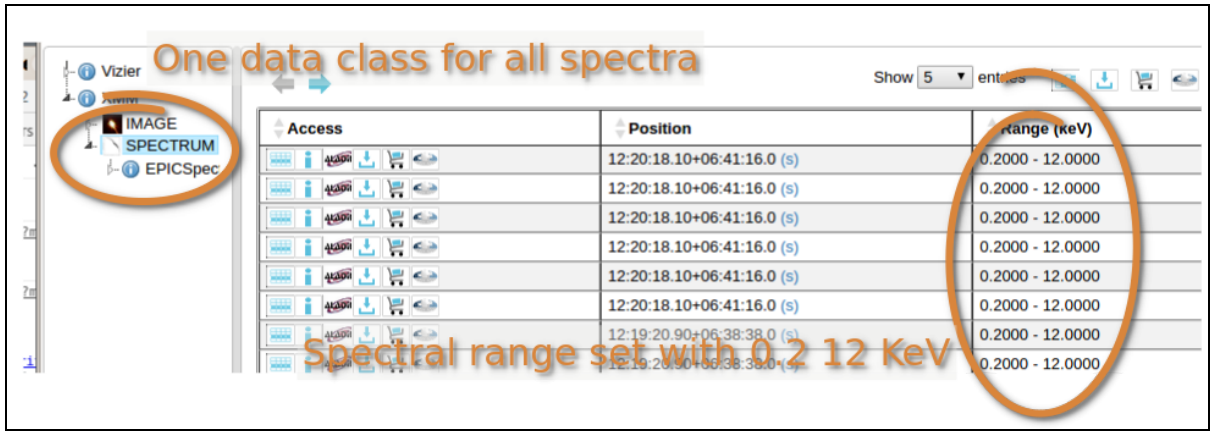
Our goal now is to load a set of 96 EPIC spectra measured by XMM-Newton. As for images, we want to use the **ClassFusion** mode. Spectra will also be stored in the *XMM* collection.
In addition to this, we would like to store the energy range of the spectra. This information cannot be extracted from the data file, but we know that the XMM-Newton EPIC camera ranges from 0.2 to 12.5 KeV (to make simple).

- Back to the admin tool
  - Select the *SPECTRUM* node of the *XMM* collection
  - Open the data loader panel
  - Select the content of the *EPIC Spectra* directory.
- Tell the data loader to gather all these spectra in one class:
  - Click on *New Loader Filter*
  - Tick the *ClassFusion* radio button
  - Set *EPICSpectrum* as class name.
- We now have to tell the data loader to set an energy range.
  - Open the *Spectral Range Mapping* form.
  - Set the *mapping* button on *only*. That means that the data loader won't look for a spectral range in the file keywords. It will just apply the user's rule
  - Select *KeV* as unit
  - Put *'0.2 12'* as range and save



- Run the data loader as usual.

- Deploy the Web application and reload the Web page.



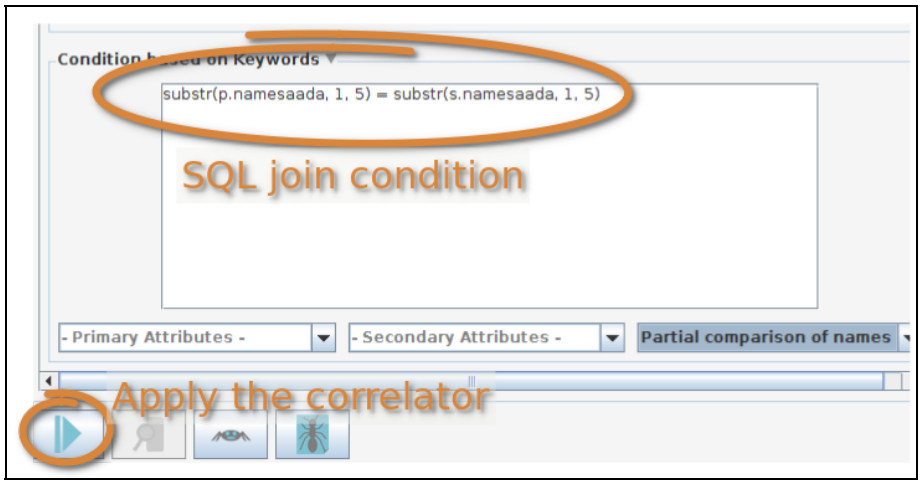
Linking catalogues and READMEs

The *README* files loaded with the table contain a text description of the catalog they are related to. In this task, we will restore this relationship in order to provide a direct access to the *README* from the catalogs on the Web page. That can be done automatically because both catalogs and *READMEs* have similar names.

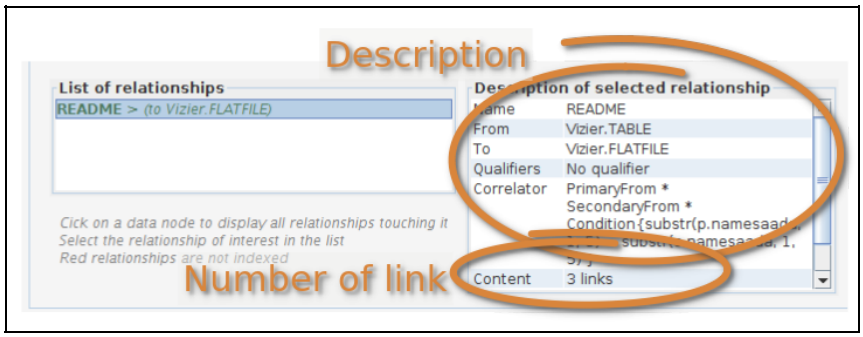
- Select *Vizier.TABLE* on the admin tool data tree. This is the data collection from which the links will start.
- Open the *Manage Relation* panel and click on *New Relation*
- Set *README* as relation name
- Drag the *Vizier.FLATFILE node*, the relation endpoint, to the *To* text field
- Run the command

An empty relationship is created and a popup proposes to populate it. Click *Yes* to open *Correlator Editor* panel

- Open the *Condition based on Keyword* panel
- Select *Partial Comparison of Names* in the *Join operator Templates* menu.
- The SQL fragment shown in the text area can be edited by hand.
- Apply the correlator, deploy the Web application and reload the Web page



The number of links is reported to the panel



On the WEB page, one can now access the README with a simple click .

Linking images, spectra and sources together

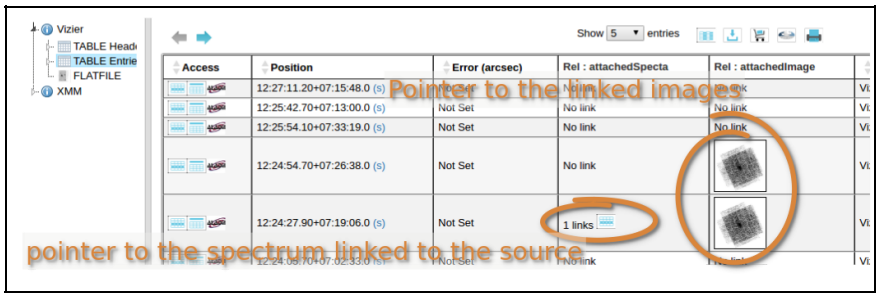
We would like to set pointers from sources to both EPIC spectra and EPIC images. The join condition is now based on the positions. We consider that a spectrum matches a source when the distance in between is lower than 1arcmin. We assume that the image field of view is 30 arcmins. Thus, sources located at less than 15arcmin from an image center will be linked with that image.

Let's start with the spectra:

- On the admin tool, select first the data collection from which the links will start (*Vizier.ENTRY*).
- Open the *Manage Relationships* panel and click on *New Relationship*
- Set *attachedSpectrum* as relation name
- Drag *XMM.SPECTRUM*, the relation endpoint, to the *To* text field
- Run the command
- Accept to populate the relationship and open the *Neighborhood Constraint* panel.
    - Select the closest neighbor (*1st-KNN*)
    - Set the distance threshold at 1 arcmin
    - Run, 3 links are created

Now you do the same for the image with a distance threshold set to 15 arcmin.

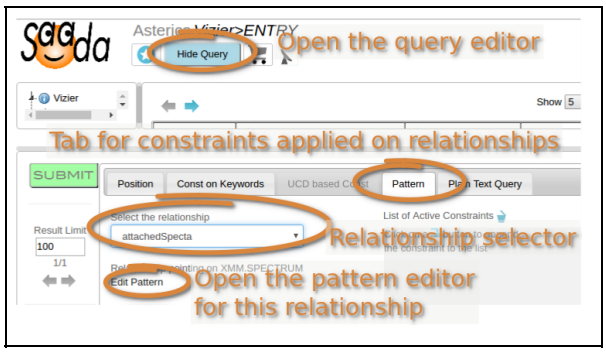
- Take *attachedImage* as relation name.
- Deploy the Web application and reload the Web page.
- Look at the *ENTRY* node of the Vizier collection



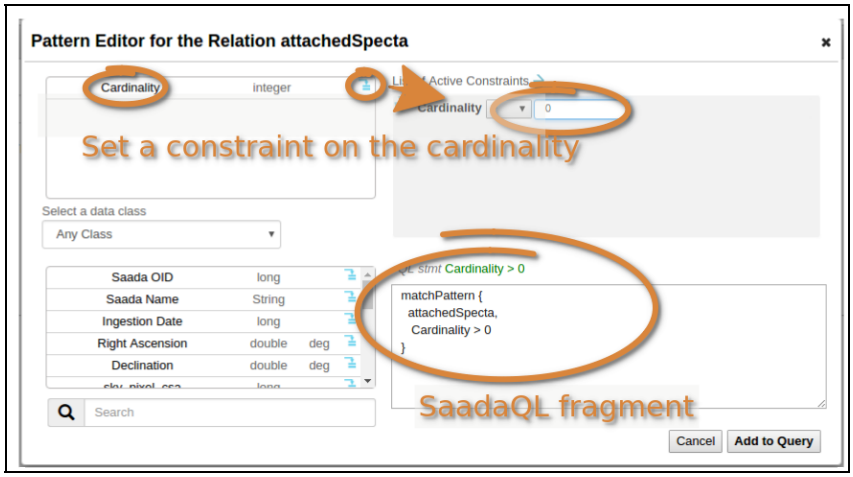
Filtering sources connected with both spectra and images (for the fun)

In this exercise we propose to use an advanced feature of *SaadaQL* to filter catalog entries according the data they are linked to.

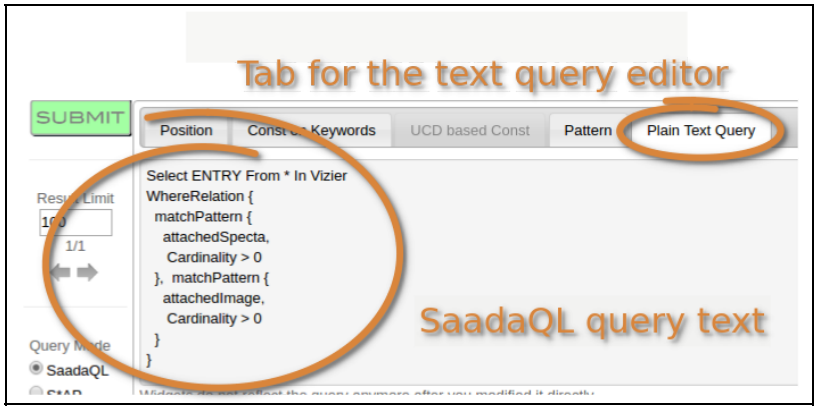
- Click on the *Refine Query* button to open the query editor



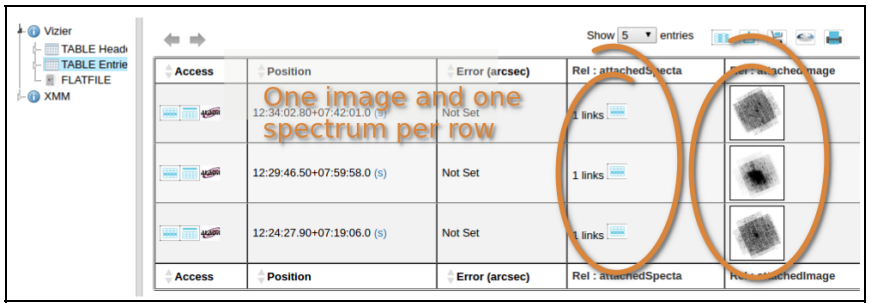
- Click on the *Pattern* tab of the query editor
- Select the relationship *attachedSpectra* from the popup menu.
- Click on *Edit Pattern* to open the pattern editor
    - Set a constraint on the cardinality (number of links)
    - Set the constraint as greater than 0. You can see the valid SaadaQL statement on the bottom right of the panel.
    - Click on *Add to Query*



- Do the same with the relation *attachedImage*.



- If you open the *Plain Text Query* panel, you can see (and modify) the SaadaQL query
- Click on *Submit*.
- Check that only catalog entries with one spectrum and one source are displayed.

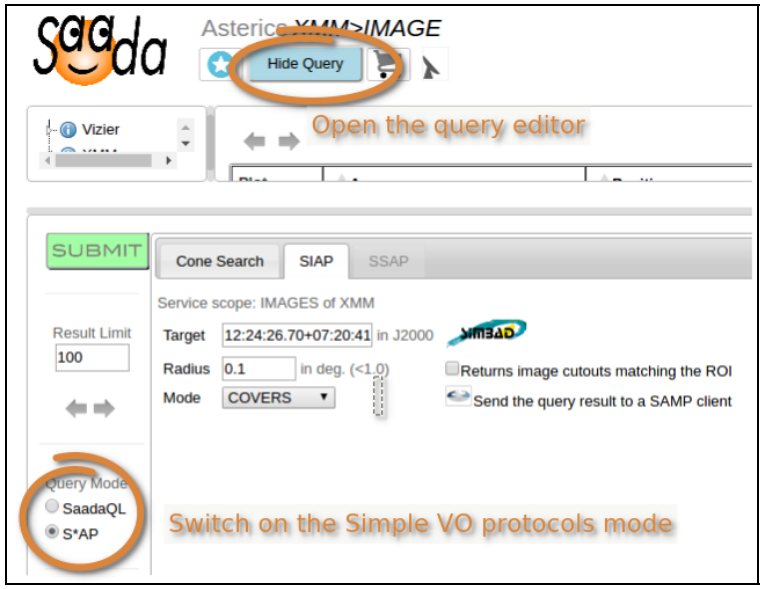


Publishing a simple VO service

The is no specific action to run a simple service.
Simple VO queries can be tested from the Web interface.

- Select a data collection (*IMAGE*, *SPECTRUM* or *ENTRY*)
- Open the query form
- Click on *S*AP* to switch on the VO mode.

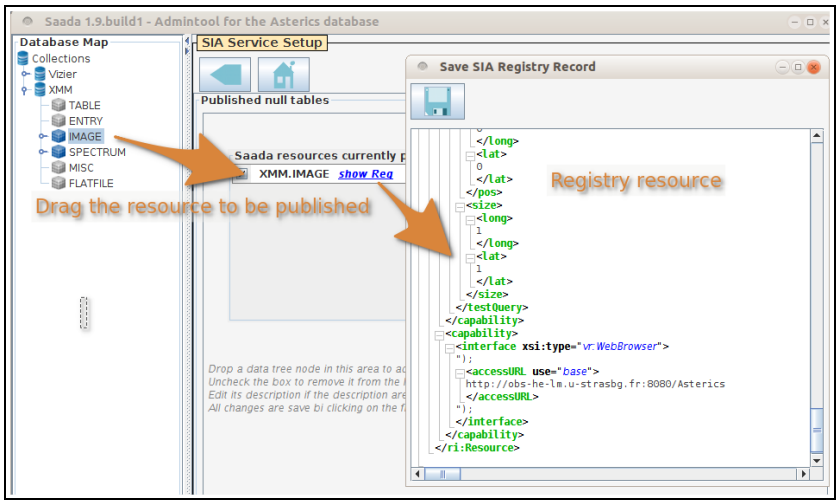
- Submit a query . The results comes in a VOTable. It could either be displayed by the browser or downloaded according to your local setup.



In order to help for the management of the published VO resources, *Saada* can however keep a reference of the data collection you wish to publish.

- Open the *VO Publishing* panel of the admin tool

- Select to type of the simple protocol you want to implement
- Drag the data collection you want to publish. The *Show Reg* anchor opens a valid registry record for the data collection.

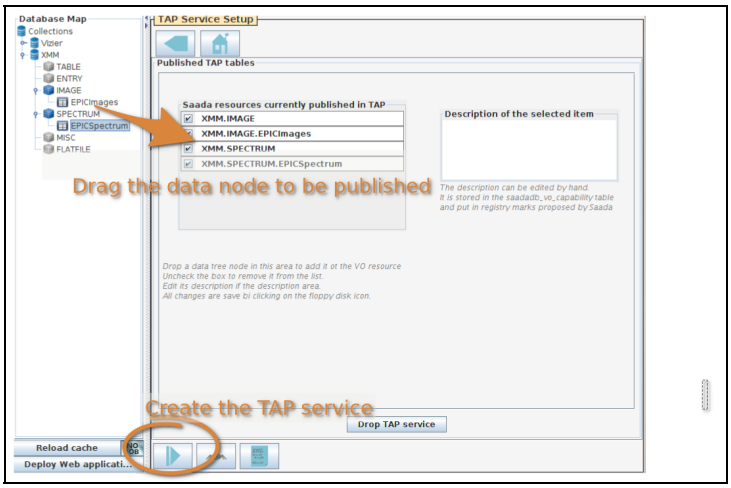


Publishing a TAP service

Unlike simple services, TAP requires to write specific resources (the *TAP_SCHEMA*) into the database.

- Open the *VO Publishing* panel of the admin tool

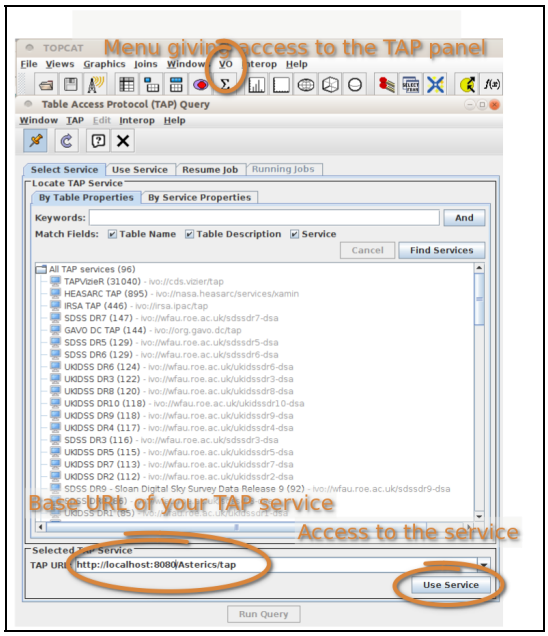
- Open the *Publish TAP* Service panel .
- Drag the data collection you want to publish.
- Execute



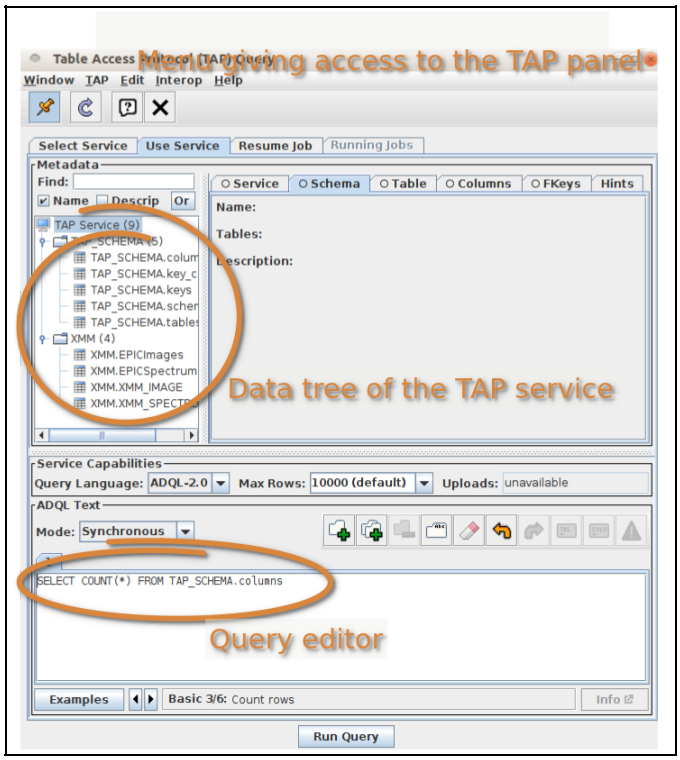
Accessing the TAP service

Although your TAP service is local, TopCAT can connect it.

- Run Topcat
    - Ex: javaws -Xmx=1024m 'http://www.star.bris.ac.uk/~mbt/topcat/topcat-full.jnlp'
- Select *Table Access Protocol* in the VO menu
- Type the service URL
    - Ex: http://localhost:8080/Asterics/tap



- Click on *Use Service* and go ahead

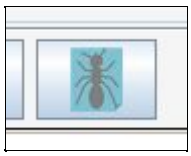

Menu giving access to the TAP panel
Window TAP Edit Interop Help
Select Service
Use Service
Resume Job
Running Jobs
Metadata
Find:
Name
Descrip
Service
Schema
Table
Columns
FKeys
Hints
Name:
Tables:
Description:
TAP Service (9)
TAP_SCHEMA (5)
TAP_SCHEMA.keys
XMM (4)
XMM.EPICImages
XMM.EPICSpectrum
XMM.XMM_IMAGE
Data tree of the TAP service
Service Capabilities
Query Language: ADQL-2.0
Max Rows: 10000 (default)
Uploads: unavailable
ADQL Text
Mode: Synchronous
SELECT COUNT(*) FROM TAP_SCHEMA.columns
Query editor
Examples
Basic 3/6: Count rows
Info
Run Query

References

- **Saada**
    - Pages: http://saada.unistra.fr
    - Paper: *Building an Archive with Saada* https://arxiv.org/abs/1409.0351
- **Aladin:** http://aladin.u-strasbg.fr/
- **Topcat:** http://www.star.bris.ac.uk/~mbt/topcat/
- **IVOA:** http://www.ivoa.net/
- **SQLite:** https://www.sqlite.org/
- **Tomcat:** http://tomcat.apache.org/
- **CDS:** http://cdsweb.u-strasbg.fr/index-fr.gml
- **NED:** https://ned.ipac.caltech.edu/

Annex: Saada Scripts

Most of the Saada commands can be run by *Ant* scripts. The Saada distribution comes with an *Ant* task descriptor allowing to run lots of commands by scripts (see *Saada* pages). In addition with this, you can download from the admin tool an *Ant* file for each command you run. Just click on the ant icon.



Creating the database by scripts

Creating a database by script is less straightforward. The code below is a descriptor of a task doing this for the present database. The *Saada* pages also provide good tips for this.

```
<project name="Creating a DB" default=”all”>
  <property name="jvm_initial_size" value="-Xms64m" />
  <property name="jvm_max_size"  value="-Xmx1024m" />
  <property name="SAADA_HOME"  value="-Xmx1024m" />
  <property name="TOMCAT_HOME"  value="-Xmx1024m" />
  <property name="jvm_max_size"  value="-Xmx1024m" />
  <property name="jvm_max_size"  value="-Xmx1024m" />
  <property name="jvm_max_size"  value="-Xmx1024m" />


    <path id="saada.classpath">
        <fileset dir="${SAADA_HOME}/dbtemplate/lib/">
        <include name="**/*.jar" />
        </fileset>
        <fileset dir="${SAADA_HOME}/jtools/">
                        <include name="**/*.jar" />
        </fileset>
    </path>

       <property name="TOMCAT_HOME" value="/rawdata"/>
  <property name="XCATDBREPDIR" value="/base2/repository/XCATDBi"/>

  <!-- Saadadb initialisation  -->
  <target name="saadadb.build" depends="saadadb.init,saadadb.create">
  </target>

  <target name="saadadb.init" >
       <delete dir="${TOMCAT_HOME}/webapps/${SAADA_DB_NAME}" failonerror="false"/>
        <exec executable="dropdb" failonerror="false">
               <arg value="-h"/>
               <arg value="${PSQL_HOST}"/>
               <arg value="-U"/>
               <arg value="${PSQL_USER}"/>
               <arg value="${SAADA_DB_NAME}"/>
        </exec>
  </target>
```

```
  <target name="saadadb.create" >
        <copy file="../../env/saadadb.xml" todir="${SAADA_HOME}/config"
failonerror="true" overwrite="true"/>
                <copy file="../../env/collection_attr.xml" todir="${SAADA_HOME}/config"
failonerror="true" overwrite="true"/>
                <copy file="../../env/saadadb.properties" todir="${SAADA_HOME}/bin"
failonerror="true" overwrite="true"/>
        <exec executable="createdb" failonerror="true">
                <arg value="-h"/>
                <arg value="${PSQL_HOST}"/>
                <arg value="-U"/>
                <arg value="${PSQL_USER}"/>
                <arg value="${SAADA_DB_NAME}"/>
        </exec>
        <java classname="saadadb.newdatabase.NewSaadaDB" failonerror="true" fork="true" >
       <classpath refid="saada.classpath"/>
              <arg value="${SAADA_HOME}"/>
        </java>
  </target>

  <target name="xmode.set">
        <chmod file="${REPOSITORY}/tmp" type="dir" perm="777"/>
  </target>
</project>
```