PLUS (Programmatic-python login for Urika SPARQL-Endpoint) 2.3 Documentation

Brown, Tyler C. < browntc@ornl.gov> Sukumar, Sreenivas R. <sukumarsr@ornl.gov>

Included Files:

\UManage.py	Version	
\SPARQLWrapper-1.5.2-py2.7.egg	1.5.2	Python dependency - SPARQL endpoint interface - This is a custom version. Be sure to install the package included within.
\test.rq		Test rq file containing a simple sparql test query
\logintest.py		Example python execution using both solarpy and SPARQLWrapper
\solarpy\profile\profile.py		Dependencies required by PLUS when using solarpy as your login method. There should be no need to edit these files for normal operation of PLUS. These should stay with your code (including distribution copies) in the structure of the distribution.
\solarpy\profile\profile.rq		
\solarpy\yarc\BlockTemplate-test.rq		
\solarpy\yarc\BlockTemplate.py		
\solarpy\yarc\common.rq		
\solarpy\yarc\Graphy.py		
\solarpy\yarc\RdfHelp.py		
\solarpy\yarc\rdfs-namespace.xml		
\solarpy\yarc\solar.rq		
\solarpy\yarc\Tracking.py		
\solarpy\yarc\Tracking.rq		
\solarpy\yarc\urika.py		
\solarpy\yarc\yarcutils.py		
\solarpy\lib\easy-install.pth		Python dependencies required by solarpy. These will need to be installed using easy_install, copying them into your local python includes directory, or other method of including dependencies.
\solarpy\lib\isodate-0.4.9-py2.6.egg	0.4.9	
\solarpy\lib\ordereddict-1.1-py2.6.egg	1.1	
\solarpy\lib\pkg_resources.py		
\solarpy\lib\pyparsing-2.0.1-py2.6.egg	2.01	
\solarpy\lib\rdflib-4.1.0-py2.6.egg	4.1.0	
\solarpy\lib\requests-2.2.1-py2.6.egg	2.2.1	
\solarpy\lib\site.py		

Version Info: Python 2.7 Fuseki 0.2.8 solarpy-1d30d56 (Custom Version)

Other Dependencies:

simplejson	Simple, fast, extensible JSON encoder/decoder for Python	https://pypi.python.org/pypi/simplejson
RDFLib	https://github.com/RDFLib/rdflib	https://github.com/RDFLib/rdflib

The above may or may not be installed with your distribution of python. Make sure these are installed in the python includes directory, if not obtain a copy of the dependency or otherwise an ImportError will be produced.

Change Log:

Version 2.3	7/23/2014	Determines the running database after login.
Version 2.2	7/11/2014	Added support to check if an existing connection to urika is active NOTE: Function call has changed!
Version 2.1	7/1/2014	Added query update functions and allowed strings
Version 2.0	6/30/2014	Added local Fuseki Support and allowed for queries to be in string format
Version 1.0	5/25/2014	Original: Allowed for queries using SPARQLWrapper or Solarpy.

Getting Started:

PLUS provides most of the framework for creating programs that use URiKa or local Fuseki environment SPARQL Endpoints. It creates a secure connection to URiKa using Yarc Data's solarpy or SPARQLWrapper. Additionally, bypasses authentication when connecting to a Fuskei endpoint – as this is not needed in a local environment. Once a connection is made, a "pipeline" is created in which multiple queries can be executed and the results returned without the need to constantly login. The connection to Urika is closed once your python code is terminated.

Requirements:

Python 2.7 or greater
Fuseki 0.2.8 or greater (if using as a local connection)
Access to Urika (If using as an external connection)
All dependencies installed (mentioned above)
RSA Token (if using as an external connection to urika)

RSA Token:

~~You may disable your RSA token if you run multiple instances of GAM (web browser interface) or simultaneous executions of your code or any combination, thereof. If your token becomes disabled you will not be able to login to the GAM or execute any code. Contact the solutions center to have the issue corrected.~~ AS OF VERSION 2.2 THIS IS NO LONGER TRUE. Version 2.2 will check for an existing urika connection using a netstat of current connected IPs and looks for 172.30.246.4, which is urika's IP address.

Includes: – The two options for importing PLUS into your program.

```
11 from PLUS import UrikaLogin
12 from PLUS import FusekiLogin
```

Connection Class:

NOTE: This is for version 2.3 ONLY! See your versions documentation for connection class calls.

```
11  conn = FusekiLogin(database)
12  conn = UrikaLogin(user,password)
```

With optional parameters:

```
11  conn = FusekiLogin(database)
12  conn = UrikaLogin(username,password, db = "database", method = "sparqlwrapper")
```

- *username* – This is the username selected to use URiKa – It may or may not be the same as your UCAMS username
- *password* – pin+RSAToken – the password is never the same, so it would be a good idea to prompt in runtime for this variable.
- *database* – Currently running database on fuseki endpoint

Optional parameters:

- *db* – Current database running on urika endpoint (default = None) will detect current db runtime.
- *method* – ["solarpy" | "sparqlwrapper"] (default = "solarpy") passed a string object

Query data (using Solarpy):

There are two ways to query the data using solarpy, one would be parsing .rq file using the `conn.urika.bloct()` function, the other would be passing a string to the `conn.urika.query()` function. Both are shown below.

- Query using an .rq file:

```
38  query = conn.urika.blockt( __file__, "test.rq")
39  results = conn.urika.query( func, file = query)
40  print results
```

```
38  query = conn.urika.blockt( __file__, "test.rq")
39  results = conn.urika.query( func, file = query, accept = "csv")
40  print results
```

- Query using a string:

```
34  query = "SELECT * WHERE { ?s ?p ?o } LIMIT 10"
35  results = conn.urika.query( func, string = query)
36  print results
```

```
34  query = "SELECT * WHERE { ?s ?p ?o } LIMIT 10"
35  results = conn.urika.query( func, string = query, accept = "csv")
36  print results
```

  - `conn.urika.blockt()` – returns an instance of the parsed .rq file
  - `conn.urika.query()` – returns the results in json unless accept is passed.

  Required Parameters:

  - "test.rq" – the absolute or relative path to the SPARQL query. (if using a .rq file)
  - func – this is the name of the your query – it is used by tracking within solarpy for debugging purposes. It is passed as a string.

- query – this should be a parsed rq file by calling conn.urika.blockt( ) or a string.
    - file = query– this should be used when passing a file
    - string = query – this should be used when passing a string

Optional Parameters:

- blockt (default = None) this is used if other templates are using when passing files.
- accept – ["csv" | "json"| "xml"] (default = "json") this is the format in which you would like the results returned

Query data (using sparqlwrapper):

```
40          conn.sparql.setQuery("SELECT * WHERE { ?s ?p ?o } LIMIT 10")
41          conn.sparql.method = "GET"
42          results = conn.sparql.query().convert()
43          for result in results["results"]["bindings"]:
44              print(result)
```

NOTE: An .ru file can be used. Simply open and read the file and store in string variable, pass as string. Also, results are returned in JSON format.

Update Post (using Solarpy):

- Using a string:

```
81          update = """PREFIX dc: <http://purl.org/dc/elements/1.1/>
82  INSERT DATA
83  { <http://example/book3> dc:title    "A new book" ;
84                           dc:creator  "A.N.Other" .
85  }"""
86
87          conn.urika.update(func, string  = update)
```

- Using an .ru file:

```
87          update = conn.urika.blockt( __file__, "test.ru")
88          conn.urika.update(func, file = update)
```

    - `conn.urika.blockt()` – returns an instance of the parsed .rq file
    - `conn.urika.query()` – executes the update, usually no value returned

Required Parameters:

- func – this is the name of the your query – it is used by tracking within solarpy for debugging purposes. It is passed as a string.
- "test.ru" – the absolute or relative path to the SPARQL query (if using a .ru file)
- update – this is the instance of the parsed .rq file returned by `conn.urika.bloct()` or a string.
    - file = update – when using a .rq file
    - string = update – when passing a string

Optional Parameters:

- blockt – (default = `None`) – this is used if other templates are using when passing files.
- memfree – (default = None) – the ratio of serv memory expected to be needed to complete the query/update (example 0.25 is 500G on a 2T urika)

Update (using SPARQLWrapper):

```
        update = """PREFIX dc: <http://purl.org/dc/elements/1.1/>
INSERT DATA
{ <http://example/book3> dc:title    "A new book" ;
                         dc:creator  "A.N.Other" .
}"""
        conn.sparql.setQuery(update)
        conn.sparql.method = "POST"
        results = conn.sparql.query()
```

NOTE: An .ru file can be used. Simply open and read the file and store in string variable, pass as string.

Troubleshooting:

• HTTPError: 404 Client Error: Not Found • EndPointNotFound: EndPointNotFound:	• Make sure you have passed the correct database. NOTE: It must be the database that is currently running. Check GAM (Web browser interface) to ensure which database is selected.
• HTTPError: 403 Client Error: Forbidden	• This is because the username and password provided does not currently have permissions to access the selected database. Someone with permissions to that database must add your username to the list. • This may also be raised when the database passed is not currently running. Check the database that is running in the GAM (web browser interface)
• Exception: urika login failed • HTTPError: HTTP Error 401: Unauthorized	• The username and password you provided is not correct. If you are certain that your username and password is correct – your RSA token may have been disabled and will need to be reset by the solutions center.
• HTTPError: 400 Client Error: Bad Request	• Check the query and try again, this usually indicates that the query is not valid sparql.

TCB, ORNL 7/23/2014