Apache Jena Framework

Jérôme David
JDev 2017 - Marseille

Présentation en partie issue d'un cours de
Philippe Genoud

Introduction

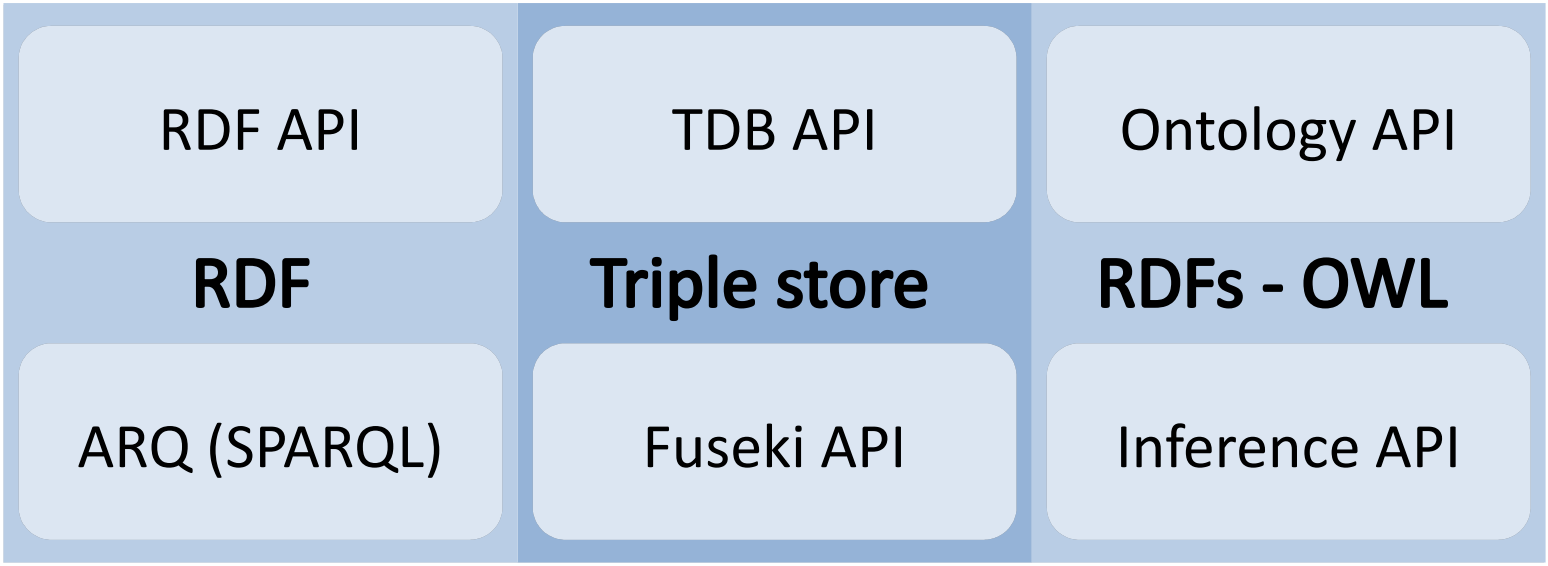
- What is Jena?
  - A free and open source Java framework for building Semantic Web and Linked Data applications.
  - developed at HP-Labs (Bristol-UK)
  - now an apache project : http://jena.apache.org/
    - November 2010: adopted by the Apache Software Foundation (incubation)
    - April 2012: graduated as a top-level project
    - Current version (on December 12, 2016) : 3.1.1

Jena APIs

Core API to create and read RDF graphs and serialize them in standard formats (RDF/XML, Turtle…)

A native high performance triple store to persist RDF data

API for handling OWL and RDFS ontologies to add extra semantics to RDF data

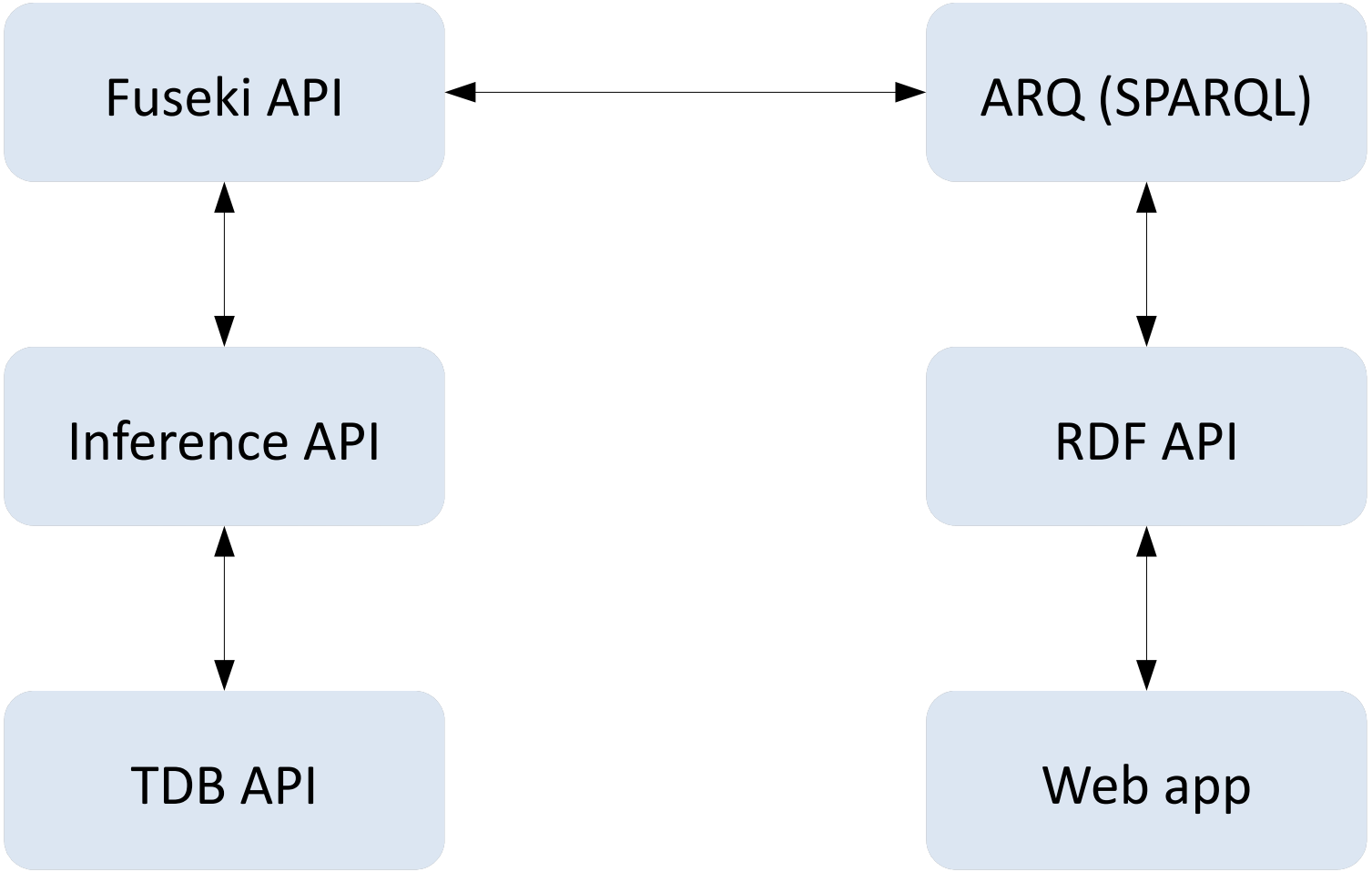


A query engine compliant with the latest SPARQL specification (1.1)

To expose RDF triples as a SPARQL end-point accessible over HTTP. Provides REST-style interaction with RDF data.

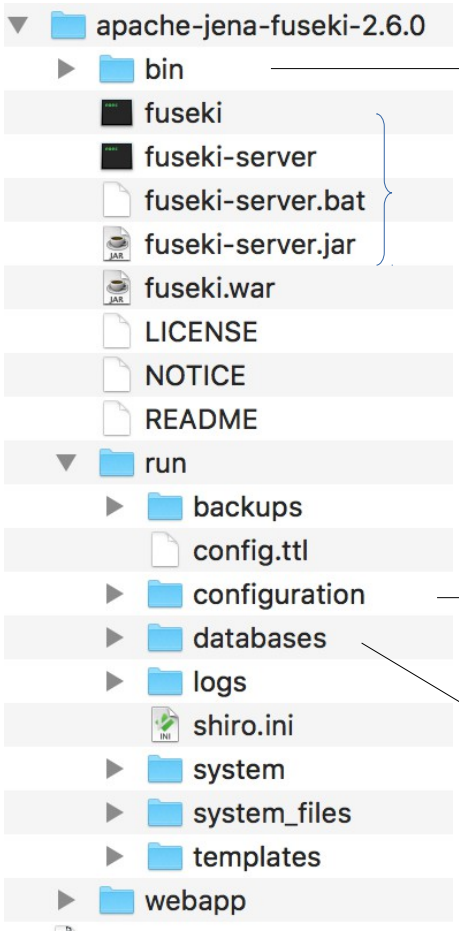
To reason over RDF data to expand and check it. Configure your own inference rules or use the built-in OWL and RDFS reasoners.

What we will try to do ?



Let's try Fuseki

- What the package contains ?



Commands (shell) to access sparql endpoint via HTTP (SOH)

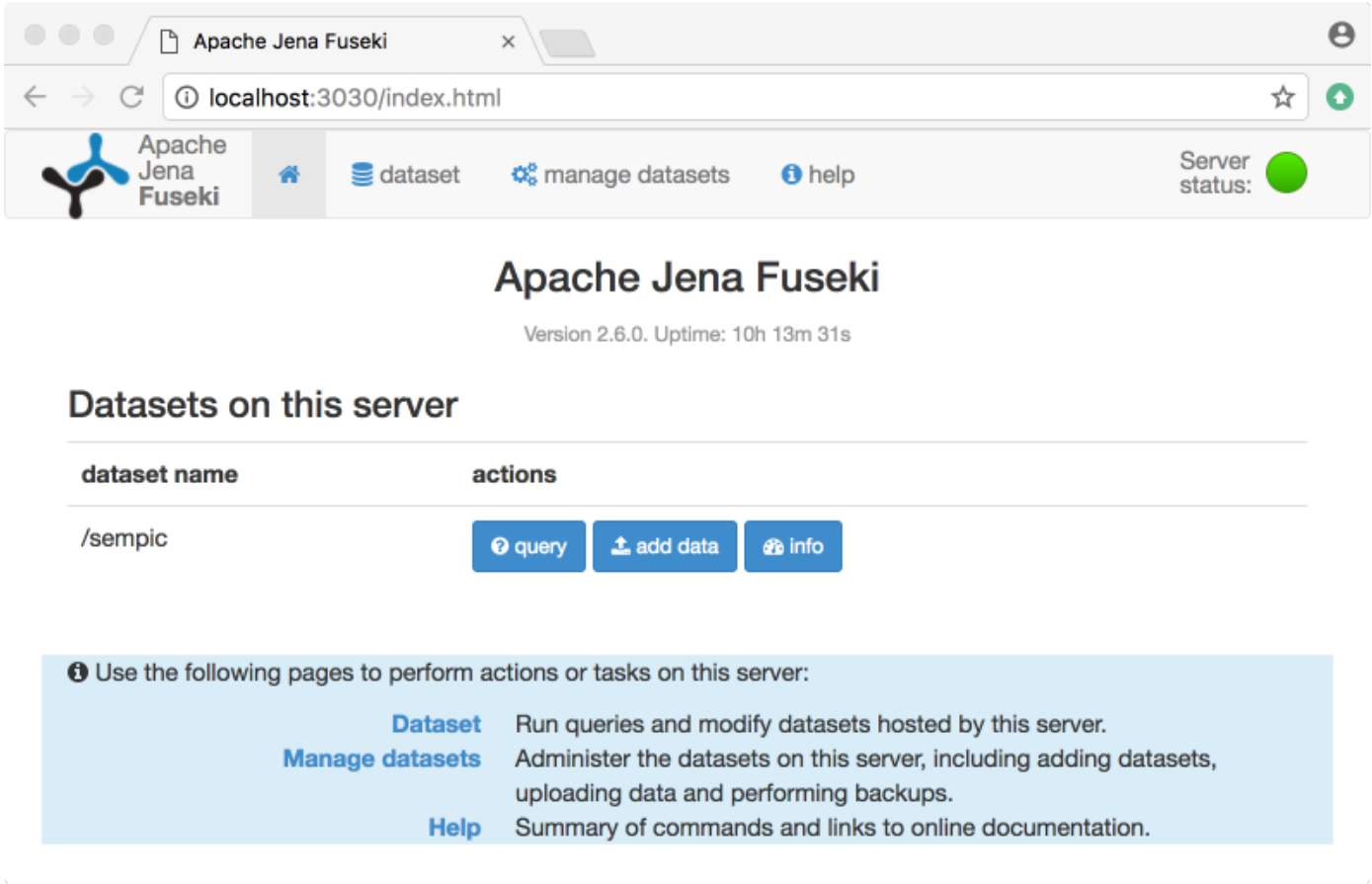
Commands to launch the server

Configuration files for each dataset. Config file is written in RDF :

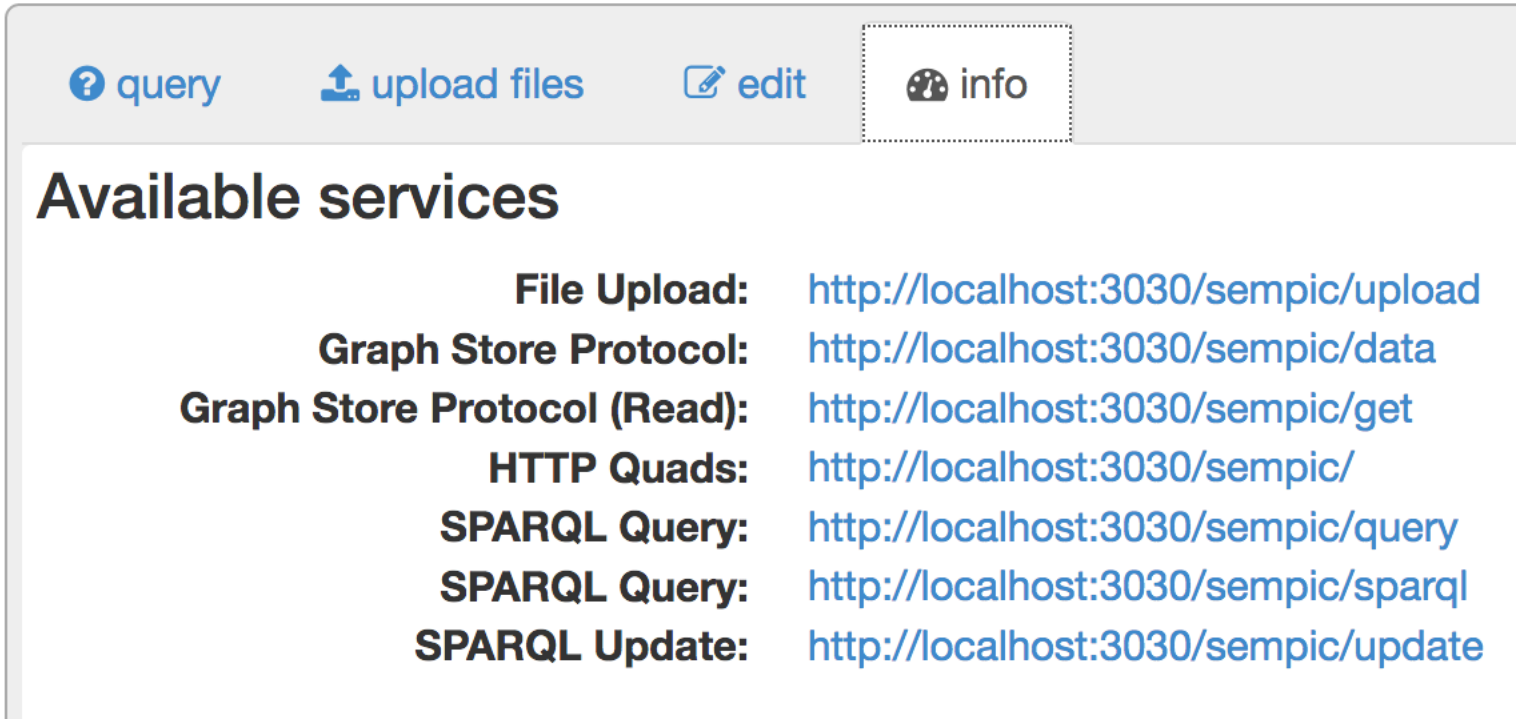
TDB files for each persistent dataset (could be store elsewhere, if needed)

Let's try Fuseki

- Launch the server
  - ./fuseki start or ./fuseki-server start
- And go to http://localhost:3030



Fuseki: services and supported protocols



Querying Fuseki from Jena

- ARQ allows to query RDF from
  - Jena models (i.e. RDF Graph) loaded into memory or via TDB
  - **Remote HTTP endpoint (for instance Fuseki)**
- Jena provides `RDFConnection` API
  - Package `org.apache.jena.rdfconnection`
  - It allows to use : SPARQL Query, SPARQL Update and Graph Store Protocol services
  - It supports transactions
  - Documentation : https://jena.apache.org/documentation/rdfconnection/

RDFConnection - SPARQL SELECT

```
import org.apache.jena.query.*;
import org.apache.jena.rdfconnection.*;

public class ExampleRDFConnection {

    private final static String ENDPOINT= "http://localhost:3030/sempic/";
    public final static String ENDPOINT_QUERY = ENDPOINT+"sparql"; // SPARQL endpoint
    public final static String ENDPOINT_UPDATE = ENDPOINT+"update"; // SPARQL UPDATE endpoint
    public final static String ENDPOINT_GSP = ENDPOINT+"data"; // Graph Store Protocol

    public static void main(String[] args) {

        RDFConnection cnx = RDFConnectionFactory.connect(ENDPOINT_QUERY, ENDPOINT_UPDATE, ENDPOINT_GSP);}

        QueryExecution qe = cnx.query("SELECT DISTINCT ?s WHERE {?s ?p ?o}");
        ResultSet rs = qe.execSelect();
        while (rs.hasNext()) {
            QuerySolution qs = rs.next();
            System.out.println(qs.getResource("s"));
        }

        cnx.close();
    }
}
```

Shorter syntax with Java 8 (lambda expressions)

```
cnx.querySelect("SELECT DISTINCT ?s WHERE {?s ?p ?o}", qs -> {
            System.out.println(qs.getResource("s"));
        });
```

RDFConnection - SPARQL CONSTRUCT

```
import org.apache.jena.query.*;
import org.apache.jena.rdfconnection.*;

public class ExampleRDFConnection {

    private final static String ENDPOINT= "http://localhost:3030/sempic/";
    public final static String ENDPOINT_QUERY = ENDPOINT+"sparql"; // SPARQL endpoint
    public final static String ENDPOINT_UPDATE = ENDPOINT+"update"; // SPARQL UPDATE endpoint
    public final static String ENDPOINT_GSP = ENDPOINT+"data"; // Graph Store Protocol

    public static void main(String[] args) {

        RDFConnection cnx = RDFConnectionFactory.connect(ENDPOINT_QUERY, ENDPOINT_UPDATE, ENDPOINT_GSP);}

        Model res = cnx.queryConstruct("CONSTRUCT {?s a ?t} WHERE {?s a ?t}");
        res.write(System.out);

        cnx.close();
    }
}
```

The type Model is the Jena type for representing RDF graphs

RDFConnection - SPARQL UPDATE

```
import org.apache.jena.query.*;
import org.apache.jena.rdfconnection.*;

public class ExampleRDFConnection {

    private final static String ENDPOINT= "http://localhost:3030/sempic/";
    public final static String ENDPOINT_QUERY = ENDPOINT+"sparql"; // SPARQL endpoint
    public final static String ENDPOINT_UPDATE = ENDPOINT+"update"; // SPARQL UPDATE endpoint
    public final static String ENDPOINT_GSP = ENDPOINT+"data"; // Graph Store Protocol

    public static void main(String[] args) {

        RDFConnection cnx = RDFConnectionFactory.connect(ENDPOINT_QUERY, ENDPOINT_UPDATE, ENDPOINT_GSP);}

        cnx.begin(ReadWrite.WRITE);
        cnx.update("UPDATE INSERT DATA {<http://test.org/dudule> a <ttp://test.org/bidule>");
        cnx.update("UPDATE DELETE WHERE {<http://test.org/dudue> ?p ?o");
         cnx.commit();

        cnx.close();
    }
}
```

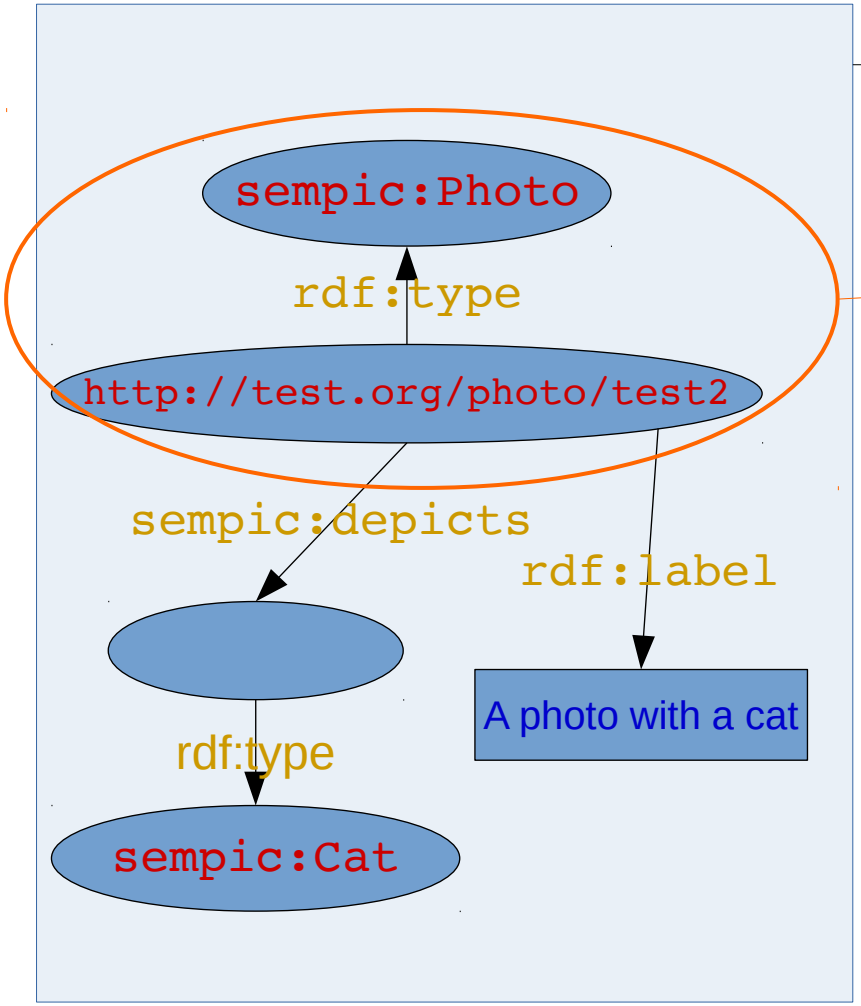
Jena provides transactions.
Use cnx.abort() if you want to abort(rollback) the transaction

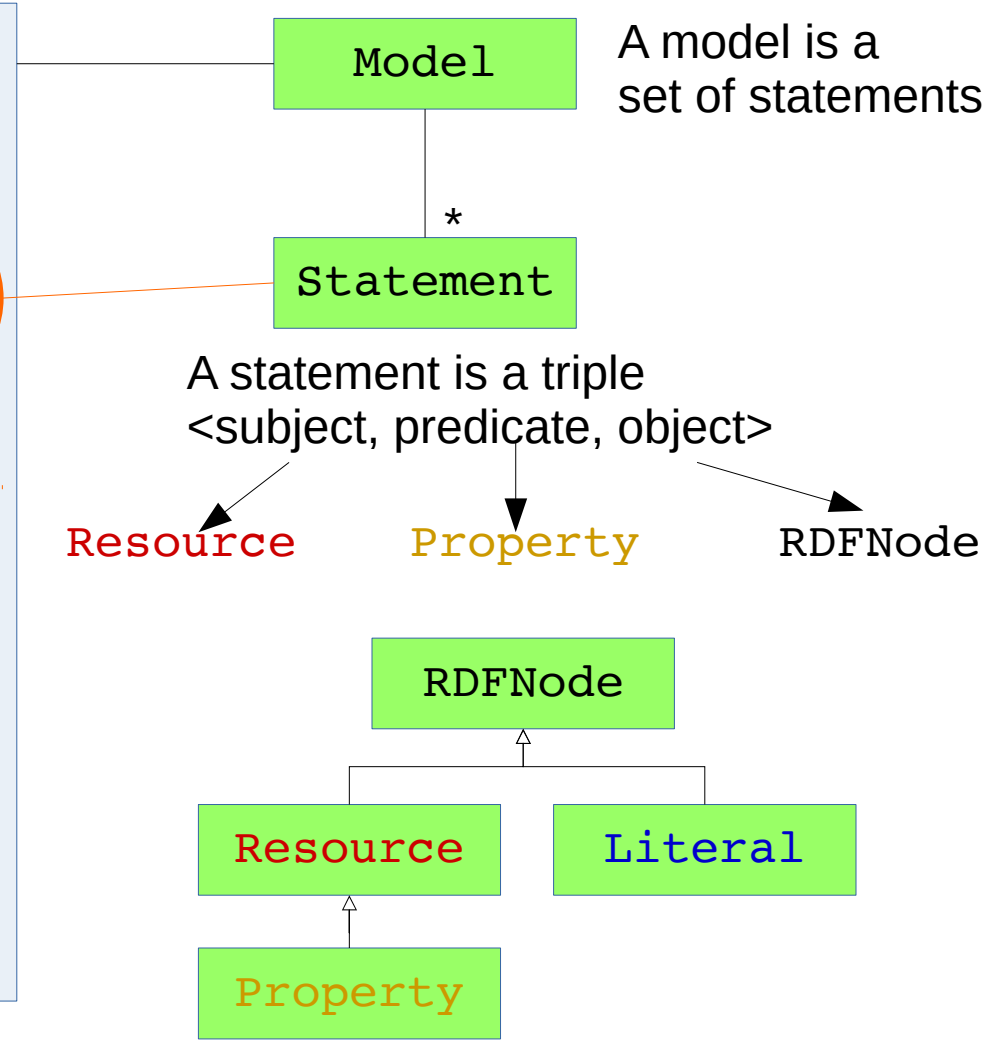
RDFConnection - Graph Store Protocol

```
import org.apache.jena.query.*;
import org.apache.jena.rdfconnection.*;

public class ExampleRDFConnection {

    private final static String ENDPOINT= "http://localhost:3030/sempic/";
    public final static String ENDPOINT_QUERY = ENDPOINT+"sparql"; // SPARQL endpoint
    public final static String ENDPOINT_UPDATE = ENDPOINT+"update"; // SPARQL UPDATE endpoint
    public final static String ENDPOINT_GSP = ENDPOINT+"data"; // Graph Store Protocol

    public static void main(String[] args) {

        RDFConnection cnx = RDFConnectionFactory.connect(ENDPOINT_QUERY, ENDPOINT_UPDATE, ENDPOINT_GSP);}

        cnx.begin(ReadWrite.WRITE);
        cnx.update("UPDATE INSERT DATA {<http://test.org/dudule> a <<ttp://test.org/bidule>");
        cnx.update("UPDATE DELETE WHERE {<http://test.org/dudue> ?p ?o");
         cnx.commit();

        cnx.close();
    }
}
```

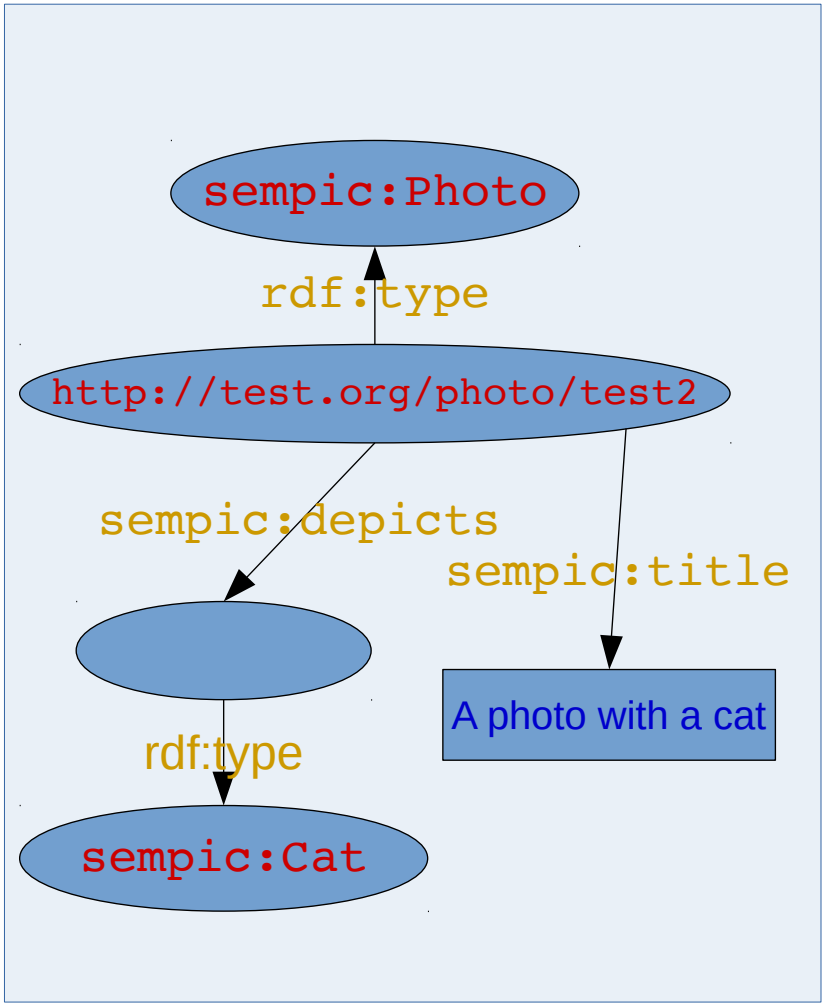
Core RDF API

package org.apache.jena.rdf.model





Core RDF API: create a model




```
// Create an empty RDF model
Model m = ModelFactory.createDefaultModel();
// create and add a named (i.e. with an URI)
// resource of type photo
// this adds statement:
// [<.../test2> rdf:type sempic:photo]
Resource photo = m.createResource(
                 "http://test.org/photo/test2",
                 Photo.type);
// this adds statement:
// [<.../test2> sempic:title "A photo with a cat"]
photo.addLiteral(Photo.title, "A photo with a cat");
// create and add an anonymous resource
// of type cat
// this adds statement:
// [_:generatedID rdf:type sempic:cat]
Resource cat = m.createResource(Types.cat);
// this adds statement:
// [<.../test2> sempic:depicts _:generatedID]
photo.addProperty(Photo.depicts, cat);
```