# tuProlog SPARQL QUERY Library
# USER GUIDE

(Giuliano Santandrea, december 2011)

DEPENDENCIES:
this library uses the Jena open source framework (in particular the ARQ query processor). To use this library in your project you need to include the ARQ library from sourceforge
(http://sourceforge.net/projects/jena/files/ARQ/ )

GOAL:
with this library you are able to query a RDF document, in particular to make SPARQL queries of SELECT or ASK type. For SELECT queries you can iterate over the solutions and retrieve variable bindings of each solution.

Note: after the execution of a new query the solutions asserted of the previous queries are removed.

API SYNTAX:

`:-load_library('progetto.SPARQLLibrary').`
Put this in your prolog Theory to load the SPARQL Library.

`setFilePath('dbRDF.rdf').`
This prolog goal sets the file to query.

`ask { triples }.`
This prolog goal makes a Sparql query of Ask type. You have to select a file with `setFilePath` before making this goal. "triples" must be a comma separated list of triples, where each triple is a prolog term of the form `triple(sub,pred,obj)` , where sub, pred and obj can be :

- an **URI**, for example
  `uri('http://www.w3.org/2001/vcard-rdf/3.0#Family')`
- a **literal**, for example
  `lit('Smith')`
- a **variable**, for example
  `var(s)`

For example this SPARQL ASK query:

```
ASK {
      ?s  <http://www.w3.org/2001/vcard-rdf/3.0#Family> "Smith"
      }
```

becomes:

```
ask { triple(var(s), uri('http://www.w3.org/2001/vcard-rdf/3.0#Family'),
lit('Smith')) }.
```

The prolog goal fails if the SPARQL Ask query fails.

`select [outputVars] where { triples }.`
This prolog goal makes a Sparql query of Select type. You have to select a file with `setFilePath` before making this goal. "outputVars" must be a comma separated list of prolog terms . "triples" must be a comma separated list of triples, where each triple is a prolog term of the form
triple(sub,pred,obj) (see the explanation for the Ask query).

For example this SPARQL Select query:

```
SELECT ?l WHERE {
                  ?s <http://www.w3.org/2001/vcard-rdf/3.0#Family>  "Smith",
                  ?s <http://www.w3.org/2001/vcard-rdf/3.0#Given> ?l
                 }
```

becomes:

```
select [l] where { triple(var(s),uri('http://www.w3.org/2001/vcard-
rdf/3.0#Family'),lit('Smith')) , triple(var(s),
uri('http://www.w3.org/2001/vcard-rdf/3.0#Given'),var(l))}.
```

As a consequence of the execution of this goal a prolog Term is asserted in the theory containing all the solutions and the bindings for each solution. For example:

```
results([
[outputVarValue(l,'John',literal), outputVarValue(p,'Paul',literal)],
[outputVarValue(l,'Rebecca',literal), outputVarValue(p,'Marc',literal)]
          ])
```

You can explore this term manually or you can use the prolog goal **nextSol**. This goal extract the first solution from the list and, for each binding, asserts the corresponding prolog terms. For example, in the previous case, executing nextSol causes the following prolog terms to be asserted:

```
outputVarValue(l,'John',literal)
outputVarValue(p,'Paul',literal)
```

The **nextSol** goal fails if the term results(..) contains an empty list.
NOTE: execting nextSol will remove from the theory every `outputVarValue(..)` previously asserted.

`nextSol`
see the Select query explanation.

`select [outputVars] where { triples } into varSave.`
See the explanation for the select query. The only difference is that the results are asserted into varSave(....) .