Apache Jena

- Home
- Download
- Learn
  - Tutorials
  - Overview
  - RDF core API tutorial
  - SPARQL tutorial
  - Manipulating SPARQL using ARQ
  - Using Jena with Eclipse
  - How-To's
  -
  - References
  - Overview
  - Javadoc
  - RDF API
  - RDF I/O
  - ARQ (SPARQL)
  - RDF Connection - SPARQL API
  - Elephas - tools for RDF on Hadoop
  - Text Search
  - TDB
  - SDB
  - SPARQL over JDBC
  - Fuseki
  - Permissions
  - Assembler
  - Ontology API
  - Inference API
  - Command-line tools
  - Extras
- Javadoc
  - Jena Core
  - ARQ
  - TDB
  - Fuseki
  - Elephas
  - Text Search
  - Spatial Search
  - Permissions
  - JDBC
  - All Javadoc
- Ask
- Get involved
  - Contribute
  - Report a bug
  -
  - Project
  - About Jena
  - Roadmap
  - Architecture
  - Project team

SPARQL Tutorial - Datasets

This section covers RDF Datasets - an RDF Dataset is the unit that is queried by a SPARQL query. It consists of a default graph, and a number of named graphs.

Querying datasets

The graph matching operation (basic patterns, `OPTIONAL`s, and `UNION`s) work on one RDF graph. This starts out being the default graph of the dataset but it can be changed by the `GRAPH` keyword.

```
GRAPH uri { ... pattern ... }

GRAPH var { ... pattern ... }
```

If a URI is given, the pattern will be matched against the graph in the dataset with that name - if there isn't one, the `GRAPH` clause fails to match at all.

If a variable is given, all the named graphs (not the default graph) are tried. The variable may be used elsewhere so that if, during execution, its value is already known for a solution, only the specific named graph is tried.

Example Data

An RDF dataset can take a variety of forms. Two common setups are to have the default graph being the union (the RDF merge) of all the named graphs or to have the default graph be an inventory of the named graphs (where they came from, when they were read etc). There are no limitations - one graph can be included twice under different names, or some graphs may share triples with others.

In the examples below we will use the following dataset that might occur for an RDF aggregator of book details:

Default graph (ds-dft.ttl):

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
@prefix xsd: <http://www.w3.org/2001/XMLSchema#> .

<ds-ng-1.ttl> dc:date "2005-07-14T03:18:56+0100"^^xsd:dateTime .
<ds-ng-2.ttl> dc:date "2005-09-22T05:53:05+0100"^^xsd:dateTime .
```

Named graph (ds-ng-1.ttl):

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .
```

```
[] dc:title "Harry Potter and the Philospher's Stone" .
[] dc:title "Harry Potter and the Chamber of Secrets" .
```

Named graph (ds-ng-2.ttl):

```
@prefix dc: <http://purl.org/dc/elements/1.1/> .

[] dc:title "Harry Potter and the Sorcerer's Stone" .
[] dc:title "Harry Potter and the Chamber of Secrets" .
```

That is, we have two small graphs describing some books, and we have a default graph which records when these graphs were last read.

Queries can be run with the command line application (this would be all one line):

```
java -cp ... arq.sparql
    --graph ds-dft.ttl --namedgraph ds-ng-1.ttl --namedgraph ds-ng-2.ttl
    --query query file
```

Datasets don't have to be created just for the lifetime of the query. They can be created and stored in a database, as would be more usual for an aggregator application.

Accessing the Dataset

The first example just accesses the default graph (q-ds-1.rq):

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX : <.>

SELECT *
{ ?s ?p ?o }
```

(The "`PREFIX : <.>`" just helps format the output)

```
-------------------------------------------------------------------
| s           | p       | o                                     |
===================================================================
| :ds-ng-2.ttl | dc:date | "2005-09-22T05:53:05+01:00"^^xsd:dateTime |
| :ds-ng-1.ttl | dc:date | "2005-07-14T03:18:56+01:00"^^xsd:dateTime |
-------------------------------------------------------------------
```

This is the default graph only - nothing from the named graphs because they aren't queried unless explicitly indicated via `GRAPH`.

We can query for all triples by querying the default graph and the named graphs (q-ds-2.rq):

```
PREFIX  xsd:    <http://www.w3.org/2001/XMLSchema#>
PREFIX  dc:     <http://purl.org/dc/elements/1.1/>
PREFIX  :       <.>

SELECT *
{
    { ?s ?p ?o } UNION { GRAPH ?g { ?s ?p ?o } }
}
```

giving:

```
-------------------------------------------------------------------------------
| s           | p       | o                                     | g           |
===============================================================================
```

```
| :ds-ng-2.ttl | dc:date  | "2005-09-22T05:53:05+01:00"^^xsd:dateTime  |              |
| :ds-ng-1.ttl | dc:date  | "2005-07-14T03:18:56+01:00"^^xsd:dateTime  |              |
| _:b0         | dc:title | "Harry Potter and the Sorcerer's Stone"    | :ds-ng-2.ttl |
| _:b1         | dc:title | "Harry Potter and the Chamber of Secrets"  | :ds-ng-2.ttl |
| _:b2         | dc:title | "Harry Potter and the Chamber of Secrets"  | :ds-ng-1.ttl |
| _:b3         | dc:title | "Harry Potter and the Philospher's Stone"  | :ds-ng-1.ttl |
-------------------------------------------------------------------------------------
```

Querying a specific graph

If the application knows the name graph, it can directly ask a query such as finding all the titles in a given graph (q-ds-3.rq):

```
PREFIX dc: <http://purl.org/dc/elements/1.1/>
PREFIX : <.>

SELECT ?title
{
  GRAPH :ds-ng-2.ttl
    { ?b dc:title ?title }
}
```

Results:

```
-------------------------------------------
| title                                   |
===========================================
| "Harry Potter and the Sorcerer's Stone"   |
| "Harry Potter and the Chamber of Secrets" |
-------------------------------------------
```

Querying to find data from graphs that match a pattern

The name of the graphs to be queried can be determined with the query itself. The same process for variables applies whether they are part of a graph pattern or the `GRAPH` form. The query below (q-ds-4.rq) sets a condition on the variable used to select named graphs, based on information in the default graph.

```
PREFIX  xsd:    <http://www.w3.org/2001/XMLSchema#>
PREFIX  dc:     <http://purl.org/dc/elements/1.1/>
PREFIX  :       <.>

SELECT ?date ?title
{
  ?g dc:date ?date . FILTER (?date > "2005-08-01T00:00:00Z"^^xsd:dateTime )
  GRAPH ?g
      { ?b dc:title ?title }
}
```

The results of executing this query on the example dataset are the titles in one of the graphs, the one with the date later than 1 August 2005.

```
-------------------------------------------------------------------------------------
| date                                      | title                                   |
=====================================================================================
| "2005-09-22T05:53:05+01:00"^^xsd:dateTime | "Harry Potter and the Sorcerer's Stone"   |
| "2005-09-22T05:53:05+01:00"^^xsd:dateTime | "Harry Potter and the Chamber of Secrets" |
-------------------------------------------------------------------------------------
```

Describing RDF Datasets - `FROM` and `FROM NAMED`

A query execution can be given the dataset when the execution object is built or it can be described in the query itself. When the details are on the command line, a temporary dataset is created but an application can create datasets and then use them in many queries.

When described in the query, `FROM <url>` is used to identify the contents to be in the default graph. There can be more than one `FROM` clause and the default graph is result of reading each file into the default graph. It is the RDF merge of the individual graphs.

Don't be confused by the fact the default graph is described by one or more URLs in `FROM` clauses. This is where the data is read from, not the name of the graph. As several FROM clauses can be given, the data can be read in from several places but none of them become the graph name.

`FROM NAMED <url>` is used to identify a named graph. The graph is given the name *url* and the data is read from that location. Multiple `FROM NAMED` clauses cause multiple graphs to be added to the dataset.

Note that graphs are loaded with the Jena FileManager which includes the ability to provide alternative locations for files. For example, the query may have `FROM NAMED <http://example/data>`, and the data actually be read from `file:local.rdf`. The name of the graph will be <http://example/data> as in the query.

For example, the query to find all the triples in both default graph and named graphs could be written as (q-ds-5.rq):

```
PREFIX xsd: <http://www.w3.org/2001/XMLSchema#>
PREFIX dc:  <http://purl.org/dc/elements/1.1/>
PREFIX :    <.>

SELECT *
FROM       <ds-dft.ttl>
FROM NAMED <ds-ng-1.ttl>
FROM NAMED <ds-ng-2.ttl>
{
   { ?s ?p ?o } UNION { GRAPH ?g { ?s ?p ?o } }
}
```

Next: results