

easy RQL (eRQL)

Karsten Tolle and Fabian Wleklinski (July 2004)

To make usage of the source information kept in RDF-S3, we further developed the query language *easy RQL* (*eRQL*). The name was given, because eRQL was originally designed on top of the *RDF Query Language* (*RQL*). The implementation that comes with RDF-S3 goes a different way and accesses directly the RDF-S3 data for performance reasons. The query results of eRQL contain the fitting triples plus their context node (the source information). One main design-goal for this query language was to be as simple as possible. The user of the query language should be able to create queries without knowing the schema or the way the data is stored. The idea was that the user can simply enter search strings, similar to current query engines he already knows (like *Google*) and retrieves reasonable results. This goal was achieved by the possibility of *one-word-queries*, e.g. "Picasso", that can be combined using boolean operators. The most valid Google-queries are valid eRQL-Queries as well. In addition eRQL supports three different modes, namely *Statement-Mode*, *POI-Mode* and *Document-Mode*:

The *Statement-Mode* corresponds to queries of existing RDF query languages and returns the triples plus their context node that fit the query. This mode is activated by enclosing the query into brackets "[...]".

The *POI-Mode*, which stands for *Point Of Interest-Mode*, includes for each triple fitting the query also the surrounding triple up to the distance defined via the query. This way we try to overcome the problem that many information and returned results are only interpretable by knowing their surrounding triples (hence the *internal context*). The POI-Mode is default but can be explicitly activated by enclosing the query into braces "{...}". In case the included neighborhood should be extended the braces can be used cumulative, e.g. "{{Picasso}}" includes all statements containing 'Picasso' plus their neighborhood with a distance of two (thus the neighborhood's neighborhood). It is the default modus which means the queries "Picasso" and "{Picasso}" are equal. Alternatively, for a more easy way to write the symbol "~" can be used, e.g. "~Picasso" ("~" can be read as "not exactly"). Please note that the default setting for POI in this case is still valid, which means that "~Picasso" is equal to "~{Picasso}" which is equal to "{{Picasso}}". The "~" can also be used cumulative. In that sense "~~~Picasso" is equal to "~~~{Picasso}" which is equal to "{{{{Picasso}}}}". and will return the found statements with a neighborhood of a distance of four.

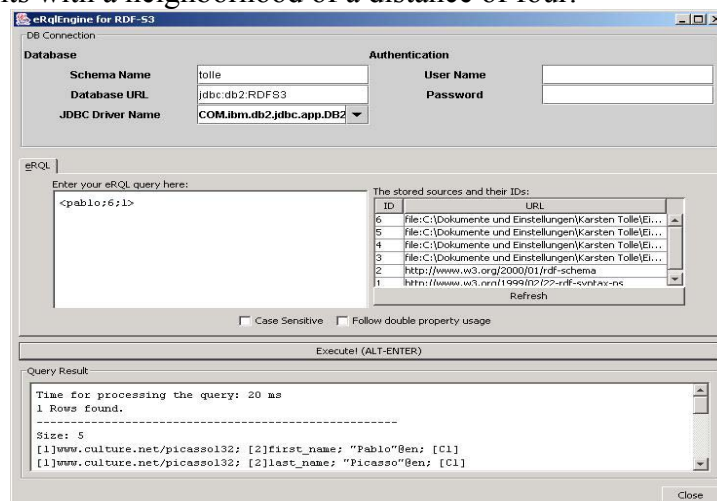


Figure 1 Screenshot of the GUI for retrieving data from RDF-S3 using eRQL.

In the *Document-Mode* the source information is not only returned in the result but also used for querying itself. It needs three inputs, an eRQL query q , a list of source IDs *source-list* (separated by comma) and b which must be either 0 or 1. The mode is active by enclosing these three into a left angle bracket and a right angle bracket and by separate them with semicolons: " $\langle q; source-list; b \rangle$ ". Depending on b , the query q is either evaluated only on the given set of sources ($b = 1$), or it is evaluated on all sources stored in RDF-S3 except those specified in the *source-list* ($b = 0$). Using this mode the user can select or unselect sources he does or doesn't trust. As shown in Figure 1 the user can retrieve the source ID information from the table on the right side in the middle of the GUI.

Figure 1 shows the query " $\langle pablo;6;1 \rangle$ ". This means we are searching only for those triples contained in the source with the ID 6 containing the string "pablo" in either one of their resource URIs or their literal value. Since the POI-Mode is default, also the surrounding triples with a distance of 1 will be included into the result. The complete result for the given query is shown in Figure 2. After denoting the processing time for the query, we read in the result: *1 Row found.* This means there is just one triple fitting the query, we call it a *hit*. Afterwards the result returns each hit plus its internal context with a distance as defined in the query (one in this case). The hit itself will always be in first position. Figure 2 also shows that for each triple also the source information is returned.

```

Time for processing the query: 20 ms
1 Rows found. — number of hits
-----
Size: 5
[1]www.culture.net/picasso132; [2]first_name; "Pablo"@en; [C1]
[1]www.culture.net/picasso132; [2]last_name; "Picasso"@en; [C1]
[1]www.culture.net/picasso132; [2]paints; [3]woman.qti; [C1]
[1]www.culture.net/picasso132; [2]paints; [3]guernica.jpg; [C1]
[1]www.culture.net/picasso132; [4]type; [2]Cubist; [C1]
-----
Namespaces:
[1] file:C:\Dokumente und Einstellungen\Karsten Tolle\Eigene
Dateien\Uni\RDF\examples\RDF_examples\culture_data2.rdf#
[2] file:D:\tmp_local_users\alexaki\RDF_vrp2_5\RDF_examples\culture.rdf#
[3] http://www.museum.es/
[4] http://www.w3.org/1999/02/22-rdf-syntax-ns#
Sources:
[C1] file:C:\Dokumente und Einstellungen\Karsten Tolle\Eigene
Dateien\Uni\RDF\examples\RDF_examples\culture_data2.rdf

```

Figure 2 Result for the query $\langle pablo;6;1 \rangle$, given as the output of the eRQL GUI of RDF-S3. The ID 6 used in the query is the internal ID from RDF-S3 for the file culture_data2.rdf.

Additional new developed features of eRQL are:

- The possibility to choose between a case sensitive and a non case sensitive search.
- A one-word-query q can be restricted to be evaluated only on resources (by using: $res(q)$) or on literals (by using double quotes: " q ").
- Wildcards "*" and "?" can be used, whereby "*" stands for any string and "?" for exact one character.