

# Uniprot Sparql Wrapper for Galaxy

Francisco Abad Navarro, Macarena López Sánchez

**This document shows how a python script has been developed to execute queries against Uniprot database using its sparql interface and how this script has been linked to galaxy by making a wrapper. Finally, this tool has been uploaded to Galaxy Tool Shed so that other Galaxy users can download and use it.**

## 1 Python Script

Firstly, a python script (**spaql\_uniprot.py**, Appendix 1) has been developed to perform queries against Uniprot sparql endpoint. In order to do that, a python library called *SPARQLWrapper*, which is necessary to install via pip, has been used in the script. *SPARQLWrapper* has been chosen because it has an easier usage. It only takes a few parameters to work:

- Sparql endpoint location.
- Query to perform against sparql endpoint.
- Output format of the server response. It could be json or xml.

Therefore, the developed script has the following main structure using *SPARQLWrapper*:

```
# Build a string with the query that we are going to perform.  
query = buildQuery(queryParams)
```

```
# Print query to stdout for debug
print query

# Create a SPARQLWrapper object linked to the specified sparql endpoint
sparql = SPARQLWrapper('http://sparql.uniprot.org/sparql')

# Set the query to perform
sparql.setQuery(query)

# Set the output format
sparql.setReturnFormat(JSON)

# Perform the query
results = sparql.query()

# Get the results in json format
json = results.convert()

# Print info about server response for debug
print results.info()

# Print the results to output file with tabular format
printResults(json, output)
```

The script receives following parameters through command line:

1. Uniprot identifier.

2. Protein name.
3. Gene name.
4. Organism name.
5. Disease annotation.
6. Domain name.
7. Similarity annotation.
8. Location annotation.
9. Function annotation.
10. Pharmaceutical annotation.
11. Output file.

All these parameters excepting output file are used as searching fields. They have to appear in this order when the command is called. If you do not want to specify any of these parameters, you must write "" in the position corresponding to the parameter. Results will be stored in the specified output file. This file has a tabular format.

In the script, *buildQuery* function receives the parameters listed before. This function return a string with the final query that *SPARQLWrapper* will execute against Uniprot sparql endpoint. The behaviour of how the query is built is described below:

- Filters are not applied to Uniprot identifier parameter so the query will search the exact identifier matching in the Uniprot database.

- Filters are applied to the rest of parameters. Because of that, regular expressions are allowed. If you want to search the exact matching you should use ^at the beginning and \$ at the end of the parameter value.
- If a parameter is empty, it is set as optional so the results may include information about it or not. In other case, parameter is not optional so only results that contain the parameter will be reported.

The script will write to stdout some information as the final query to perform against Uniprot or the server response for debugging purposes.

## 2 Linking script to Galaxy

So as to include the script **sparql\_uniprot.py** into galaxy we have to create **sparql\_uniprot.xml** (Appendix 2), which is used by Galaxy to extract information about how to call the script, expected parameters, the output files, help text to be shown to the user or requirements. When both files are ready, it is necessary to move them into biomaster server. We will add the files to a folder called *sparql\_uniprot* and then we will copy it into **/home/galaxy2016/galaxy/tools**.

After that, it is necessary to register the new tool by adding it into **tool\_conf.xml**, which is inside Galaxy installation folder (**/home/galaxy2016/galaxy/config/tool\_conf.xml.sample**).

### 2.1 sparql\_uniprot.xml

The command and parameters to call **sparql\_uniprot.xml** is defined in this file. We have to specify the interpreter, which is Python in this case, and the parameters that it takes as follows:

```
<command interpreter="python">sparql_uniprot.py "${protein}"  
"${proteinName}"  
"${geneName}" "${organismName}"  
"${diseaseAnnotation}" "${domainName}"  
"${similarityAnnotation}" "${locationAnnotation}"  
"${functionAnnotation}" "${pharmaceuticalAnnotation}"  
"${output}"</command>
```

By defining parameter as "\$parameterName" instead of \$parameterName, Galaxy will send an empty string ("" ) to the command when a search field will not filled. In other case, Galaxy will not include the quotes, so the script will fail because it expect eleven parameters.

The parameter type is defined as follows:

```
<inputs>
```

```

<param name="protein" type="text" label="Uniprot Identifier"/>
<param name="proteinName" type="text" label="Protein Name">
    <sanitizer sanitize="False"/>
</param>
<param name="geneName" type="text" label="Gene Name">
    <sanitizer sanitize="False"/>
</param>
</inputs>

```

All input parameters are defined as text type, so galaxy will include a text box where users will write. All input parameters except the Uniprot identifier allow regular expression. In these parameters we have set sanitize option as false in order to write special characters like '\$'. In other case, for security reasons, these characters will be replace by 'X'.

Output parameters is defined as tabular format as follows:

```

<outputs>
    <data format="tabular" name="output" />
</outputs>

```

On the other hand **sparql\_uniprot.py** needs *python* and *SPARQLWrapper* python module to work. This fact can be specified in requirements section:

```

<requirements>
<requirement type="binary">python</requirement>
<requirement type="package" version="1.7.6">SPARQLWrapper</requirement>
</requirements>

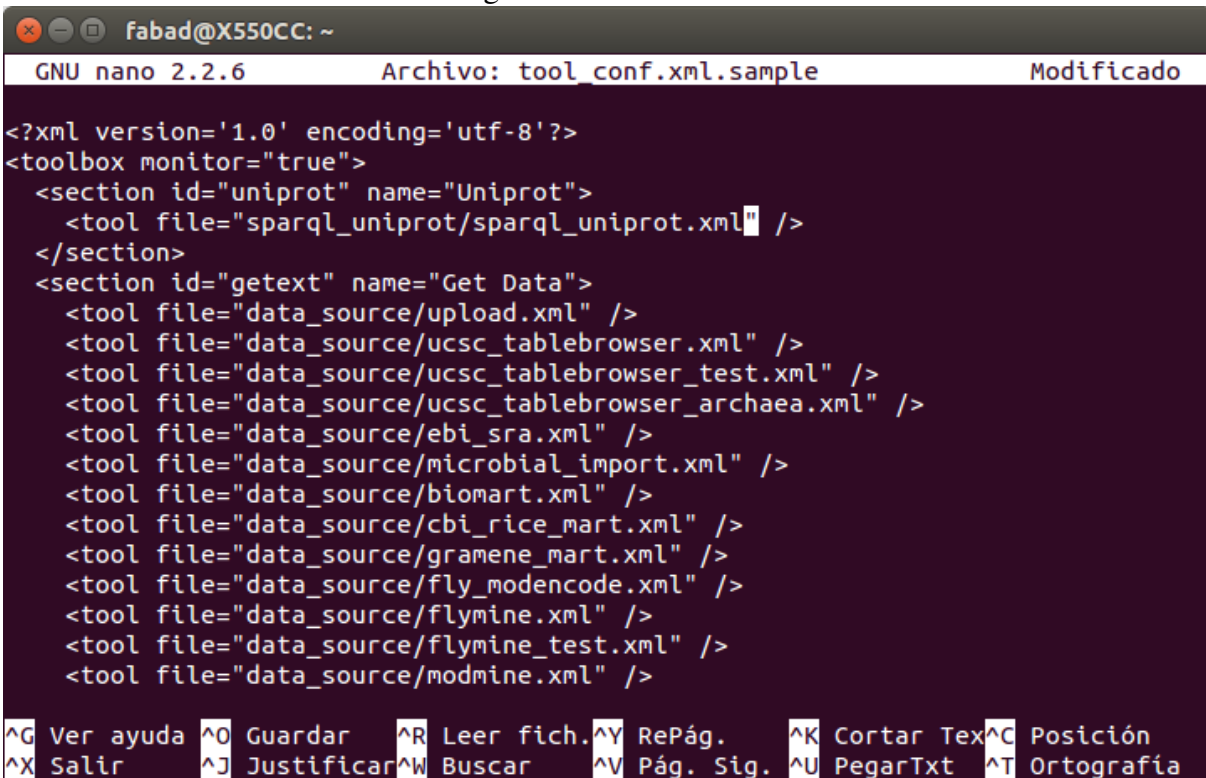
```

Finally a help text have been added in "help" tags.

## 2.2 tool\_conf.xml

This file is used to register tools in galaxy. We have to modify it in the server in order to add our tool as figure 1 shows:

Figure 1: tool\_conf.xml



```
fabad@X550CC: ~
GNU nano 2.2.6      Archivo: tool_conf.xml.sample      Modificado
<?xml version='1.0' encoding='utf-8'?>
<toolbox monitor="true">
  <section id="uniprot" name="Uniprot">
    <tool file="sparql_uniprot/sparql_uniprot.xml" />
  </section>
  <section id="gettext" name="Get Data">
    <tool file="data_source/upload.xml" />
    <tool file="data_source/ucsc_tablebrowser.xml" />
    <tool file="data_source/ucsc_tablebrowser_test.xml" />
    <tool file="data_source/ucsc_tablebrowser_archaea.xml" />
    <tool file="data_source/ebi_sra.xml" />
    <tool file="data_source/microbial_import.xml" />
    <tool file="data_source/biomart.xml" />
    <tool file="data_source/cbi_rice_mart.xml" />
    <tool file="data_source/gramene_mart.xml" />
    <tool file="data_source/fly_modencode.xml" />
    <tool file="data_source/flymine.xml" />
    <tool file="data_source/flymine_test.xml" />
    <tool file="data_source/modmine.xml" />
  </section>
</toolbox>
```

^G Ver ayuda ^O Guardar ^R Leer fich. ^Y RePág. ^K Cortar Tex ^C Posición  
^X Salir ^J Justificar ^W Buscar ^V Pág. Sig. ^U PegarTxt ^T Ortografía

The new tool has been inserted into a new section called Uniprot.

### 3 Galaxy Tool Shed

We have decided to share the new tool we have developed with other Galaxy users through Galaxy Tool Shed. To do this, a repository called *sparql\_uniprot* has been created at [https://toolshed.g2.bx.psu.edu/view/fabad/sparql\\_uniprot](https://toolshed.g2.bx.psu.edu/view/fabad/sparql_uniprot). This repository contains the following files:

- **sparql\_uniprot.py** - The script that query against Uniprot database.
- **sparql\_uniprot.xml** - The configuration file that tells Galaxy how to run *sparql\_uniprot.py*.
- **tool\_dependencies.xml** - The configuration file that allows Galaxy to manage dependencies.

The new file that is not explained before is **tool\_dependencies.xml** (Appendix 3). It contains information about how galaxy must manage the dependencies that **sparql\_uniprot.py** has, which are described in **sparql\_uniprot.xml**. In this case, python module *SPARQLWrapper* is required to run the tool, so **tool\_dependencies.xml** indicate how to install it. By doing this, when a user download the tool via Galaxy Tool Shed, *SPARQLWrapper* will be downloaded automatically.

Notice that *setup\_virtualenv* action will install python module *SPARQLWrapper* via pip.



## 4 Some examples

Different searches have been made to check if the tool is working correctly. This queries are summarized by the following points:

- Q13563, uniprot identifier, is used to find out different parameters related with it (Figure 2).
- ^and \$ have been used to perform a perfect match search about INS gene (Figure 4).
- Search about the PTEN gene in the 'sapiens' organism (Figure 6).
- Search of proteins belong to 'Mouse' organism and It are secreted (Figure 8).
- 'Associates with actin filament appendages that are formed in the inclusion appendages of the parasitophorous vacuole during infection of the host erythrocyte' has been used as function annotation for searching the protein that accomplish it(Figure 10).

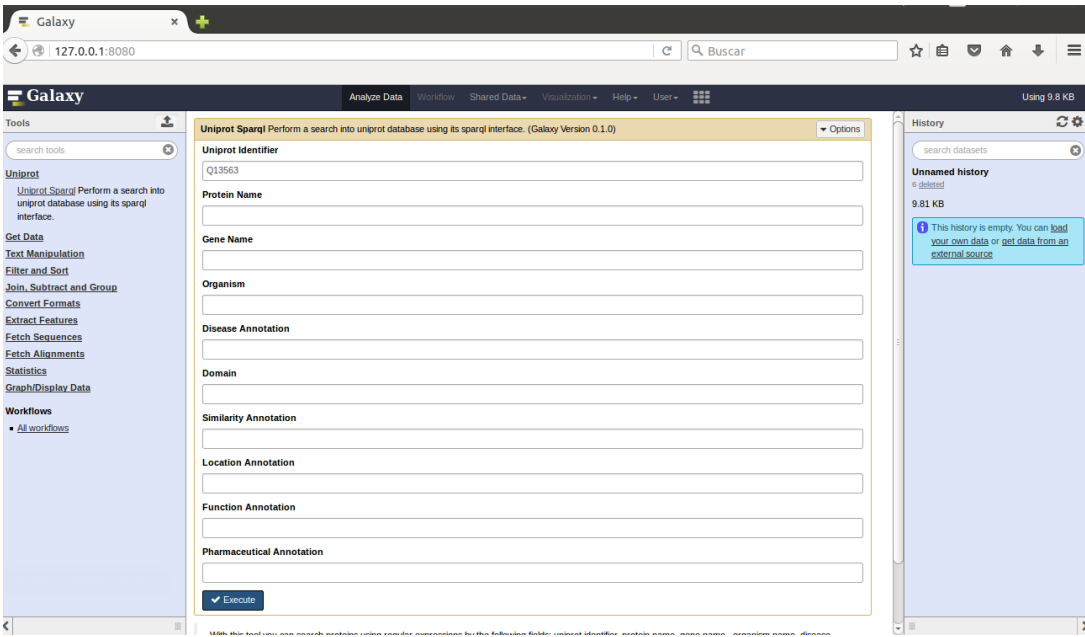


Figure 2: Search 1

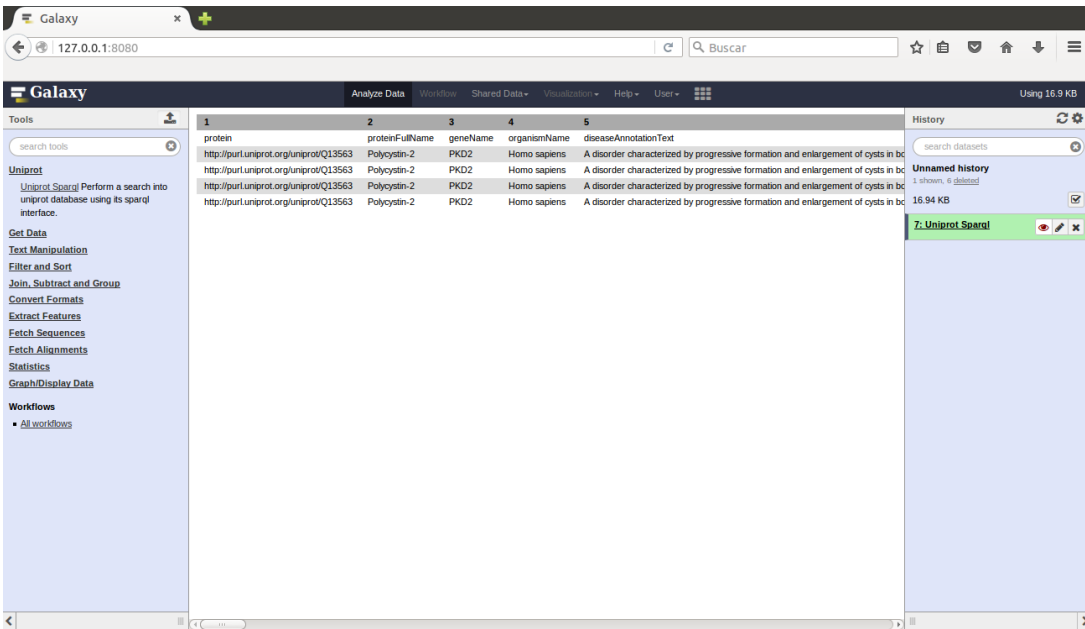


Figure 3: Results of search 1

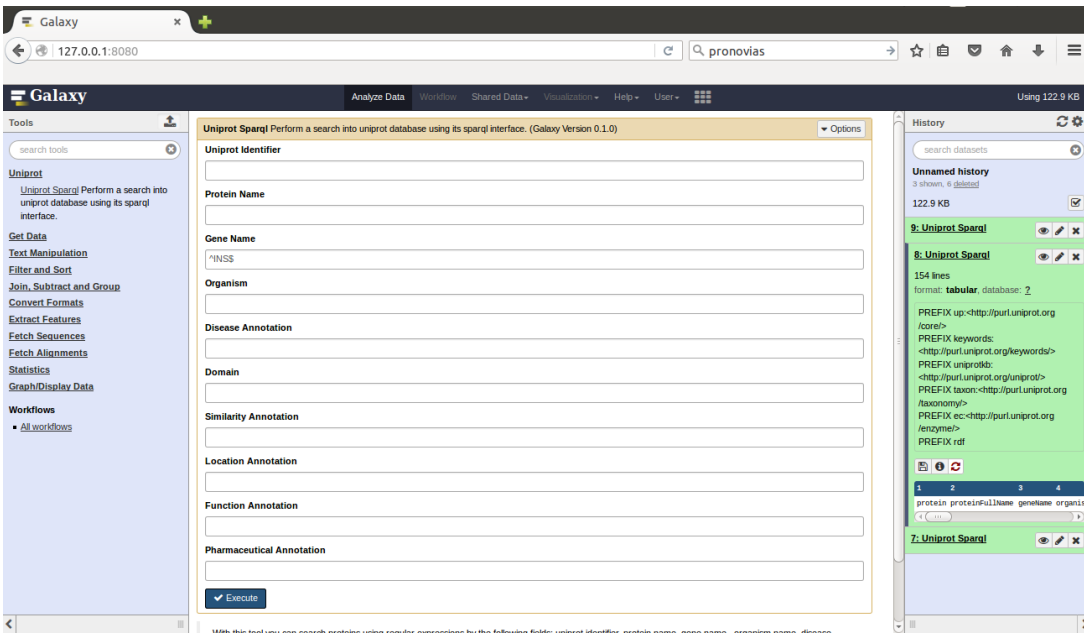


Figure 4: Search 2

Galaxy

127.0.0.1:8080

pronovias

Galaxy

Analyze Data Workflow Shared Data Visualization Help User

Tools

search tools

Uniprot

Uniprot Searq Perform a search into uniprot database using its sparql interface.

Get Data

Text Manipulation

Filter and Sort

Join, Subtract and Group

Convert Formats

Extract Features

Fetch Sequences

Fetch Alignments

Statistics

Graph/Display Data

Workflows

All workflows

1	2	3	4	5
protein	proteinFullname	geneName	organismName	diseaseAnnotationText
http://purl.uniprot.org/uniprot/AA000FNE3	ins	Burkholderia sp.	MSHR3999	
http://purl.uniprot.org/uniprot/Q6TM12	ins	Burkholderia multivorans		
http://purl.uniprot.org/uniprot/P01317	Insulin	INS	Bos taurus	
http://purl.uniprot.org/uniprot/B5X6U5	INS	INS	Salmo salar	
http://purl.uniprot.org/uniprot/B5XDT4	INS	INS	Salmo salar	
http://purl.uniprot.org/uniprot/B5XEB7	INS	INS	Salmo salar	
http://purl.uniprot.org/uniprot/AA0AM4UN8	INS	INS	Eubiepharis macularius	
http://purl.uniprot.org/uniprot/F6QQU6	INS	INS	Equus caballus	
http://purl.uniprot.org/uniprot/AA0AM3V60	INS	INS	Gecko gecko	
http://purl.uniprot.org/uniprot/P01308	Insulin	INS	Homo sapiens	An autosomal dominant condition characterized by elev
http://purl.uniprot.org/uniprot/P01308	Insulin	INS	Homo sapiens	A rare form of diabetes distinct from childhood-onset au
http://purl.uniprot.org/uniprot/P01308	Insulin	INS	Homo sapiens	A form of diabetes that is characterized by an automa
http://purl.uniprot.org/uniprot/P01308	Insulin	INS	Homo sapiens	A multifactorial disorder of glucose homeostasis that is c
http://purl.uniprot.org/uniprot/P01329	Insulin	INS	Cavia porcellus	
http://purl.uniprot.org/uniprot/P04667	Insulin	ins	Oncorhynchus keta	
http://purl.uniprot.org/uniprot/P01333	Insulin	INS	Anas platyrhynchos	
http://purl.uniprot.org/uniprot/P01320	Insulin	INS	Camelus dromedarius	
http://purl.uniprot.org/uniprot/P01339	Insulin	ins	Thunnus thynnus	
http://purl.uniprot.org/uniprot/P67972	Insulin	INS	Actus trivirgatus	
http://purl.uniprot.org/uniprot/P12708	Insulin	INS	Ptyas dhumnades	
http://purl.uniprot.org/uniprot/P01334	Insulin	INS	Crotalus atrox	
http://purl.uniprot.org/uniprot/P67970	Insulin	INS	Gallus gallus	
http://purl.uniprot.org/uniprot/P67968	Insulin	INS	Meleagris gallopavo	
http://purl.uniprot.org/uniprot/P30410	Insulin	INS	Pan troglodytes	
http://purl.uniprot.org/uniprot/P01321	Insulin	INS	Canis lupus familiaris	
http://purl.uniprot.org/uniprot/P06306	Insulin	INS	Felis catus	
http://purl.uniprot.org/uniprot/P01310	Insulin	INS	Equus caballus	
http://purl.uniprot.org/uniprot/P01315	Insulin	INS	Sus scrofa	
http://purl.uniprot.org/uniprot/P01319	Insulin	INS	Capra hircus	
http://purl.uniprot.org/uniprot/P01318	Insulin	INS	Ovis aries	

History

search datasets

Unnamed history

3 shown, 6 datasets

122.9 KB

9: Uniprot Searq

8: Uniprot Searq

154 lines

format: tabular, database: 2

PREFIX up-<http://purl.uniprot.org/

core/>

PREFIX keywords:

<http://purl.uniprot.org/keywords/>

PREFIX uniprot:

<http://purl.uniprot.org/uniprot/>

PREFIX taxon-<http://purl.uniprot.org/

/taxonomy/>

PREFIX ec-<http://purl.uniprot.org/

/enzyme/>

PREFIX rdf

1 2 3 4

protein proteinFullname geneName organism

7: Uniprot Searq

Figure 5: Results of search 2



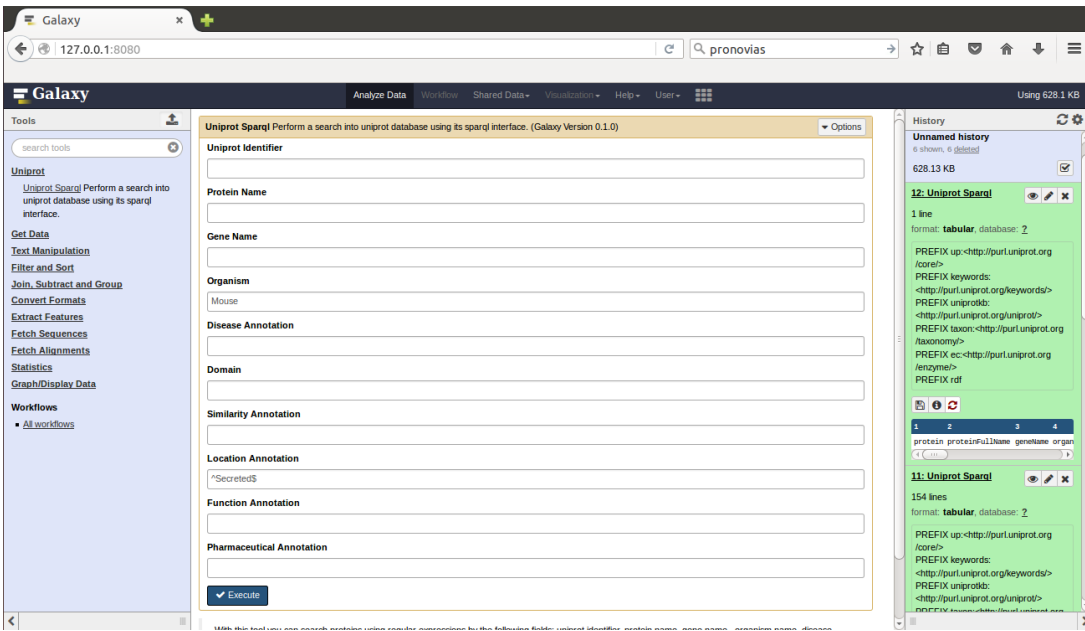


Figure 8: Search 4

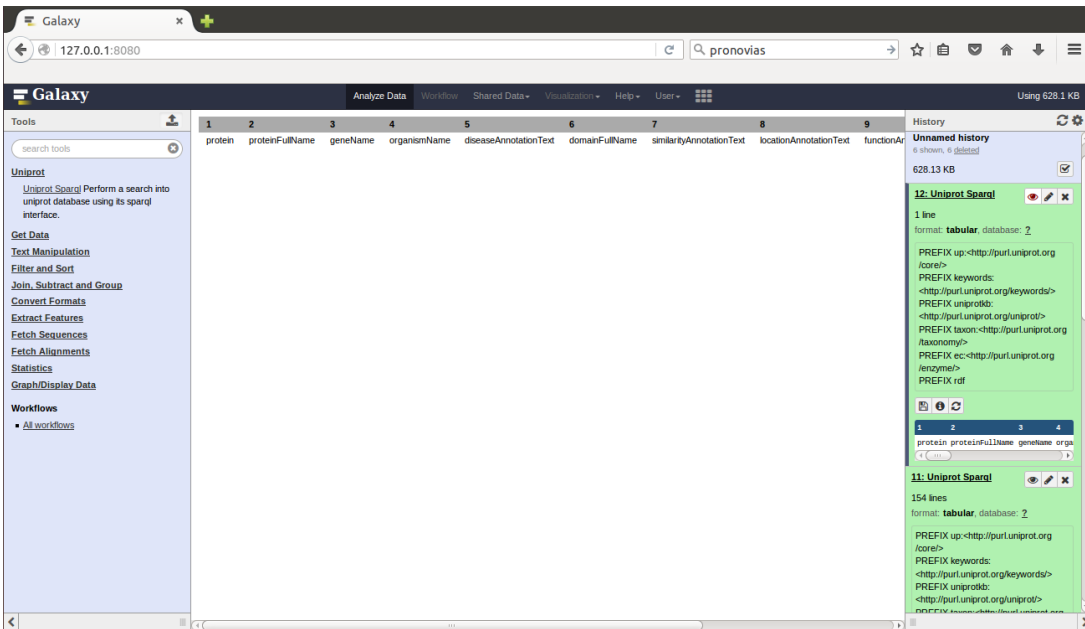


Figure 9: Results of search 4

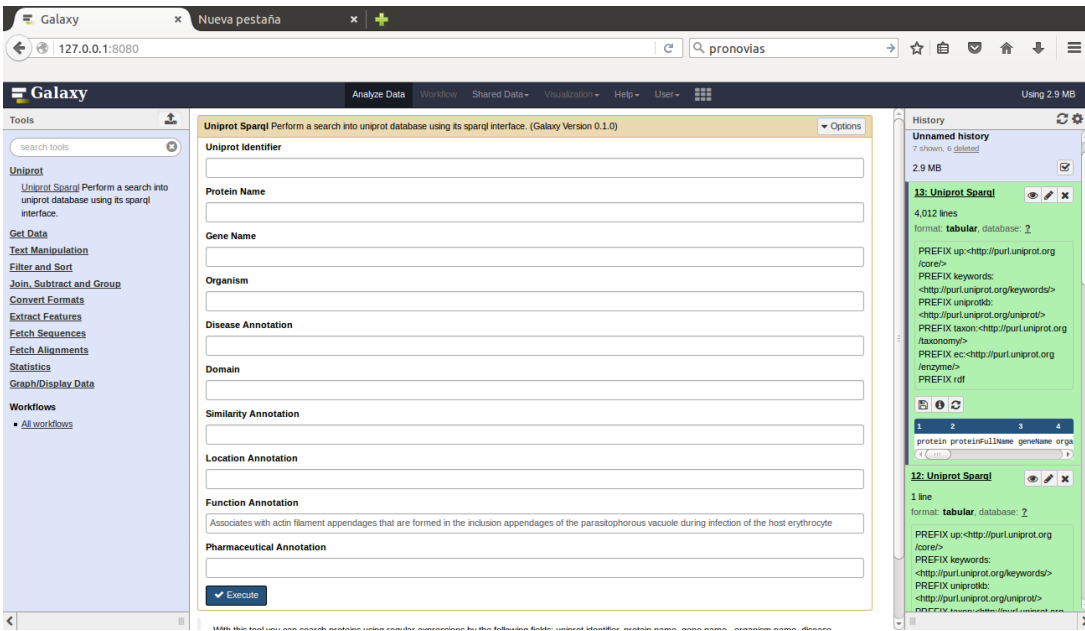


Figure 10: Search 5

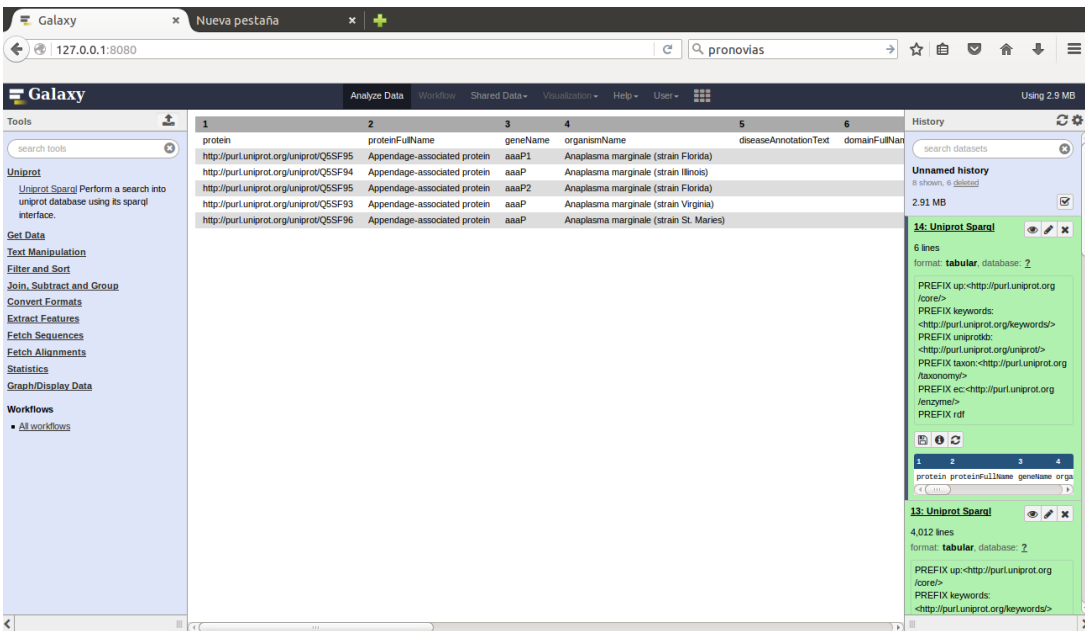


Figure 11: Results of search 5

## Appendix 1

```
from SPARQLWrapper import SPARQLWrapper, JSON
import sys

# Constante con los prefijos comunes a usar en queries
COMMON_PREFIXES = """
    PREFIX up:<http://purl.uniprot.org/core/>
    PREFIX keywords:<http://purl.uniprot.org/keywords/>
    PREFIX uniprotkb:<http://purl.uniprot.org/uniprot/>
    PREFIX taxon:<http://purl.uniprot.org/taxonomy/>
    PREFIX ec:<http://purl.uniprot.org/enzyme/>
    PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
    PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
    PREFIX skos:<http://www.w3.org/2004/02/skos/core#>
    PREFIX owl:<http://www.w3.org/2002/07/owl#>
    PREFIX bibo:<http://purl.org/ontology/bibo/>
    PREFIX dc:<http://purl.org/dc/terms/>
    PREFIX xsd:<http://www.w3.org/2001/XMLSchema#>
    PREFIX faldo:<http://biohackathon.org/resource/faldo#>
"""

# Lista con los nombres de las variables que obtenemos de la base de datos.
paramList = [
    'protein',
```

```
'proteinFullName',  
'geneName',  
'organismName',  
'diseaseAnnotationText',  
'domainFullName',  
'similarityAnnotationText',  
'locationAnnotationText',  
'functionAnnotationText',  
'pharmaceuticalAnnotationText']
```

```
def buildQuery(  
    proteinId,  
    proteinName,  
    geneName,  
    organismName,  
    diseaseAnnotation,  
    domainName,  
    similarityAnnotation,  
    locationAnnotation,  
    functionAnnotation,  
    pharmaceuticalAnnotation):  
    query = COMMON_PREFIXES  
    query += "select distinct \n"  
    query += "        ?protein\n"
```



```

query += "        ?proteinFullName\n"
query += "        ?geneName\n"
query += "        ?organismName\n"
query += "        ?diseaseAnnotationText\n"
query += "        ?domainFullName\n"
query += "        ?similarityAnnotationText\n"
query += "        ?locationAnnotationText\n"
query += "        ?functionAnnotationText\n"
query += "        ?pharmaceuticalAnnotationText\n"
query += "where{\n"

query += "        ?protein a up:Protein .\n"

if (proteinId != ''):
    query += "        VALUES ?protein {uniprotkb:" + proteinId + "}\n"

query += "\n"

if (proteinName == ''):
    query += "        OPTIONAL {\n"
query += "        ?protein up:recommendedName ?proteinName .\n"
query += "        ?proteinName up:fullName ?proteinFullName .\n"
if (proteinName != ''):
    query += "        filter( regex(str(?proteinFullName), " + \
        "'" + proteinName + "'" + ", \"i\" )) .\n"

```

```

if (proteinName == ''):
    query += "    }\n"
query += "\n"

if (geneName == ''):
    query += "    OPTIONAL {\n"
query += "    ?protein up:encodedBy ?gene .\n"
query += "    ?gene skos:prefLabel ?geneName .\n"
if (geneName != ''):
    query += "    filter( regex(str(?geneName), " + \
        "'" + geneName + "'" + ", \"i\") )) .\n"
if (geneName == ''):
    query += "    }\n"

query += "\n"

if (organismName == ''):
    query += "    OPTIONAL {\n"
query += "    ?protein up:organism ?organism .\n"
query += "    ?organism up:scientificName ?organismName .\n"
if (organismName != ''):
    query += "    filter( regex(str(?organismName), " + \
        "'" + organismName + "'" + ", \"i\") )) .\n"
if (organismName == ''):
    query += "    }\n"

```

```

query += "\n"

if (diseaseAnnotation == ''):
    query += "        OPTIONAL {\n"
query += "        ?protein up:annotation ?diseaseAnnotation .\n"
query += "        ?diseaseAnnotation a up:Disease_Annotation .\n"
query += "        ?diseaseAnnotation up:disease ?disease .\n"
query += "        ?disease rdfs:comment ?diseaseAnnotationText\n"
if (diseaseAnnotation != ''):
    query += "        filter( regex(str(?diseaseAnnotationText), " + \
        "'" + diseaseAnnotation + "'" + ", \"i\" )) .\n"
if (diseaseAnnotation == ''):
    query += "        }\n"

query += "\n"

if (domainName == ''):
    query += "        OPTIONAL {\n"
query += "        ?protein up:domain ?domain .\n"
query += "        ?domain up:recommendedName ?domainName .\n"
query += "        ?domainName up:fullName ?domainFullName .\n"
if (domainName != ''):
    query += "        filter( regex(str(?domainFullName), " + \
        "'" + domainName + "'" + ", \"i\" )) .\n"

```

```

if (domainName == ''):
    query += "        }\n"

query += "\n"

if (similarityAnnotation == ''):
    query += "        OPTIONAL {\n"
query += "        ?protein up:annotation ?similarityAnnotation .\n"
query += "        ?similarityAnnotation a up:Similarity_Annotation .\n"
query += "        ?similarityAnnotation rdfs:comment ?similarityAnnotationText .\n"
if (similarityAnnotation != ''):
    query += "        filter( regex(str(?similarityAnnotationText), " + \
        "'" + similarityAnnotation + "'" + ", \"i\" )) .\n"
if (similarityAnnotation == ''):
    query += "        }\n"

query += "\n"

if (locationAnnotation == ''):
    query += "        OPTIONAL {\n"
query += "        ?protein up:annotation ?locationAnnotation .\n"
query += "        ?locationAnnotation a up:Subcellular_Location_Annotation .\n"
query += "        ?locationAnnotation up:locatedIn ?location .\n"
query += "        ?location up:cellularComponent ?cellComponent .\n"
query += "        ?cellComponent rdfs:comment ?locationAnnotationText .\n"

```

```

if (locationAnnotation != ''):
    query += "        filter( regex(str(?locationAnnotationText), " + \
        "'" + locationAnnotation + "'" + ", \"i\" )) .\n"
if (locationAnnotation == ''):
    query += "        }\n"

query += "\n"

if (functionAnnotation == ''):
    query += "        OPTIONAL {\n"
query += "        ?protein up:annotation ?functionAnnotation .\n"
query += "        ?functionAnnotation a up:Function_Annotation .\n"
query += "        ?functionAnnotation rdfs:comment ?functionAnnotationText\n"
if (functionAnnotation != ''):
    query += "        filter( regex(str(?functionAnnotationText), " + \
        "'" + functionAnnotation + "'" + ", \"i\" )) .\n"
if (functionAnnotation == ''):
    query += "        }\n"

query += "\n"

if (pharmaceuticalAnnotation == ''):
    query += "        OPTIONAL {\n"
query += "        ?protein up:annotation ?pharmaceuticalAnnotation .\n"
query += "        ?pharmaceuticalAnnotation a up:Pharmaceutical_Annotation\n"

```

```

query += "          ?pharmaceuticalAnnotation rdfs:comment ?pharmaceuticalA
if (pharmaceuticalAnnotation != ''):
    query += "          filter( regex(str(?pharmaceuticalAnnotationText), '
        ''' + pharmaceuticalAnnotation + ''' + ",\\"i\\" )) .\n"
if (pharmaceuticalAnnotation == ''):
    query += "          }\n"
query += "}\n"
#query += "limit 30\n"

return query

```

```

def printResults(json, output):
    # Abrir fichero para escritura
    fileRes = open(output, 'w')
    # Imprimir cabecera
    for param in paramList:
        fileRes.write(param + "\t")
    fileRes.write("\n")

    # El formato json se puede recorrer de esta manera
    # para ir obteniendo valores de la respuesta.
    for entrada in json["results"]["bindings"]:
        for param in paramList:
            if (entrada.get(param)):

```

```

        fileRes.write(entrada.get(param) ["value"] + "\t")
    else:
        fileRes.write("\t")
    fileRes.write("\n")
fileRes.close()

```

```

def sparqlwrap(
    proteinId,
    proteinName,
    geneName,
    organismName,
    diseaseAnnotation,
    domainName,
    similarityAnnotation,
    locationAnnotation,
    functionAnnotation,
    pharmaceuticalAnnotation,
    output):

```

```

query = buildQuery(
    proteinId,
    proteinName,
    geneName,
    organismName,

```

```

        diseaseAnnotation,
        domainName,
        similarityAnnotation,
        locationAnnotation,
        functionAnnotation,
        pharmaceuticalAnnotation)
print query

# Creamos un objeto del tipo SPARQLWrapper indicando en que
# direccion esta el servicio que recibe consultas en sparql
# y responde a estas.
sparql = SPARQLWrapper('http://sparql.uniprot.org/sparql')

# Especificamos la consulta que queremos hacer en sparql.
sparql.setQuery(query)

# Indicamos en que formato queremos que nos devuelva
# los resultados de la consulta. Puede ser json, xml,
# rfd, turtle... Simplemente son distintos formatos
# para representar los datos en ficheros de texto.
sparql.setReturnFormat(JSON)

# Esta es la instruccion que realiza la consulta a
# uniprot. Devuelve un objeto de python que hay que
# tratar.

```



```

print "Ejecutando query"
results = sparql.query()

# Con esto, convertimos el objeto devuelto por
# el servicio al formato que especificamos antes.
# En este caso, json.
print "Convirtiendo a json"
json = results.convert()
print "Fin conversion a json"

# Dentro de la variable results tenemos informacion
# (metadatos) de lo que ha devuelto el servidor de
# uniprot.
print results.info()

# Imprimir resultados
printResults(json, output)

# Obtener parametros de la linea de comandos.
proteinId = sys.argv[1]
proteinName = sys.argv[2]
geneName = sys.argv[3]
organismName = sys.argv[4]
diseaseAnnotation = sys.argv[5]

```

```
domainName = sys.argv[6]
similarityAnnotation = sys.argv[7]
locationAnnotation = sys.argv[8]
functionAnnotation = sys.argv[9]
pharmaceuticalAnnotation = sys.argv[10]
output = sys.argv[11]

# Llamada a la funcion que realiza la consulta.
sparqlwrap(
    proteinId,
    proteinName,
    geneName,
    organismName,
    diseaseAnnotation,
    domainName,
    similarityAnnotation,
    locationAnnotation,
    functionAnnotation,
    pharmaceuticalAnnotation,
    output)
```

## Appendix 2

```
<tool id="sparql_uniprot" name="Uniprot Sparql" version="0.1.0">
  <description>Perform a search into uniprot database using its sparql inte
  <requirements>
    <requirement type="binary">python</requirement>
    <requirement type="package" version="1.7.6">SPARQLWrapper</requirement>
  </requirements>
  <command interpreter="python">
    sparql_uniprot.py "${protein}" "${proteinName}"
    "${geneName}" "${organismName}"
    "${diseaseAnnotation}" "${domainName}"
    "${similarityAnnotation}" "${locationAnnotation}"
    "${functionAnnotation}" "${pharmaceuticalAnnotation}"
    "${output}"
  </command>
  <inputs>
    <param name="protein" type="text" label="Uniprot Identifier"/>

    <param name="proteinName" type="text" label="Protein Name">
      <!-- sanitizer para que nos deje usar el caracter $ -->
      <sanitizer sanitize="False"/>
    </param>

    <param name="geneName" type="text" label="Gene Name">
```

```

        <sanitizer sanitize="False"/>
</param>

<param name="organismName" type="text" label="Organism">
    <sanitizer sanitize="False"/>
</param>

<param name="diseaseAnnotation" type="text" label="Disease Annotation">
    <sanitizer sanitize="False"/>
</param>

<param name="domainName" type="text" label="Domain">
    <sanitizer sanitize="False"/>
</param>

<param name="similarityAnnotation" type="text" label="Similarity Annotation">
    <sanitizer sanitize="False"/>
</param>

<param name="locationAnnotation" type="text" label="Location Annotation">
    <sanitizer sanitize="False"/>
</param>

<param name="functionAnnotation" type="text" label="Function Annotation">
    <sanitizer sanitize="False"/>

```

```

</param>

<param name="pharmaceuticalAnnotation" type="text" label="Pharmaceutical
    <sanitizer sanitize="False"/>
</param>
</inputs>

<outputs>
    <data format="tabular" name="output" />
</outputs>

<help>
With this tool you can search proteins using regular expressions
by the following fields:uniprot identifier, protein name, gene name,
organism name, disease annotation, domain name,similarity annotation,
location annotation, function annotation or pharmaceutical annotation.
You can fill the fields you want and the tool will return all the data
found in uniprot by filling the other fields. Results will be stored
in a tabular format. If you want to search the exact match of a field
you should use ^parameter$, for example ^insulin$.

</help>

</tool>

```

## Appendix 3

```
<?xml version="1.0"?>
<tool_dependency>
  <package name="SPARQLWrapper" version="1.7.6">
    <readme>
      Install SPARQLWrapper via virtualenv.
    </readme>
    <install version="1.0">
      <actions>
        <action type="setup_virtualenv">SPARQLWrapper==1.7.6</action>
      </actions>
    </install>
  </package>
</tool_dependency>
```