R and Bio Web Semantic Technologies

Dott. Francesco Bardozzo

I would to thank Prof. Sabrina Senatore for her important review.

The aim of this work is to show
some relevant solutions about
the relationship between the
R Framework and the Semantic Web.

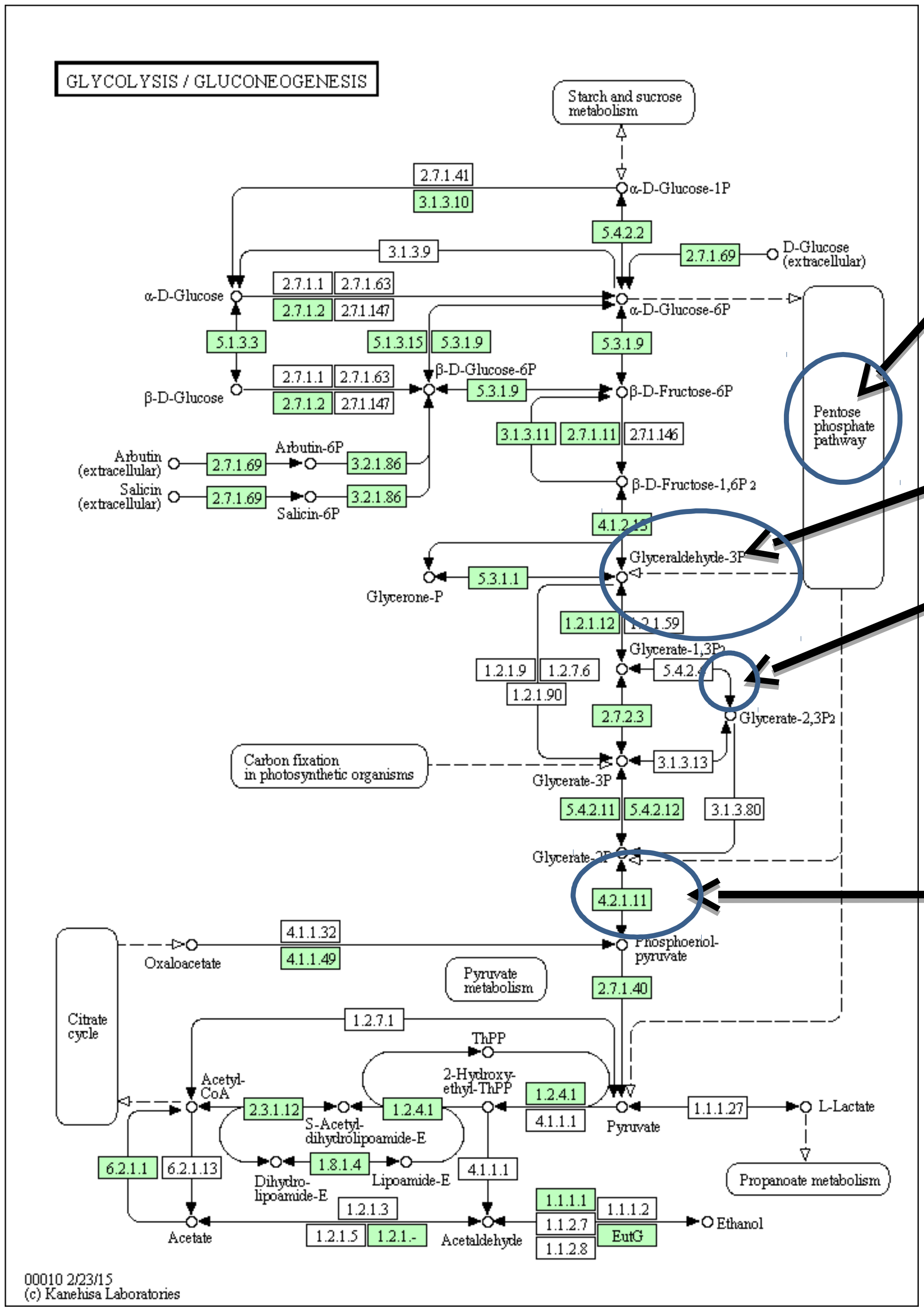
Some examples from the world of Bioinformatics and R:

- **M**odelling and Mapping a Knowledge Base: XSLT to RDF
- **I**mport and Update an RDF\OWL
- **C**hanging the semantic formats
- **I**nterrogation of an Ontology Web Service
- **I**nteraction with SPARQL
- **I**nteraction with a Triple Store
- **I**nteraction with a
  SPARQL endpoint and building
  of a new model with the
  “construct” SPARQL.

Knowledge Base Modelling. The example of KEGG’s Metabolic Networks

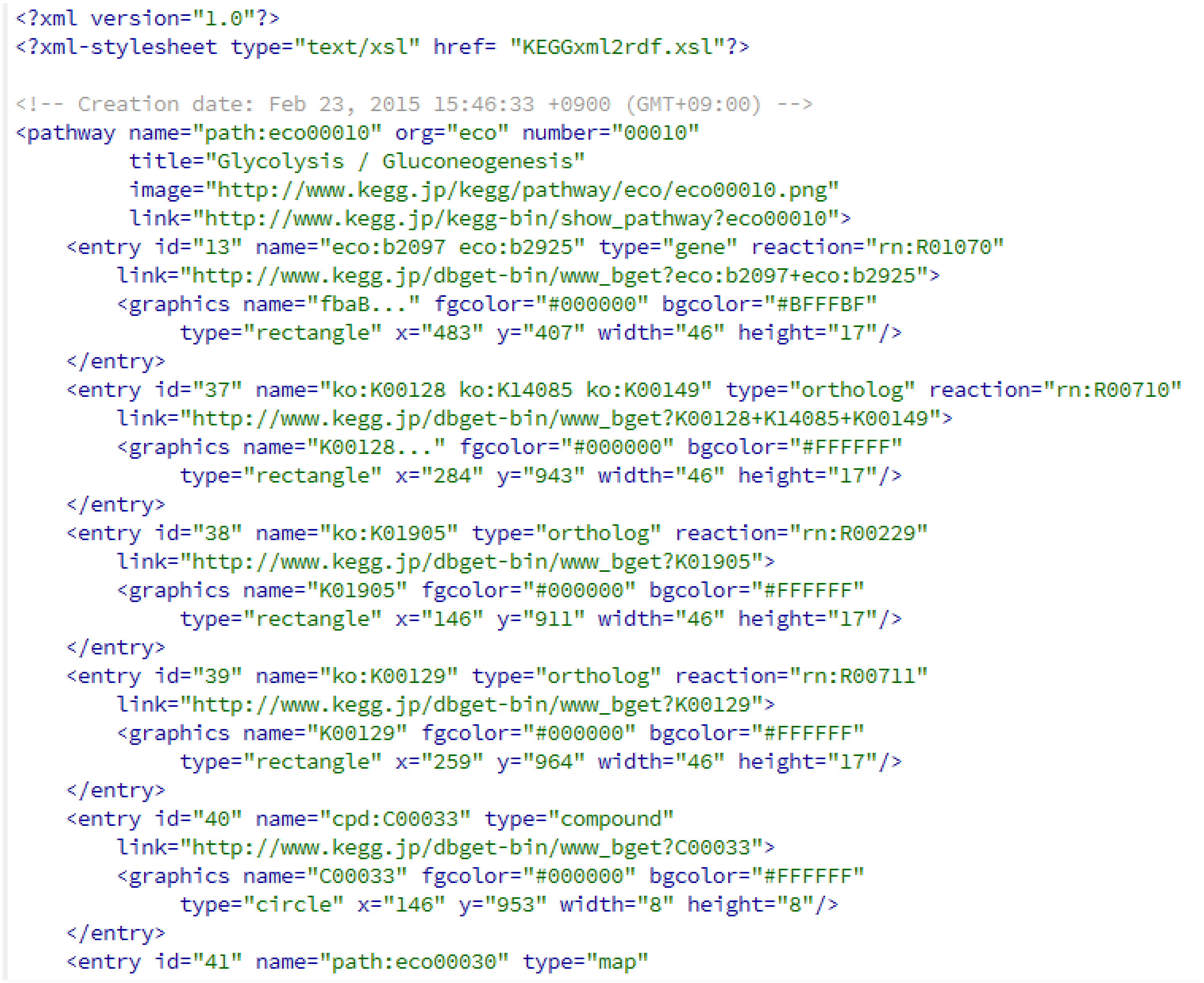
- A Metabolic Network is a directed complex graph where each edge represents a chemical reaction for the transformation of a compound in another and each node is a protein that catalyze this reaction.

- Do a mapping that makes sense: XML + XSLT = RDF



KEGG pathway

- Refers to other netwoks
- Compounds
- Reactions reversible or not reversible - edges
- Enzymes\Proteins - nodes



```xml
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href= "KEGGxml2rdf.xsl"?>

<!-- Creation date: Feb 23, 2015 15:46:33 +0900 (GMT+09:00) -->
<pathway name="path:eco00010" org="eco" number="00010"
         title="Glycolysis / Gluconeogenesis"
         image="http://www.kegg.jp/kegg/pathway/eco/eco00010.png"
         link="http://www.kegg.jp/kegg-bin/show_pathway?eco00010">
    <entry id="13" name="eco:b2097 eco:b2925" type="gene" reaction="rn:R01070"
        link="http://www.kegg.jp/dbget-bin/www_bget?eco:b2097+eco:b2925">
        <graphics name="fbaB..." fgcolor="#000000" bgcolor="#BFFFBF"
             type="rectangle" x="483" y="407" width="46" height="17"/>
    </entry>
    <entry id="37" name="ko:K00128 ko:K14085 ko:K00149" type="ortholog" reaction="rn:R00710"
        link="http://www.kegg.jp/dbget-bin/www_bget?K00128+K14085+K00149">
        <graphics name="K00128..." fgcolor="#000000" bgcolor="#FFFFFF"
             type="rectangle" x="284" y="943" width="46" height="17"/>
    </entry>
    <entry id="38" name="ko:K01905" type="ortholog" reaction="rn:R00229"
        link="http://www.kegg.jp/dbget-bin/www_bget?K01905">
        <graphics name="K01905" fgcolor="#000000" bgcolor="#FFFFFF"
             type="rectangle" x="146" y="911" width="46" height="17"/>
    </entry>
    <entry id="39" name="ko:K00129" type="ortholog" reaction="rn:R00711"
        link="http://www.kegg.jp/dbget-bin/www_bget?K00129">
        <graphics name="K00129" fgcolor="#000000" bgcolor="#FFFFFF"
             type="rectangle" x="259" y="964" width="46" height="17"/>
    </entry>
    <entry id="40" name="cpd:C00033" type="compound"
        link="http://www.kegg.jp/dbget-bin/www_bget?C00033">
        <graphics name="C00033" fgcolor="#000000" bgcolor="#FFFFFF"
             type="circle" x="146" y="953" width="8" height="8"/>
    </entry>
    <entry id="41" name="path:eco00030" type="map"
```

Abstract

KEGG data involves a network of distributed data that receives input from different communities. Expressing this data on semantic web will express the network of relations in a format supported by the underlying architecture. This document proposes a partial mapping from the KEGG DTD to an RDF format and provides XSLT to do that transfrtomation.

Status of this Document

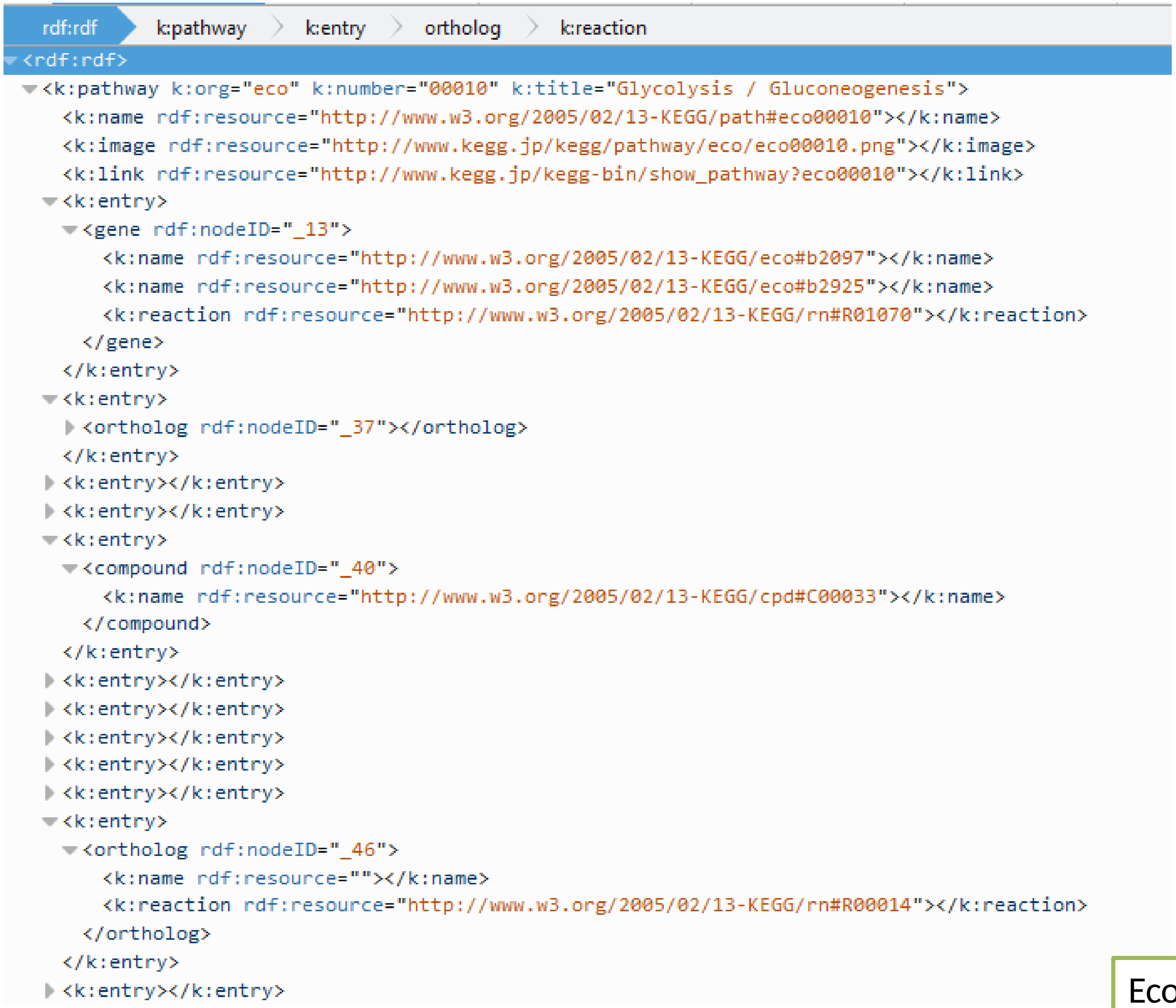
This documents experiments by the author. It is not endorsed by the W3C Team or Membership. It is hoped that the work described here will be pertinent to the content selection work pursued by W3C.

Introduction

RDF good. We like. @@@ perhaps someone should expand upon this theme a bit.@@@ This document serves as an HTML representation of the namespace `http://www.w3.org/2005/02/13-KEGG/#`.

Mapping

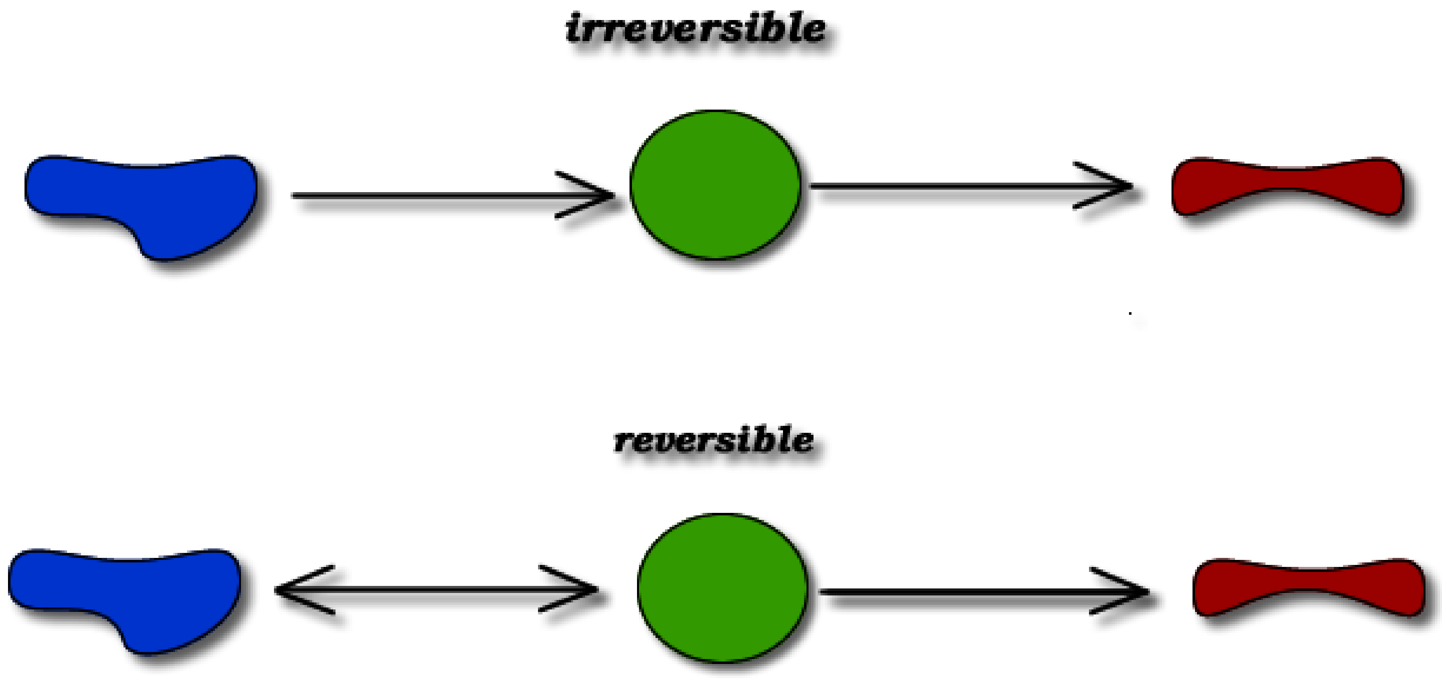
An XSLT stylesheet tranforms KGML documents into RDF/XML documents. Applying the XSLT to KGML document desribing Glycolysis / Gluconeogenesis yields an equally expressive RDF form.



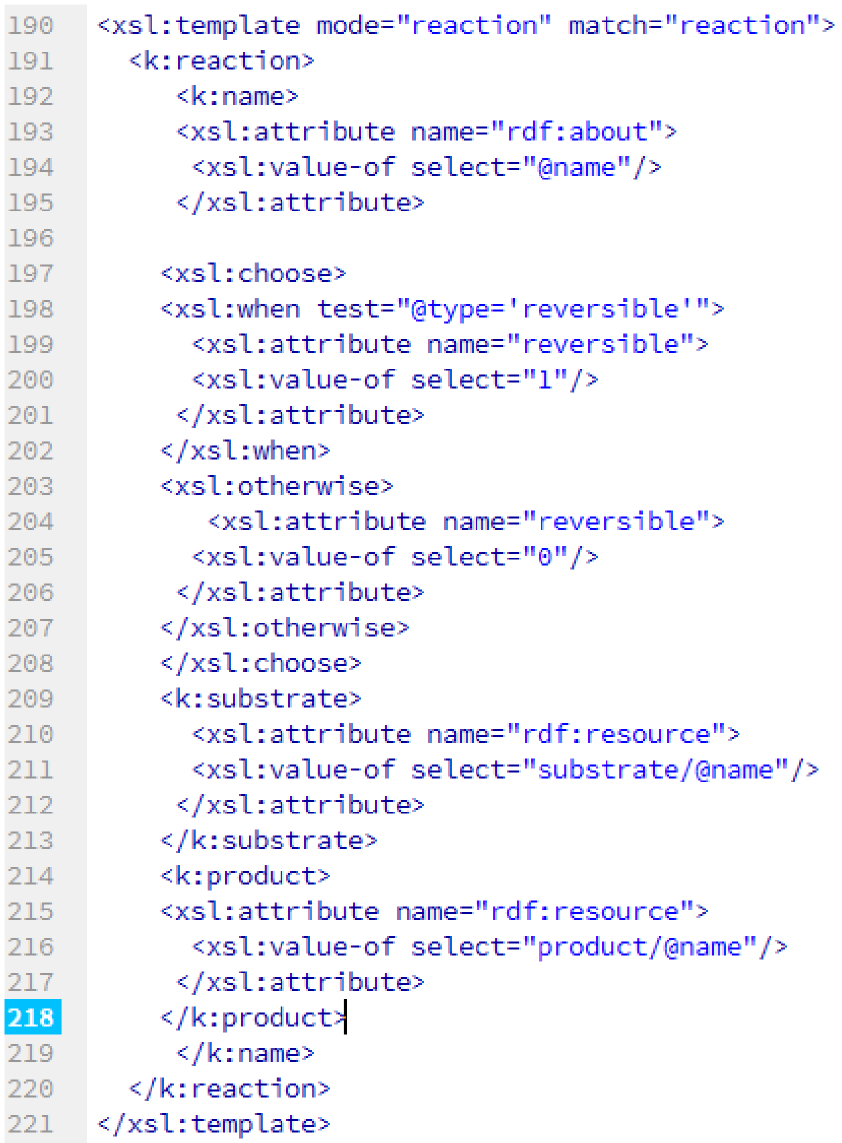
```
<rdf:rdf>
  <k:pathway k:org="eco" k:number="00010" k:title="Glycolysis / Gluconeogenesis">
    <k:name rdf:resource="http://www.w3.org/2005/02/13-KEGG/path#eco00010"></k:name>
    <k:image rdf:resource="http://www.kegg.jp/kegg/pathway/eco/eco00010.png"></k:image>
    <k:link rdf:resource="http://www.kegg.jp/kegg-bin/show_pathway?eco00010"></k:link>
    <k:entry>
      <gene rdf:nodeID="_13">
        <k:name rdf:resource="http://www.w3.org/2005/02/13-KEGG/eco#b2097"></k:name>
        <k:name rdf:resource="http://www.w3.org/2005/02/13-KEGG/eco#b2925"></k:name>
        <k:reaction rdf:resource="http://www.w3.org/2005/02/13-KEGG/rn#R01070"></k:reaction>
      </gene>
    </k:entry>
    <k:entry>
      <ortholog rdf:nodeID="_37"></ortholog>
    </k:entry>
    <k:entry></k:entry>
    <k:entry></k:entry>
    <k:entry>
      <compound rdf:nodeID="_40">
        <k:name rdf:resource="http://www.w3.org/2005/02/13-KEGG/cpd#C00033"></k:name>
      </compound>
    </k:entry>
    <k:entry></k:entry>
    <k:entry></k:entry>
    <k:entry></k:entry>
    <k:entry></k:entry>
    <k:entry></k:entry>
    <k:entry>
      <ortholog rdf:nodeID="_46">
        <k:name rdf:resource=""></k:name>
        <k:reaction rdf:resource="http://www.w3.org/2005/02/13-KEGG/rn#R00014"></k:reaction>
      </ortholog>
    </k:entry>
    <k:entry></k:entry>
```

Eco00010.rdf

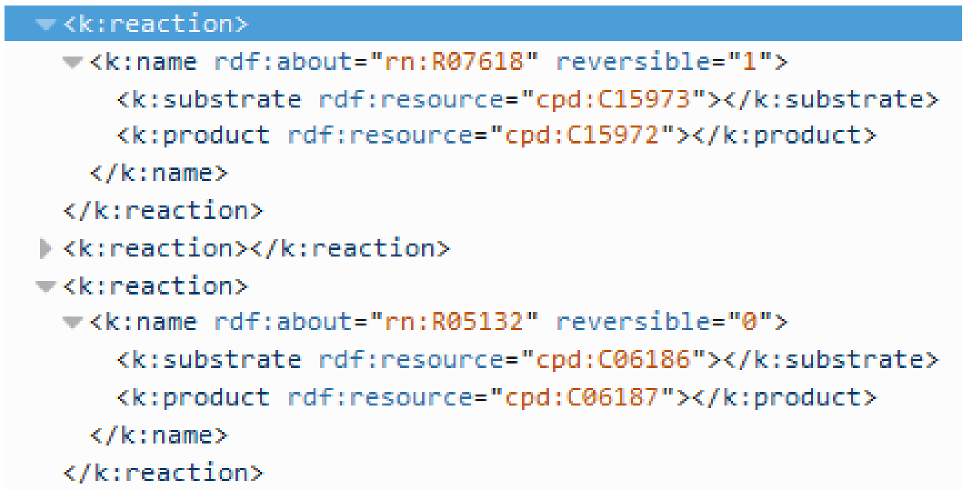
A semantic problem is correlated to the meaning of a Metabolic Reaction



A possible semantic implementation of a metabolic reaction:



```
<xsl:template mode="reaction" match="reaction">
  <k:reaction>
     <k:name>
     <xsl:attribute name="rdf:about">
      <xsl:value-of select="@name"/>
     </xsl:attribute>

    <xsl:choose>
    <xsl:when test="@type='reversible'">
      <xsl:attribute name="reversible">
      <xsl:value-of select="1"/>
     </xsl:attribute>
    </xsl:when>
    <xsl:otherwise>
       <xsl:attribute name="reversible">
      <xsl:value-of select="0"/>
     </xsl:attribute>
    </xsl:otherwise>
    </xsl:choose>
    <k:substrate>
      <xsl:attribute name="rdf:resource">
      <xsl:value-of select="substrate/@name"/>
     </xsl:attribute>
    </k:substrate>
    <k:product>
    <xsl:attribute name="rdf:resource">
      <xsl:value-of select="product/@name"/>
     </xsl:attribute>
    </k:product>
     </k:name>
  </k:reaction>
</xsl:template>
```

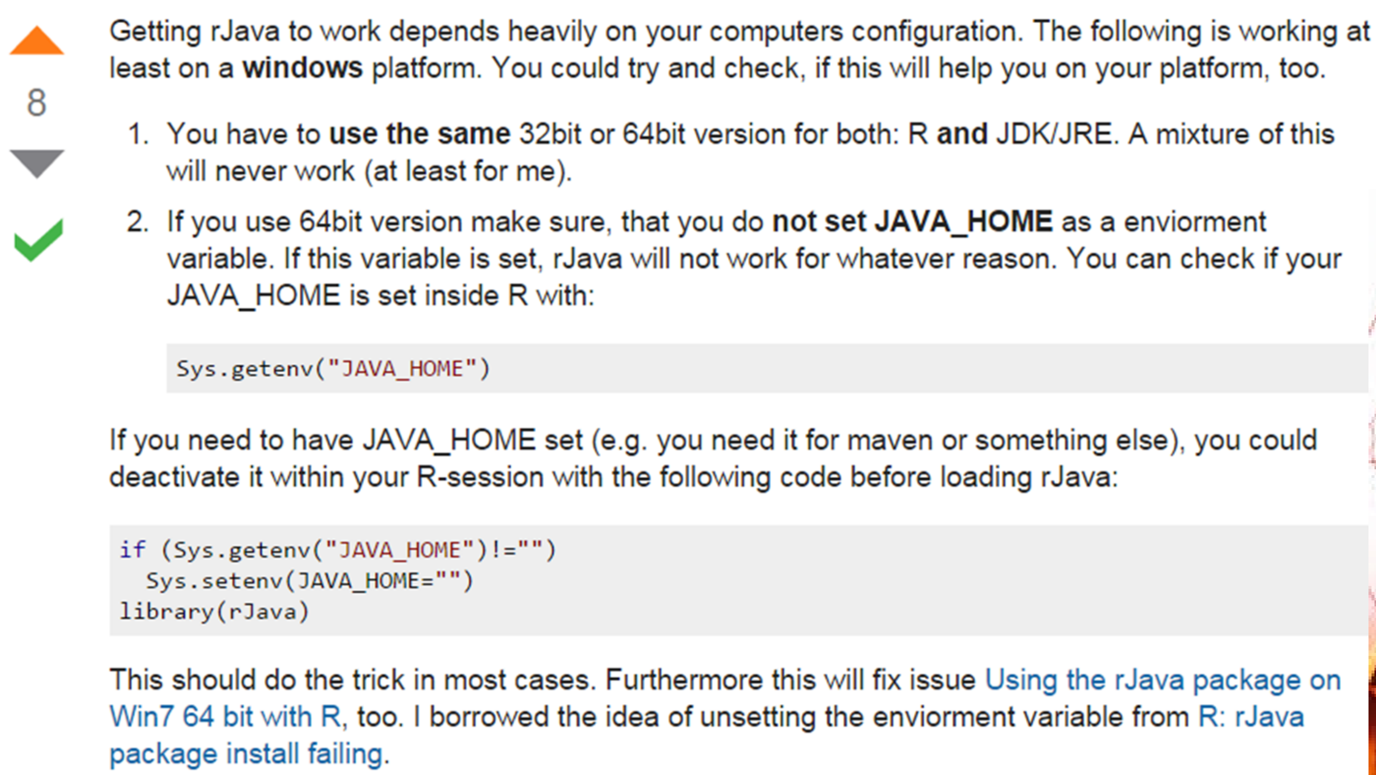


```
<k:reaction>
  <k:name rdf:about="rn:R07618" reversible="1">
    <k:substrate rdf:resource="cpd:C15973"></k:substrate>
    <k:product rdf:resource="cpd:C15972"></k:product>
  </k:name>
</k:reaction>
<k:reaction></k:reaction>
<k:reaction>
  <k:name rdf:about="rn:R05132" reversible="0">
    <k:substrate rdf:resource="cpd:C06186"></k:substrate>
    <k:product rdf:resource="cpd:C06187"></k:product>
  </k:name>
</k:reaction>
```

But it could be further improved…

We can use the RDF generated from the modified XSLT and put this one in R…

so… we need some R packages:
rJava, rrdf, rrdflibs… rCurl…


8

Getting rJava to work depends heavily on your computers configuration. The following is working at least on a **windows** platform. You could try and check, if this will help you on your platform, too.

1. You have to **use the same** 32bit or 64bit version for both: R **and** JDK/JRE. A mixture of this will never work (at least for me).
2. If you use 64bit version make sure, that you do **not set JAVA_HOME** as a enviorment variable. If this variable is set, rJava will not work for whatever reason. You can check if your JAVA_HOME is set inside R with:

```
Sys.getenv("JAVA_HOME")
```

If you need to have JAVA_HOME set (e.g. you need it for maven or something else), you could deactivate it within your R-session with the following code before loading rJava:

```
if (Sys.getenv("JAVA_HOME")!="")
  Sys.setenv(JAVA_HOME="")
library(rJava)
```

This should do the trick in most cases. Furthermore this will fix issue Using the rJava package on Win7 64 bit with R, too. I borrowed the idea of unsetting the enviorment variable from R: rJava package install failing.

Package ‘rrdflibs’ was removed from the CRAN repository.

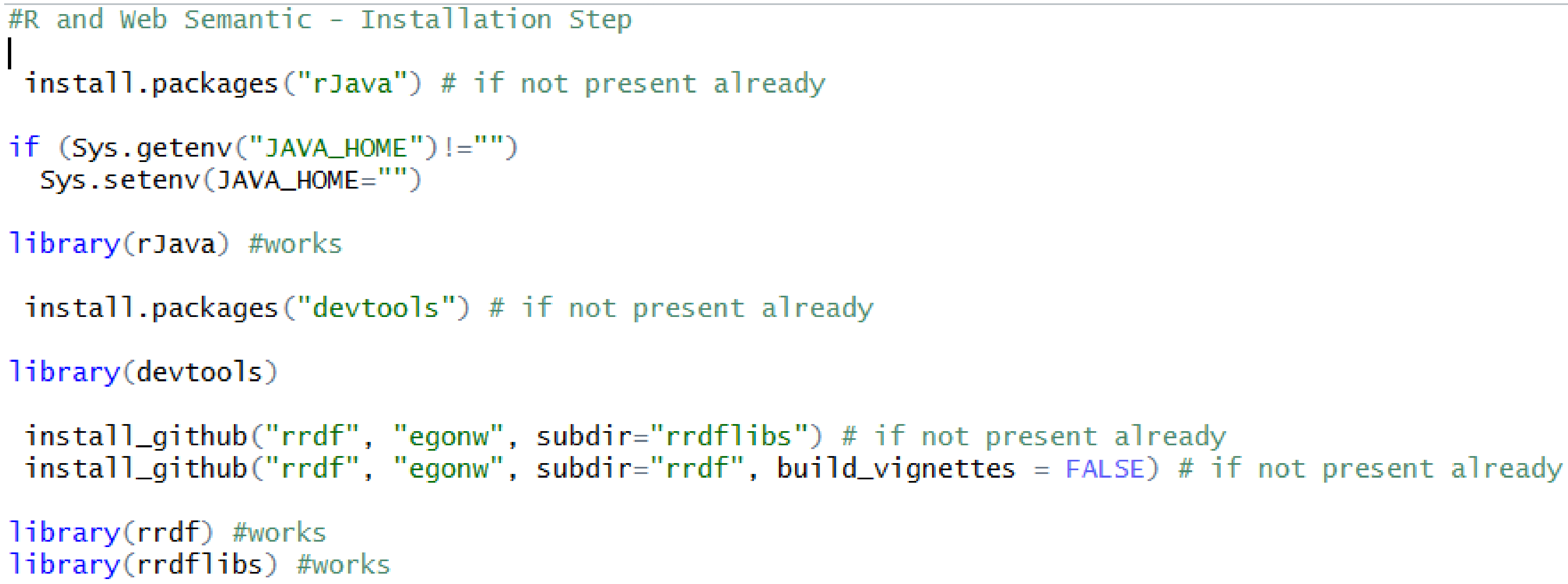
Not True...

Formerly available versions can be obtained from the archive.

Archived on 2014-08-29 as does not comply with policy on Java sources.

Mission Possible

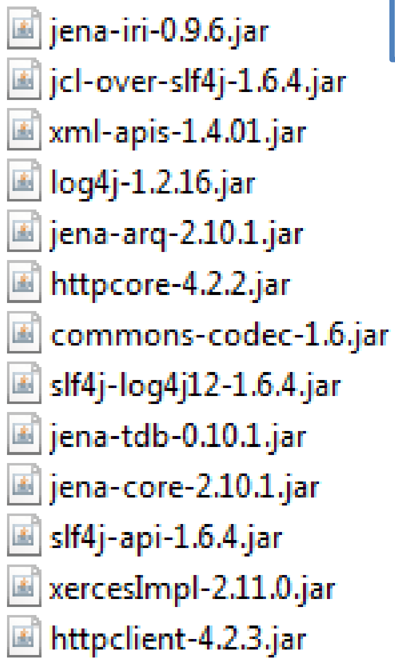
Installation Packages: rJava, rrdf, rrdflibs



```
#R and Web Semantic - Installation Step

 install.packages("rJava") # if not present already

if (Sys.getenv("JAVA_HOME")!="")
  Sys.setenv(JAVA_HOME="")

library(rJava) #works

 install.packages("devtools") # if not present already

library(devtools)

 install_github("rrdf", "egonw", subdir="rrdflibs") # if not present already
 install_github("rrdf", "egonw", subdir="rrdf", build_vignettes = FALSE) # if not present already

library(rrdf) #works
library(rrdflibs) #works
```

Rrdf is a bridge developed for using Rrdflibs that is a bridge for using Apache Jena libraries. These libraries are developed in Java, we need another bridge (rJava) for the system communication to JVM.

The Apache Jena libraries bridged from rrdflibs



jena-iri-0.9.6.jar
jcl-over-slf4j-1.6.4.jar
xml-apis-1.4.01.jar
log4j-1.2.16.jar
jena-arq-2.10.1.jar
httpcore-4.2.2.jar
commons-codec-1.6.jar
slf4j-log4j12-1.6.4.jar
jena-tdb-0.10.1.jar
jena-core-2.10.1.jar
slf4j-api-1.6.4.jar
xercesImpl-2.11.0.jar
httpclient-4.2.3.jar

Jena Core: Jena is a Java framework for building Semantic Web applications. It provides a programmatic environment for RDF, RDFS and OWL, SPARQL and includes a rule-based inference engine.

The IRI module provides an implementation of the IRI and URI specifications (RFC 3987 and 3986) which are used across Jena in order to comply with relevant W3C specifications for RDF and SPARQL which require conformance to these specifications.

ARQ is a SPARQL 1.1 query engine for Apache Jena

TDB is a storage subsystem for Jena and ARQ, it is a native triple store providing persisent disk based storage of triples/quads.

And others like DOM – XML tree (Xerces) [ex: postorder sealing] – remember that the rdf\owl Model is a Graph in Apache Jena.

Create a new RDF triple store object:

new.rdf(ontology=TRUE)

Arguments

Ontology : boolean - Indicates if the model should be an ontological model (the default).
An ontology model with the *default* settings, which are set for maximum compatibility with the previous version of Jena.

These defaults are:

- OWL-Full language
- in-memory storage
- RDFS inference, which principally produces entailments from the sub-class and sub-property hierarchies.

Important note: this means that the *default ontology model* **does** include some inference, with consequences both for the performance of the model, and for the triples which appear in the model.

Value

A Java Model object containing the triples loaded from the file.

Examples

```
store = new.rdf()
store = new.rdf(ontology=FALSE)
```

Loads and appends triples from a RDF serialization:

load.rdf(filename.rdf, format = "RDF/XML", appendTo=NULL)

Arguments

filename - Name of the file to read the triples from.
format - Format of the RDF file. Accepted formats: **RDF/XML, TURTLE, N-TRIPLES, and N3.**
appendTo - Optional Java Model object to which read statements are added.

Value
A Java Model object containing the triples loaded from the file.

Examples
model = load.rdf("someFile.xml", "RDF/XML")
model = new.rdf(ontology=FALSE)
load.rdf("someFile.xml", "RDF/XML", model)

Saves triples to a RDF serialization:

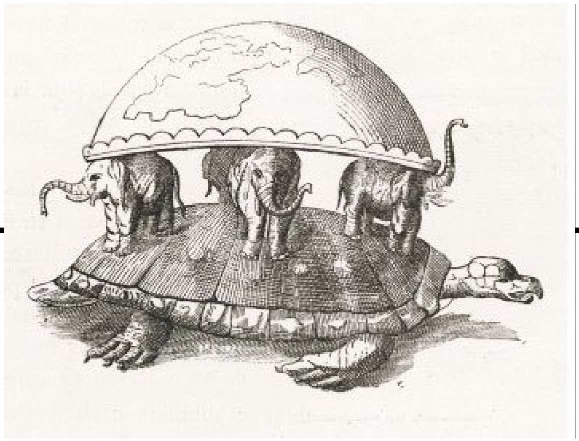
save.rdf(store, filename, format = "RDF/XML")

Arguments

Store - A triple store, for example create with new.rdf().
Filename - Name of the file to read the triples from.
Format - Format of the RDF file, e.g. RDF/XML. Other formats are RDF/XML-ABBREV, N3, TURTLE, and N-TRIPLE.

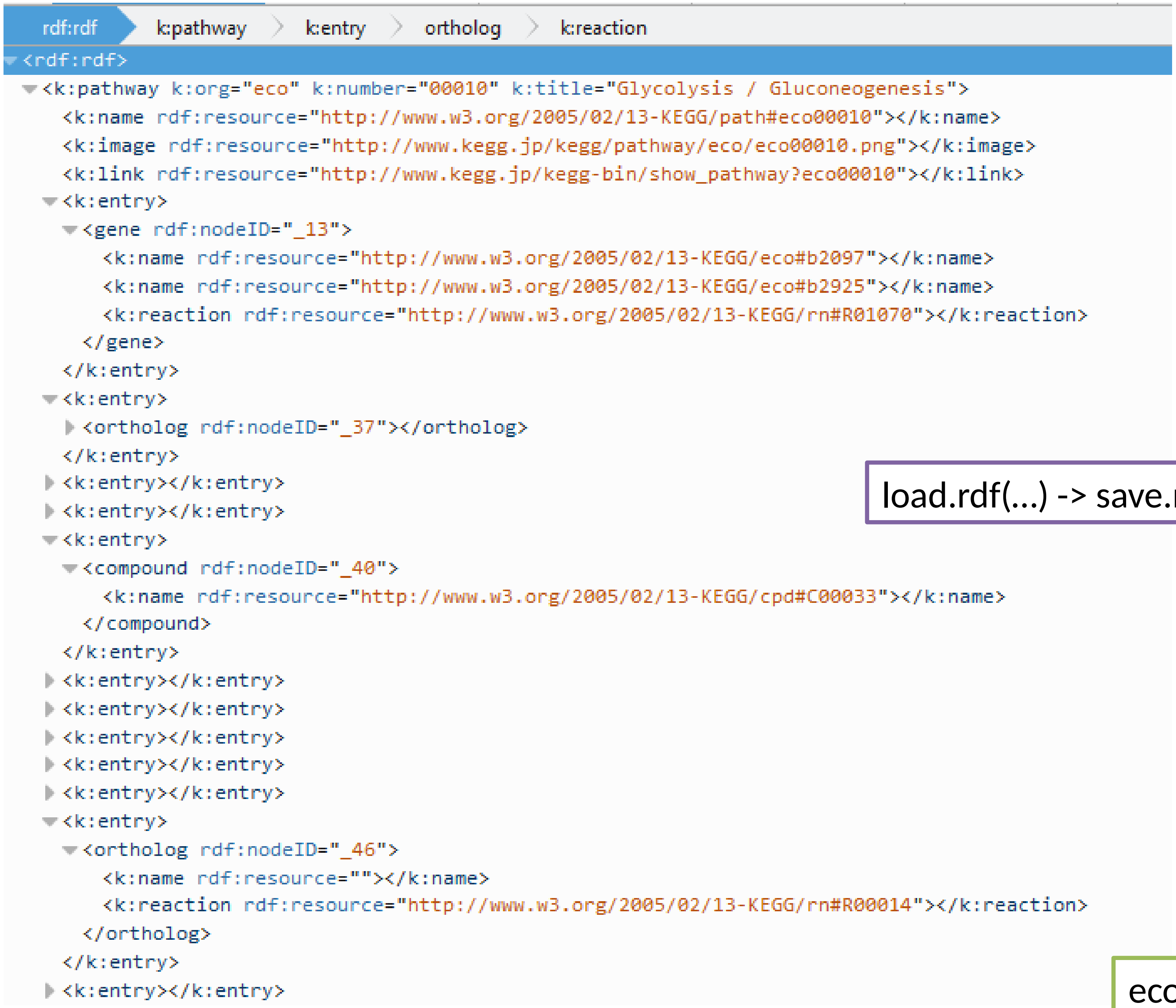
Value
A Java Model object containing the triples loaded from the file.

Examples
save.rdf(store, "someFile.xml", "N3")



Ah yes, we will never get to the bottom of some things.
Infinite regressions. What existed before the universe existed?
If God created the universe, what created God?
It's turtles all the way down. John Locke

Example with the RDF of the Escherichia Coli’ Glycolisis Metabolic Networks – pt1



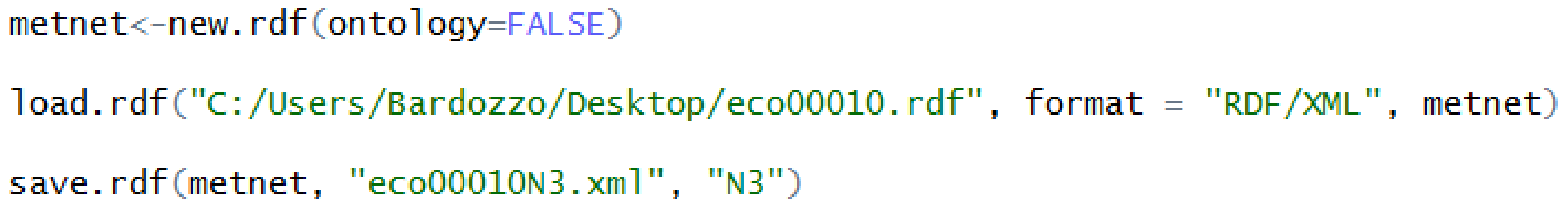
```
<rdf:rdf>
  <k:pathway k:org="eco" k:number="00010" k:title="Glycolysis / Gluconeogenesis">
    <k:name rdf:resource="http://www.w3.org/2005/02/13-KEGG/path#eco00010"></k:name>
    <k:image rdf:resource="http://www.kegg.jp/kegg/pathway/eco/eco00010.png"></k:image>
    <k:link rdf:resource="http://www.kegg.jp/kegg-bin/show_pathway?eco00010"></k:link>
    <k:entry>
      <gene rdf:nodeID="_13">
        <k:name rdf:resource="http://www.w3.org/2005/02/13-KEGG/eco#b2097"></k:name>
        <k:name rdf:resource="http://www.w3.org/2005/02/13-KEGG/eco#b2925"></k:name>
        <k:reaction rdf:resource="http://www.w3.org/2005/02/13-KEGG/rn#R01070"></k:reaction>
      </gene>
    </k:entry>
    <k:entry>
      <ortholog rdf:nodeID="_37"></ortholog>
    </k:entry>
    <k:entry></k:entry>
    <k:entry></k:entry>
    <k:entry>
      <compound rdf:nodeID="_40">
        <k:name rdf:resource="http://www.w3.org/2005/02/13-KEGG/cpd#C00033"></k:name>
      </compound>
    </k:entry>
    <k:entry></k:entry>
    <k:entry></k:entry>
    <k:entry></k:entry>
    <k:entry></k:entry>
    <k:entry></k:entry>
    <k:entry>
      <ortholog rdf:nodeID="_46">
        <k:name rdf:resource=""></k:name>
        <k:reaction rdf:resource="http://www.w3.org/2005/02/13-KEGG/rn#R00014"></k:reaction>
      </ortholog>
    </k:entry>
    <k:entry></k:entry>
```

load.rdf(…) -> save.rdf(…) ?

eco00010.rdf

Example with the RDF of the Escherichia Coli' Glycolisis Metabolic Networks – pt2

Loading a file RDF\XML and save this one in N3



```
metnet<-new.rdf(ontology=FALSE)

load.rdf("C:/Users/Bardozzo/Desktop/eco00010.rdf", format = "RDF/XML", metnet)

save.rdf(metnet, "eco00010N3.xml", "N3")
```

A piece of eco00010N3.xml….

```
k:entry    [ a        k:Gene ;
             k:name     <http://www.w3.org/2005/02/13-KEGG/eco#b1002> ;
             k:reaction  <http://www.w3.org/2005/02/13-KEGG/rn#R00947>
           ] ;
```

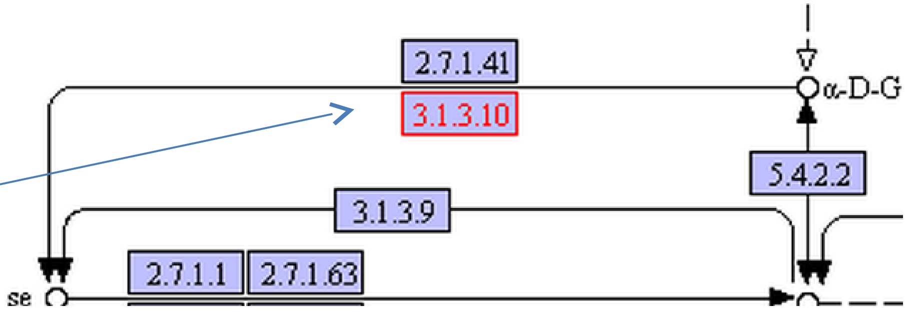
Example with the RDF of the Escherichia Coli' Glycolisis Metabolic Networks – pt3

```
k:entry    [ a         k:Gene ;
             k:name      <http://www.w3.org/2005/02/13-KEGG/eco#b1002> ;
             k:reaction  <http://www.w3.org/2005/02/13-KEGG/rn#R00947>
           ] ;
```

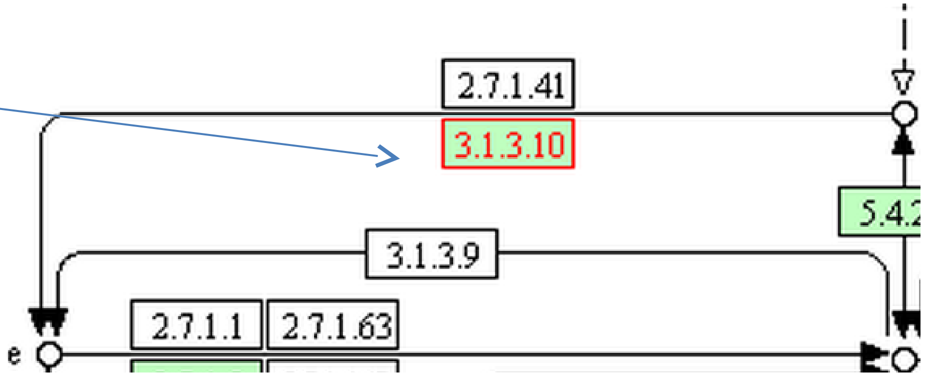
Testing if the triple is correct:

The reaction map and the protein map shows the same node.

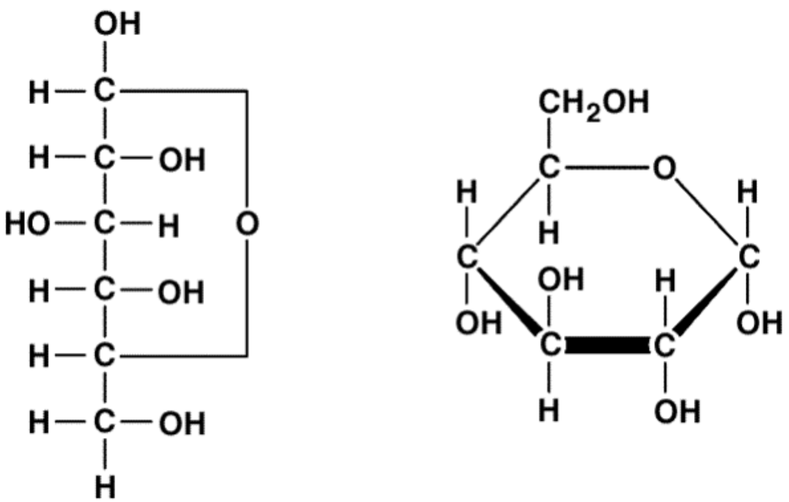
http://www.genome.jp/kegg-bin/show_pathway?**rn00010+R00947**



http://www.genome.jp/kegg-bin/show_pathway?**eco00010+b1002**



Another toy example**: you can take an ontology from the web service ambit in R using RCurl.** The metabolic network showed before is designed to transform glucose, but what is “glucose”?



Glucose

The formula becomes a string!
This type of encoding is called SMILES

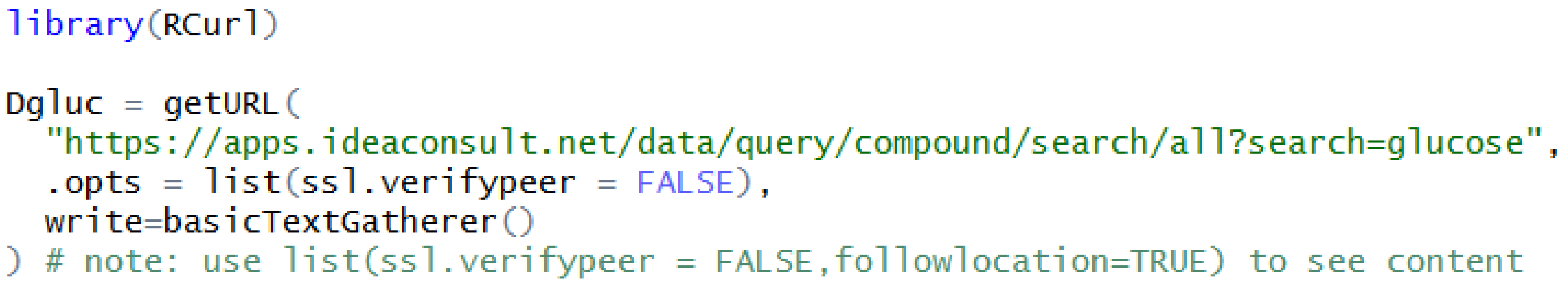
```
O[C@H]1[C@H](O)[C@@H](CO)OC(O)
[C@@H]1O|OC1C(O)C(CO)OC(O)C1O|
C([C@@H]1[C@H]([C@@H]([C@@H]
(C(O1)O)O)O)O)O|C([C@@H]1[C@H]([C@@H]
([C@H]([C@H](O1)O)O)O)O)O|O[C@H]1[C@H]
(O)[C@@H](CO)O[C@H](O)[C@@H]1O|
C([C@@H]1[C@H]([C@H]([C@H]([C@@H]
(O1)O)O)O)O)O
```

Another toy example: you can take an ontology from a web service ambit in R.

We used another package: Rcurl.

https - because of the SSL – May have some problems.

I suggest to use **basicTextGatherer():** This is called when the libcurl engine finds sufficient data on the stream from which it is reading the response. It cumulates these bytes and hands them to a C routine in this package which calls the actual gathering function (or a suitable replacement) returned as the update component from this function of **multiTextGatherer()** is used when we are downloading multiple URIs concurrently in a single libcurl operation.



```r
library(RCurl)

Dgluc = getURL(
  "https://apps.ideaconsult.net/data/query/compound/search/all?search=glucose",
  .opts = list(ssl.verifypeer = FALSE),
  write=basicTextGatherer()
) # note: use list(ssl.verifypeer = FALSE,followlocation=TRUE) to see content
```

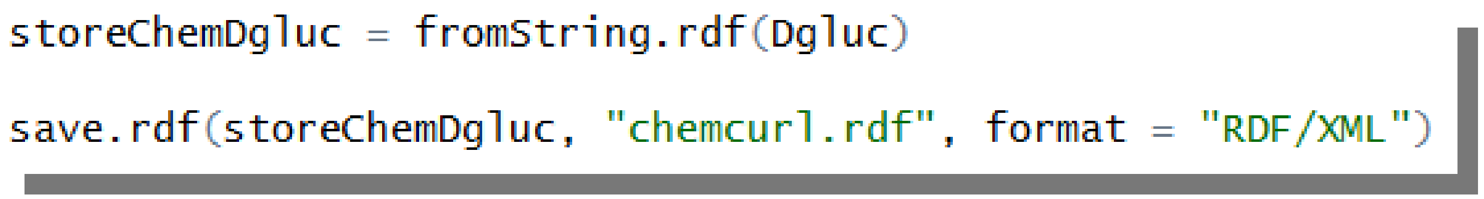
Loads triples from a string with RDF serialization.

fromString.rdf(rdfContent, format = "RDF/XML", appendTo=NULL)

Arguments

rdfContent - RDF serialization content.
Format - Format of the RDF content, e.g. RDF/XML.
appendTo - Optional Java Model object to which read statements are added.

Value
A Java Model object containing the triples loaded from the file.

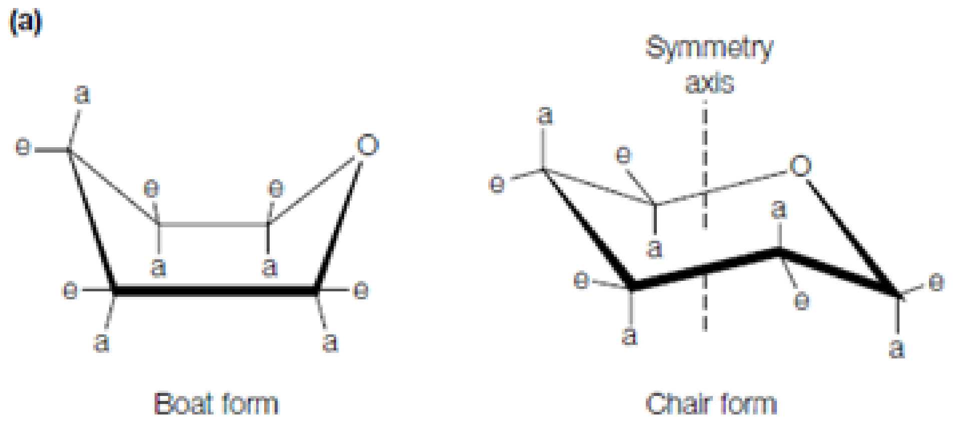


```
storeChemDgluc = fromString.rdf(Dgluc)

save.rdf(storeChemDgluc, "chemcurl.rdf", format = "RDF/XML")
```

```
          <ot:FeatureValue>
            <ot:value rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >O[C@H]1[C@H](O)[C@@H](CO)OC(O)[C@@H]1O
OC1C(O)C(CO)OC(O)C1O
C([C@@H]1[C@H]([C@@H]([C@@H](C(O1)O)O)O)O)O
C([C@@H]1[C@H]([C@@H]([C@H]([C@H](O1)O)O)O)O)O
O[C@H]1[C@H](O)[C@@H](CO)O[C@H](O)[C@@H]1O
C([C@@H]1[C@H]([C@H]([C@H]([C@@H](O1)O)O)O)O)O</ot:value>
            <ot:feature>
              <ot:Feature>
                <owl:sameAs rdf:resource="http://www.opentox.org/api/1.1#SMILES"/>
                <dc:title>http://www.opentox.org/api/1.1#SMILES</dc:title>
              </ot:Feature>
            </ot:feature>
          </ot:FeatureValue>
        </ot:values>
        <ot:values>
          <ot:FeatureValue>
            <ot:value rdf:datatype="http://www.w3.org/2001/XMLSchema#string"
            >dextrose
             Glucose
             GLUCOSE
             dextrosum/glucosum
             D-Glucose
             galactose
             d-mannose
             glucose
             alpha-d-glucopyranose
             beta-d-allose
          </ot:value>
            <ot:feature>
              <ot:Feature>
                <owl:sameAs rdf:resource="http://www.opentox.org/api/1.1#ChemicalName"/>
                <dc:title>http://www.opentox.org/api/1.1#ChemicalName</dc:title>
```

chemcurl.rdf

But the service gives us two answers… in particular two conformation of the same compound (glucose).



```
<ot:compound>
  <ot:Compound rdf:about="https://apps.ideaconsult.net/data/compound/39246/conformer/62230"/>
</ot:compound>

<ot:compound>
  <ot:Compound rdf:about="https://apps.ideaconsult.net/data/compound/2671/conformer/14456"/>
</ot:compound>
```

At this point we can use SPARQL to obtain the compounds: *chair* and *boat* conformation of glucose.

```
library(rrdf) #works
library(rrdflibs) #works
library(RCurl)

Dgluc = getURL(
  "https://apps.ideaconsult.net/data/query/compound/search/all?search=glucose",
  .opts = list(ssl.verifypeer = FALSE),
  write=basicTextGatherer()
) # note: use list(ssl.verifypeer = FALSE,followlocation=TRUE) to see content

storeChemDgluc = fromString.rdf(Dgluc)

compounds = sparql.rdf(storeChemDgluc, paste(
  "PREFIX ot: <http://www.opentox.org/api/1.1#>",
  "SELECT DISTINCT ?compound WHERE {",
  " ?compound a ot:Compound",
  "}"
)
)

# compound
# [1,] "https://apps.ideaconsult.net/data/compound/39246/conformer/62230"
# [2,] "https://apps.ideaconsult.net/data/compound/2671/conformer/14456"
```

Chair

Boat

- Interactions with a triple store.

  . add.triple

  . add.data.triple

- Build a new model from a SPARQL query

  construct.rdf

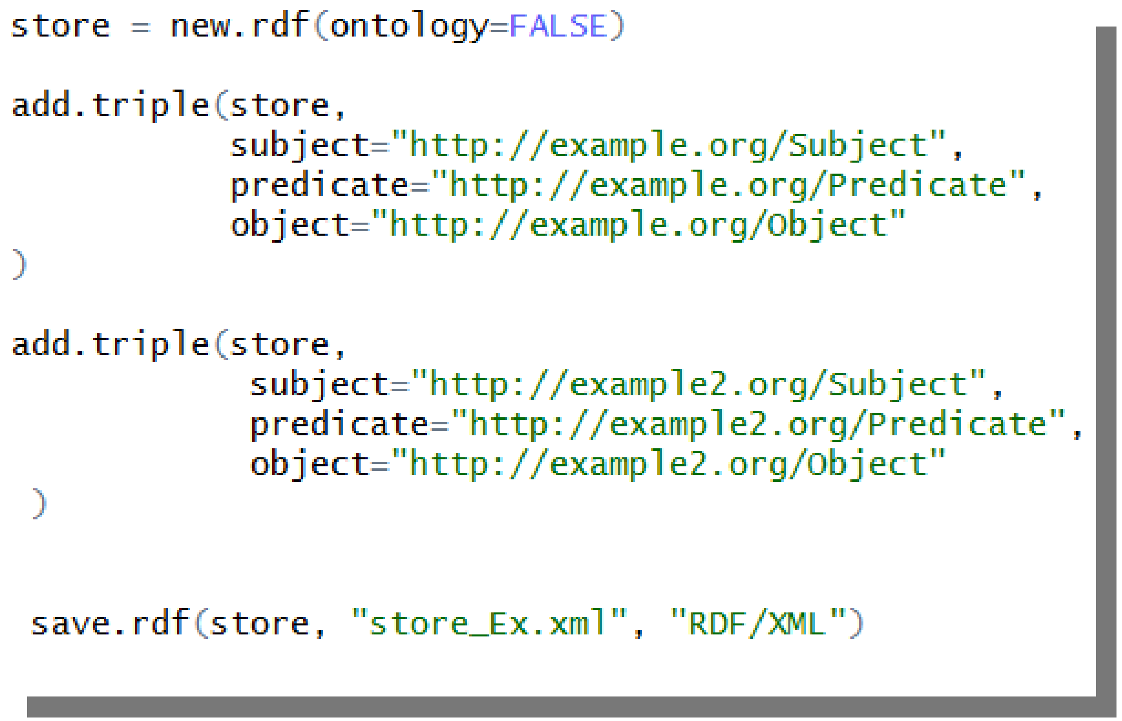
  construct.remote (endpoint SPARQL)

Interactions with a Triple Store - pt 1

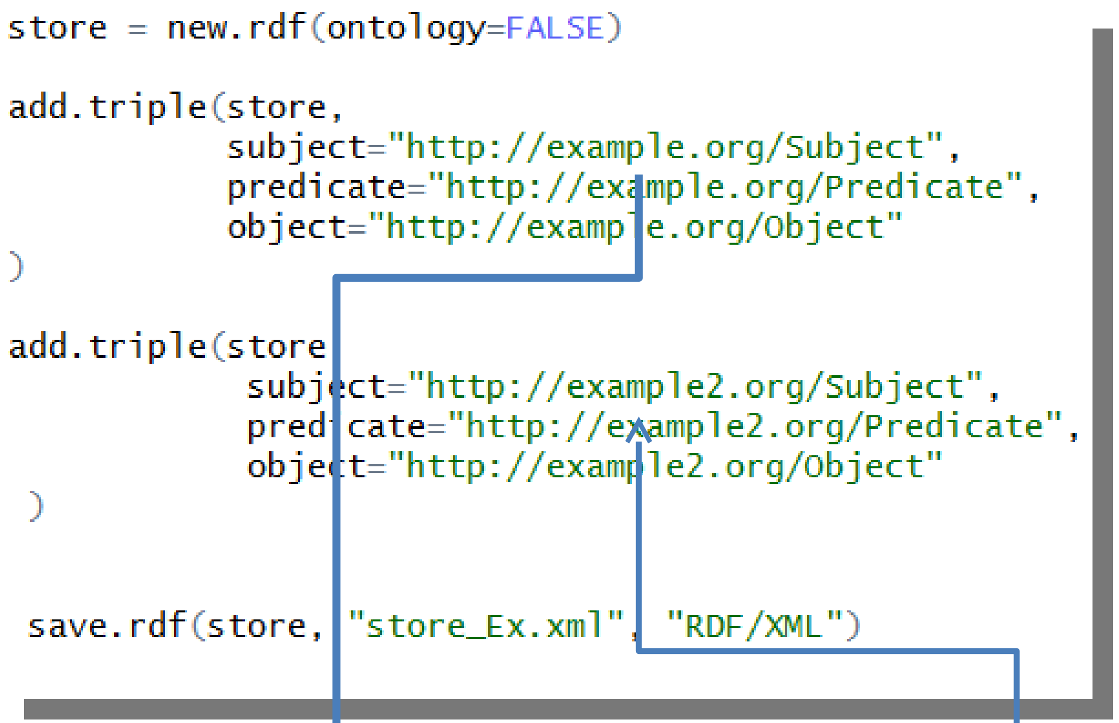
add.triple adds an object property to the model.

Arguments:

store	A triple store, for example create with new.rdf().
subject	URI of the subject.
predicate	URI of the predicate.
object	URI of the object.

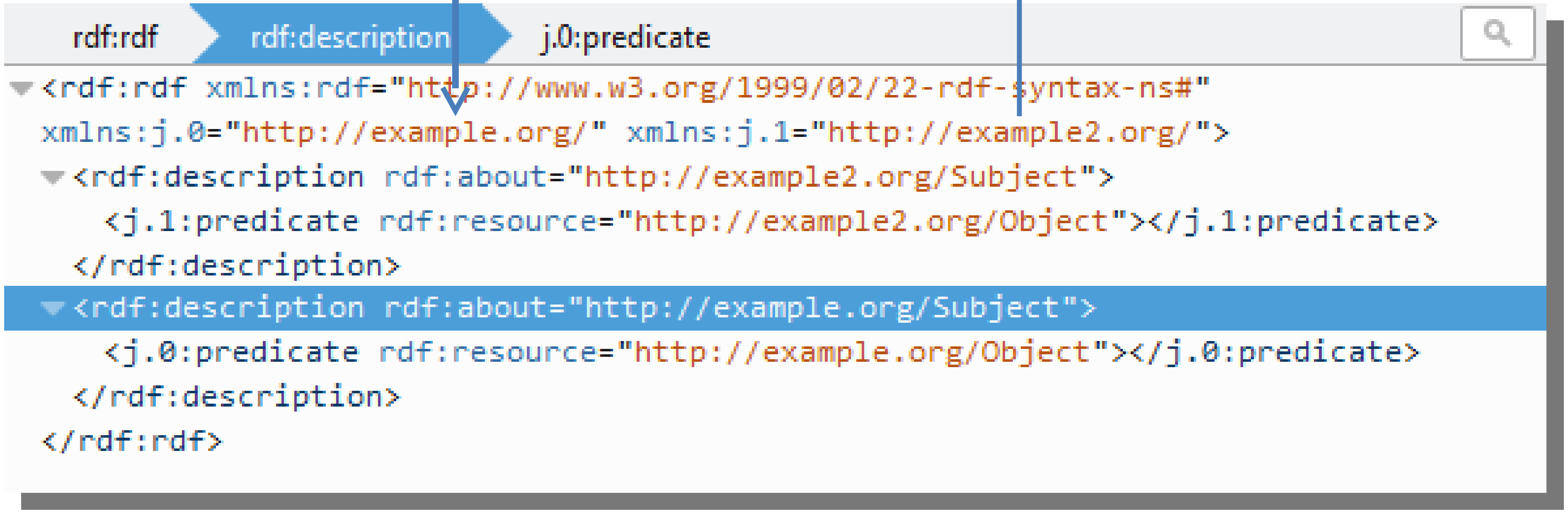


```
store = new.rdf(ontology=FALSE)

add.triple(store,
           subject="http://example.org/Subject",
           predicate="http://example.org/Predicate",
           object="http://example.org/Object"
)

add.triple(store,
            subject="http://example2.org/Subject",
            predicate="http://example2.org/Predicate",
            object="http://example2.org/Object"
 )


 save.rdf(store, "store_Ex.xml", "RDF/XML")
```



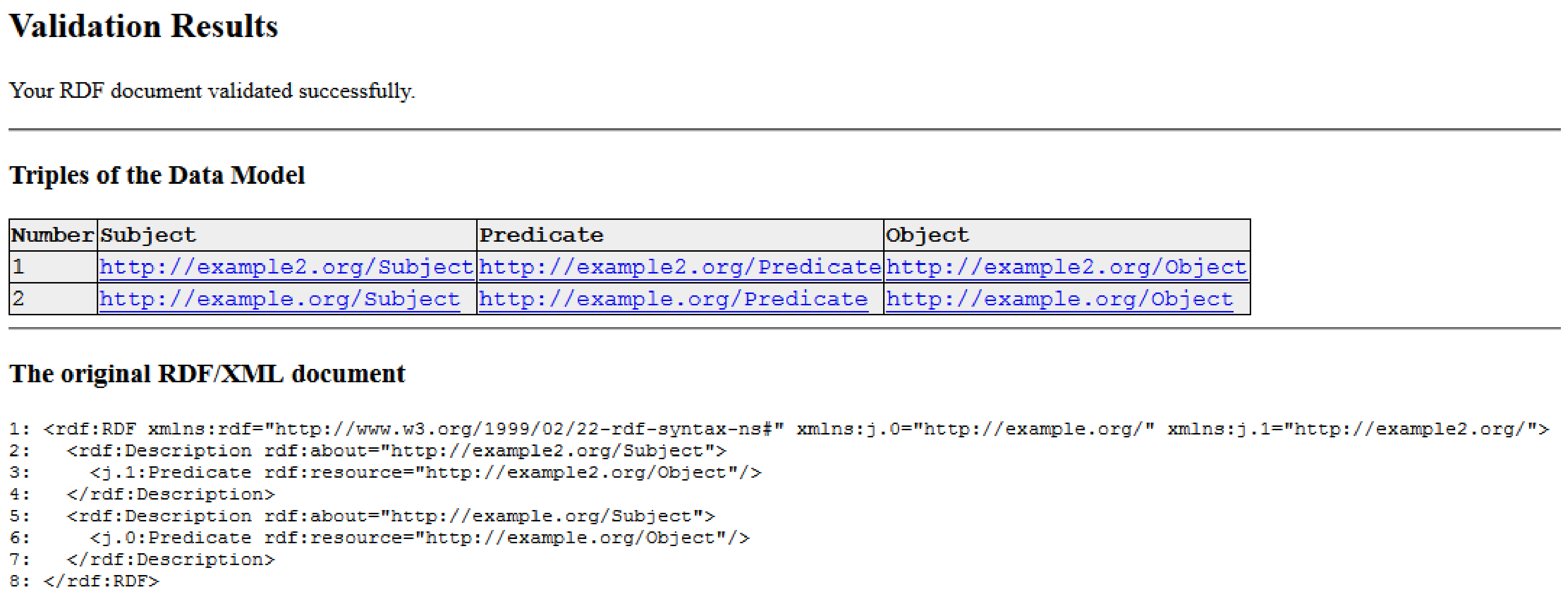
```
store = new.rdf(ontology=FALSE)

add.triple(store,
           subject="http://example.org/Subject",
           predicate="http://example.org/Predicate",
           object="http://example.org/Object"
)

add.triple(store,
            subject="http://example2.org/Subject",
            predicate="http://example2.org/Predicate",
            object="http://example2.org/Object"
 )


 save.rdf(store, "store_Ex.xml", "RDF/XML")
```

Namespaces were added automatically.



```
<rdf:rdf xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:j.0="http://example.org/" xmlns:j.1="http://example2.org/">
  <rdf:description rdf:about="http://example2.org/Subject">
    <j.1:predicate rdf:resource="http://example2.org/Object"></j.1:predicate>
  </rdf:description>
  <rdf:description rdf:about="http://example.org/Subject">
    <j.0:predicate rdf:resource="http://example.org/Object"></j.0:predicate>
  </rdf:description>
</rdf:rdf>
```

http://www.w3.org/RDF/Validator/rdfval



Validation Results

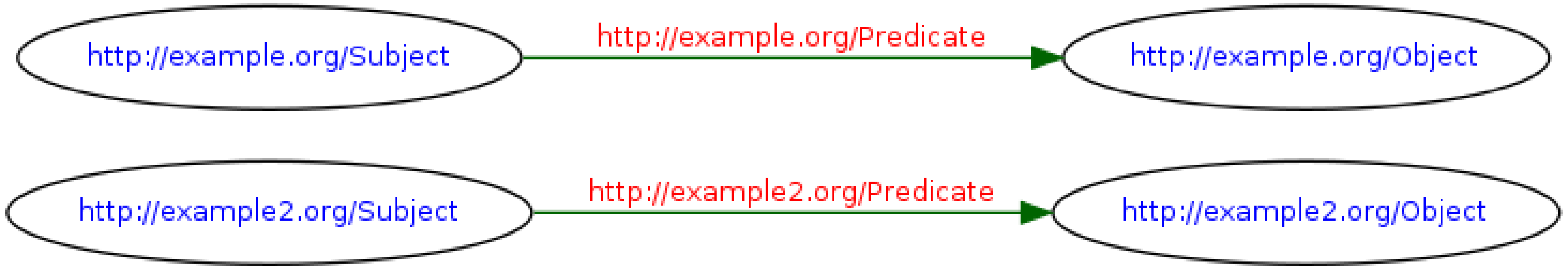
Your RDF document validated successfully.

Triples of the Data Model

Number	Subject	Predicate	Object
1	http://example2.org/Subject	http://example2.org/Predicate	http://example2.org/Object
2	http://example.org/Subject	http://example.org/Predicate	http://example.org/Object

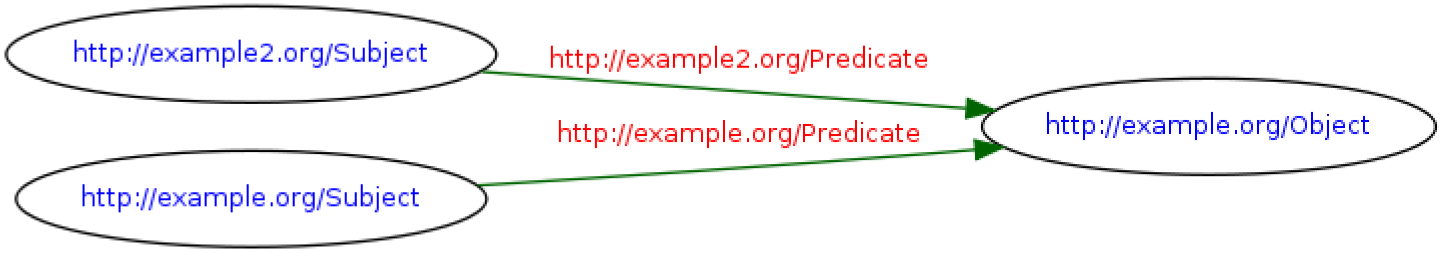
The original RDF/XML document

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:j.0="http://example.org/" xmlns:j.1="http://example2.org/">
  <rdf:Description rdf:about="http://example2.org/Subject">
    <j.1:Predicate rdf:resource="http://example2.org/Object"/>
  </rdf:Description>
  <rdf:Description rdf:about="http://example.org/Subject">
    <j.0:Predicate rdf:resource="http://example.org/Object"/>
  </rdf:Description>
</rdf:RDF>
```


http://example.org/Subject
http://example.org/Predicate
http://example.org/Object
http://example2.org/Subject
http://example2.org/Predicate
http://example2.org/Object

Interactions with a Triple Store - pt 1 – Another simple example….

```
store = new.rdf(ontology=FALSE)

add.triple(store,
           subject="http://example.org/Subject",
           predicate="http://example.org/Predicate",
           object="http://example.org/Object"
)

add.triple(store,
            subject="http://example2.org/Subject",
            predicate="http://example2.org/Predicate",
            object="http://example.org/Object"
 )


 save.rdf(store, "store_Ex.xml", "RDF/XML")
```

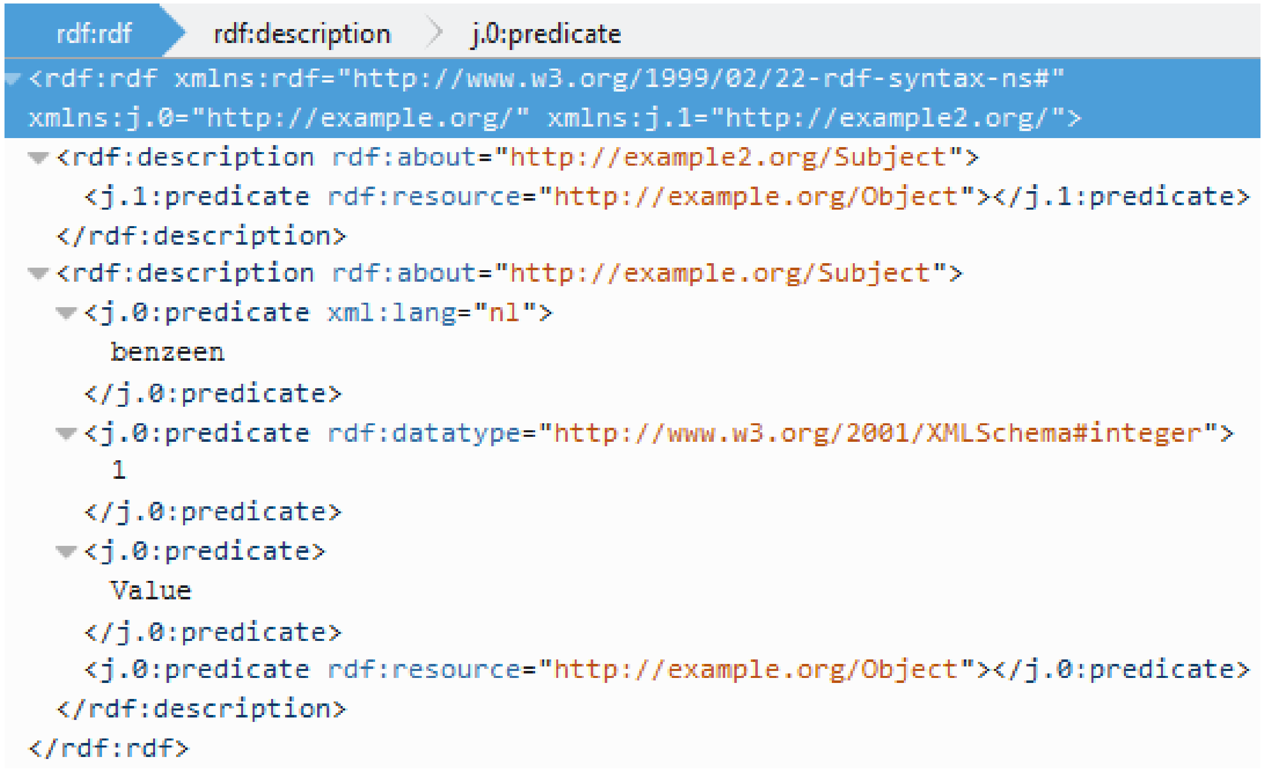

http://example2.org/Subject
http://example2.org/Predicate
http://example.org/Predicate
http://example.org/Object
http://example.org/Subject

Interactions with a Triple Store - pt 2

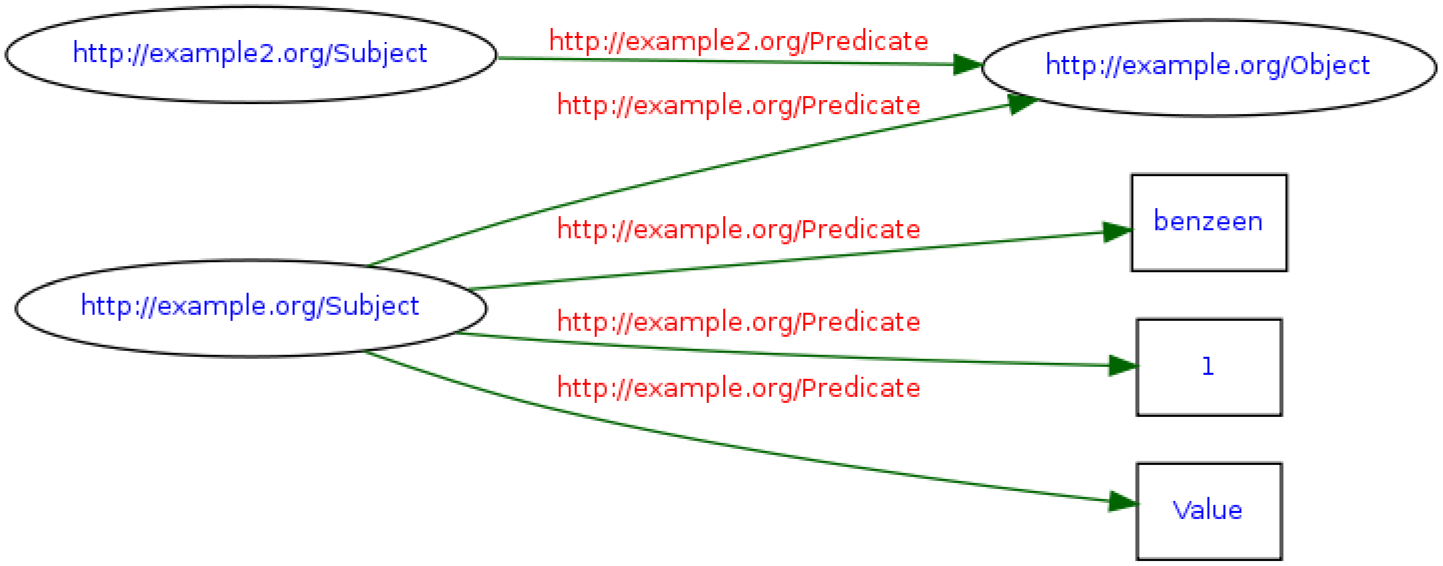
add.data.triple adds an object property to the model.

store	A triple store, for example created with new.rdf().
subject	URI of the subject.
predicate	URI of the predicate.
data	A data value.
type	Optional parameter for the data value type. Can be "**string**", "**float**", "**double**", or **any other XML Schema Data Type**. **It cannot be used at the same time as lang.**
lang	Optional parameter with the two-letter code for the language of the literal content, for example "**en**".

```
store = new.rdf(ontology=FALSE)

add.triple(store, subject="http://example.org/Subject",
           predicate="http://example.org/Predicate",
           object="http://example.org/Object" )

add.triple(store, subject="http://example2.org/Subject",
            predicate="http://example2.org/Predicate",
            object="http://example.org/Object"  )


 add.data.triple(store, subject="http://example.org/Subject",
                 predicate="http://example.org/Predicate",
                 data="Value")
 add.data.triple(store, subject="http://example.org/Subject",
                 predicate="http://example.org/Predicate",
                 data="1", type="integer")
 add.data.triple(store, subject="http://example.org/Subject",
                 predicate="http://example.org/Predicate",
                 data="benzeen", lang="nl")

 save.rdf(store, "store_Ex.xml", "RDF/XML")
```

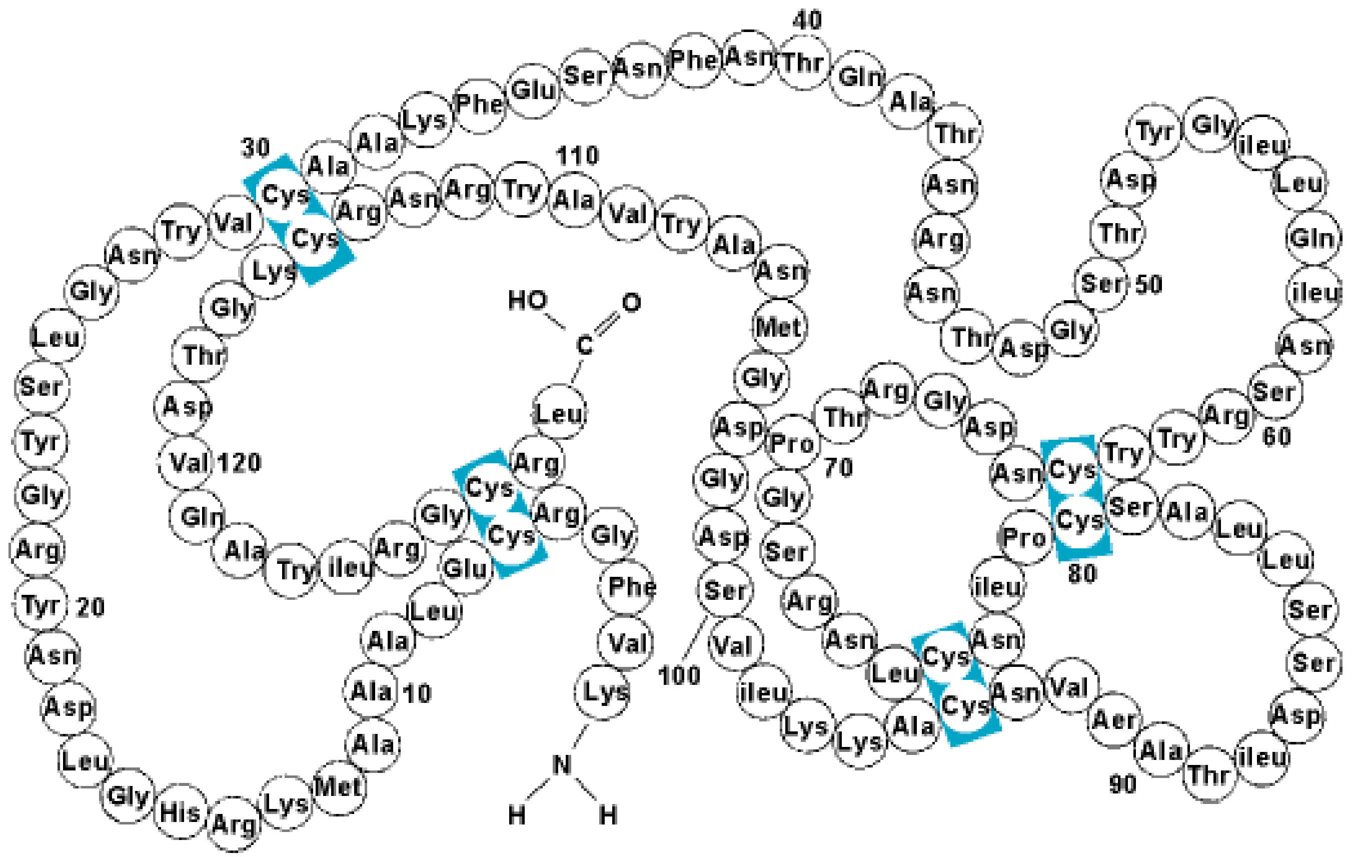


```
<rdf:rdf xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:j.0="http://example.org/" xmlns:j.1="http://example2.org/">
  <rdf:description rdf:about="http://example2.org/Subject">
    <j.1:predicate rdf:resource="http://example.org/Object"></j.1:predicate>
  </rdf:description>
  <rdf:description rdf:about="http://example.org/Subject">
    <j.0:predicate xml:lang="nl">
      benzeen
    </j.0:predicate>
    <j.0:predicate rdf:datatype="http://www.w3.org/2001/XMLSchema#integer">
      1
    </j.0:predicate>
    <j.0:predicate>
      Value
    </j.0:predicate>
    <j.0:predicate rdf:resource="http://example.org/Object"></j.0:predicate>
  </rdf:description>
</rdf:rdf>
```

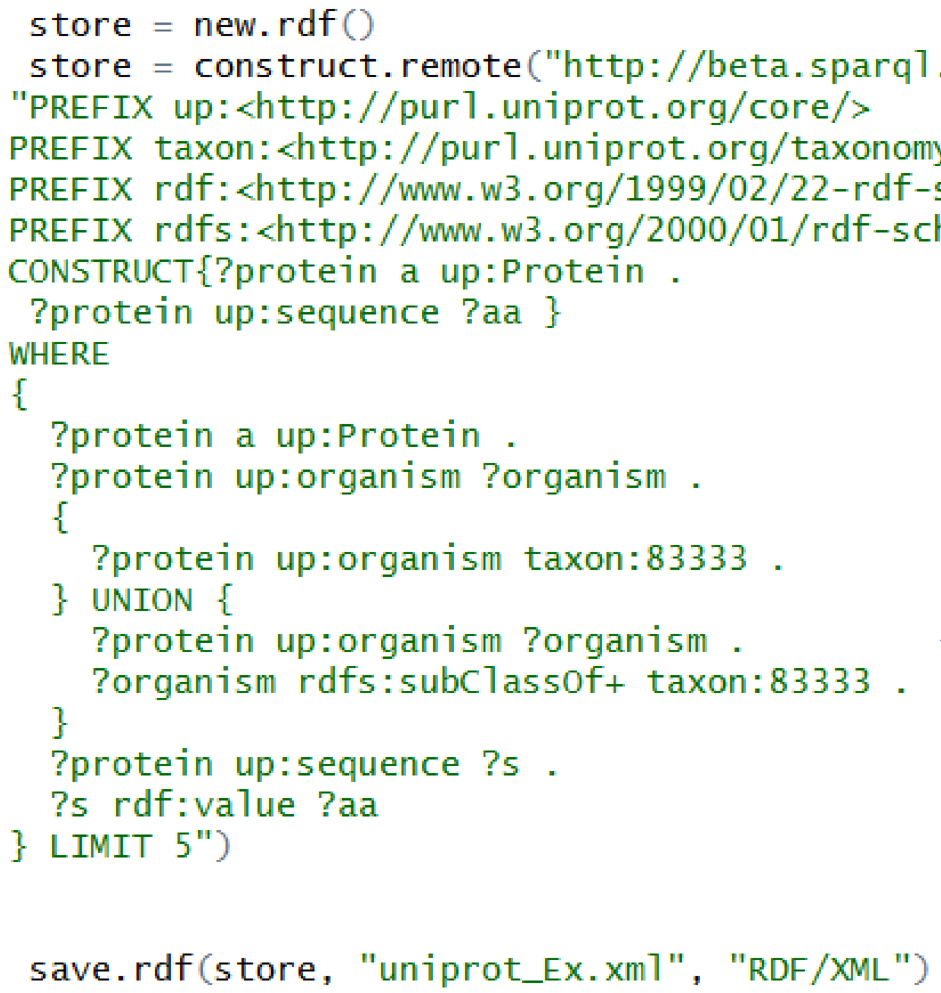

http://example2.org/Subject
http://example2.org/Predicate
http://example.org/Object
http://example.org/Predicate
benzeen
http://example.org/Predicate
http://example.org/Subject
http://example.org/Predicate
1
http://example.org/Predicate
Value

Build a new model from a SPARQL query :

A protein is described with an alphabet of 20 letters (amino acids) .



Build a new model from a SPARQL query : **Run a SPARQL CONSTRUCT query on a SPARQL end point and construct a new model with the results.**
UniProt End Point : **http://beta.sparql.uniprot.org/sparql/**.

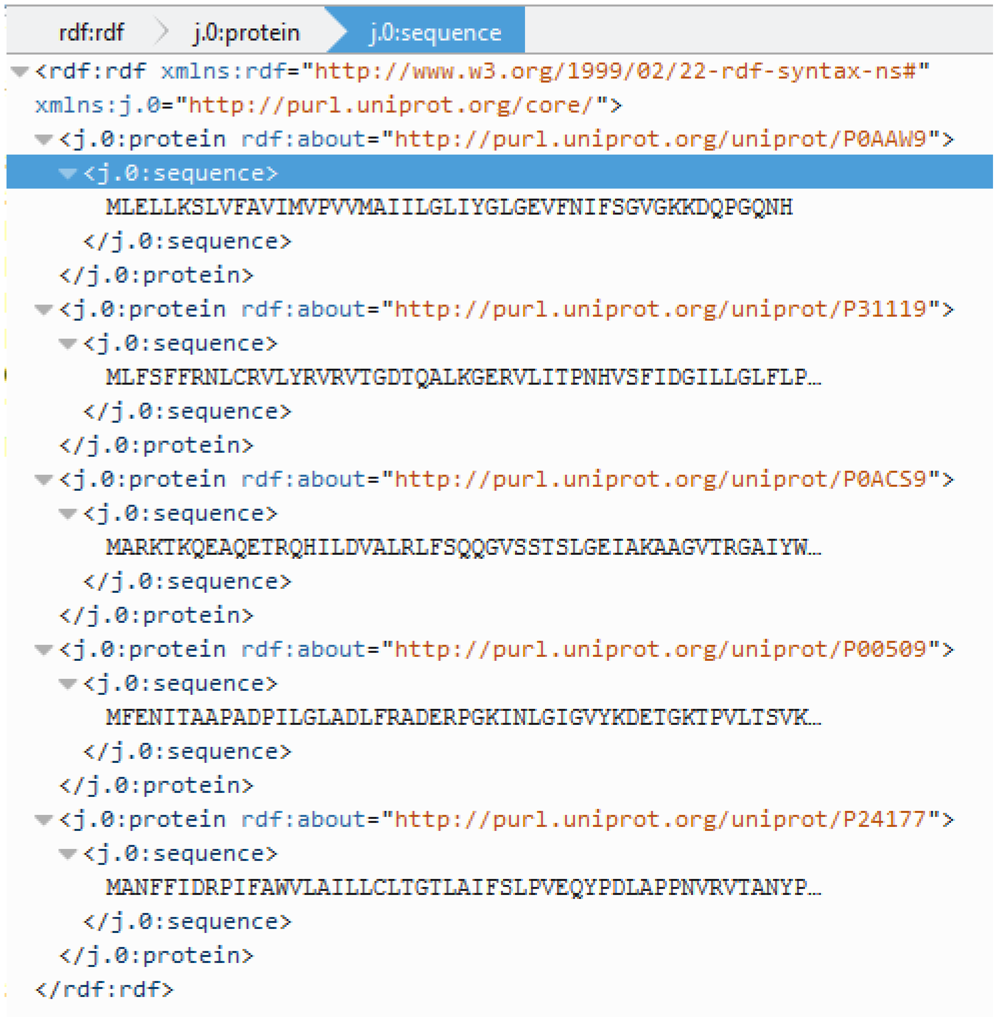


```
 store = new.rdf()
 store = construct.remote("http://beta.sparql.uniprot.org/sparql/",
"PREFIX up:<http://purl.uniprot.org/core/>
PREFIX taxon:<http://purl.uniprot.org/taxonomy/>
PREFIX rdf:<http://www.w3.org/1999/02/22-rdf-syntax-ns#>
PREFIX rdfs:<http://www.w3.org/2000/01/rdf-schema#>
CONSTRUCT{?protein a up:Protein .
 ?protein up:sequence ?aa }
WHERE
{
  ?protein a up:Protein .
  ?protein up:organism ?organism .
  {
    ?protein up:organism taxon:83333 .
  } UNION {
    ?protein up:organism ?organism .
    ?organism rdfs:subClassOf+ taxon:83333 .
  }
  ?protein up:sequence ?s .
  ?s rdf:value ?aa
} LIMIT 5")


 save.rdf(store, "uniprot_Ex.xml", "RDF/XML")
```

EndPoint

SPARQL query

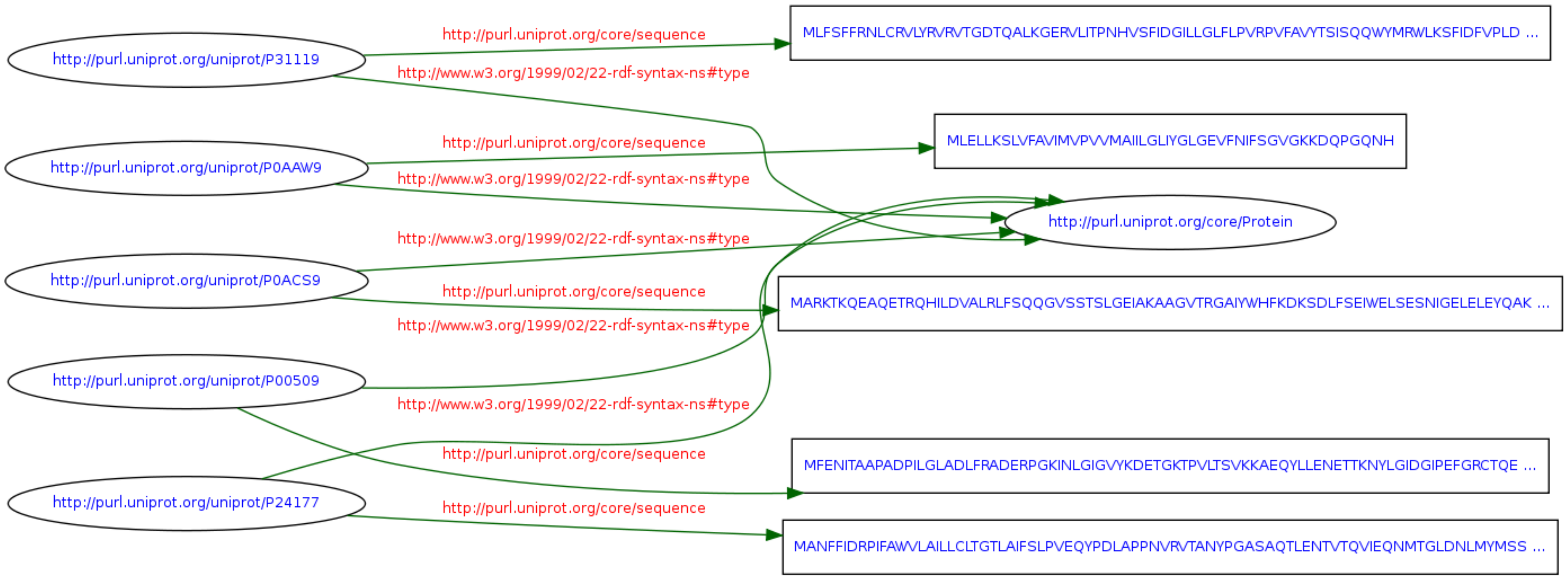
Select 5 Escherichia Coli K12 (taxon:83333) (including strains) UniProt entries and their amino acid sequence.



rdf:rdf › j.0:protein › j.0:sequence

```
<rdf:rdf xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
xmlns:j.0="http://purl.uniprot.org/core/">
  <j.0:protein rdf:about="http://purl.uniprot.org/uniprot/P0AAW9">
    <j.0:sequence>
      MLELLKSLVFAVIMVPVVMAIILGLIYGLGEVFNIFSGVGKKDQPGQNH
    </j.0:sequence>
  </j.0:protein>
  <j.0:protein rdf:about="http://purl.uniprot.org/uniprot/P31119">
    <j.0:sequence>
      MLFSFFRNLCRVLYRVRVTGDTQALKGERVLITPNHVSFIDGILLGLFLP…
    </j.0:sequence>
  </j.0:protein>
  <j.0:protein rdf:about="http://purl.uniprot.org/uniprot/P0ACS9">
    <j.0:sequence>
      MARKTKQEAQETRQHILDVALRLFSQQGVSSTSLGEIAKAAGVTRGAIYW…
    </j.0:sequence>
  </j.0:protein>
  <j.0:protein rdf:about="http://purl.uniprot.org/uniprot/P00509">
    <j.0:sequence>
      MFENITAAPADPILGLADLFRADERPGKINLGIGVYKDETGKTPVLTSVK…
    </j.0:sequence>
  </j.0:protein>
  <j.0:protein rdf:about="http://purl.uniprot.org/uniprot/P24177">
    <j.0:sequence>
      MANFFIDRPIFAWVLAILLCLTGTLAIFSLPVEQYPDLAPPNVRVTANYP…
    </j.0:sequence>
  </j.0:protein>
</rdf:rdf>
```

uniprot_Ex.xml


http://purl.uniprot.org/uniprot/P31119
http://purl.uniprot.org/uniprot/P0AAW9
http://purl.uniprot.org/uniprot/P0ACS9
http://purl.uniprot.org/uniprot/P00509
http://purl.uniprot.org/uniprot/P24177
http://purl.uniprot.org/core/sequence
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://purl.uniprot.org/core/sequence
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://purl.uniprot.org/core/sequence
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://www.w3.org/1999/02/22-rdf-syntax-ns#type
http://purl.uniprot.org/core/sequence
http://purl.uniprot.org/core/sequence
MLFSFFRNLCRVLYRVRVTGDTQALKGERVLITPNHVSFIDGILLGLFLPVRPVFAVYTSISQQWYMRWLKSFIDFVPLD ...
MLELLKSLVFAVIMVPVVMAIILGLIYGLGEVFNIFSGVGKKDQPGQNH
http://purl.uniprot.org/core/Protein
MARKTKQEAQETRQHILDVALRLFSQQGVSSTSLGEIAKAAGVTRGAIYWHFKDKSDLFSEIWELSESNIGELELEYQAK ...
MFENITAAPADPILGLADLFRADERPGKINLGIGVYKDETGKTPVLTSVKKAEQYLLENETTKNYLGIDGIPEFGRCTQE ...
MANFFIDRPIFAWVLAILLCLTGTLAIFSLPVEQYPDLAPPNVRVTANYPGASAQTLENTVTQVIEQNMTGLDNLMYMSS ...

The example of Gene Ontolgy

The Scope of GO

The Gene Ontology (GO) project is a collaborative effort to address the need for consistent descriptions of **gene products** across databases.

The following areas are outside the scope of GO, and terms in these domains will not appear in the ontologies:

Gene products: e.g. cytochrome c is not in the ontologies, but attributes of cytochrome c, such as oxidoreductase activity, are.

Processes, functions or components that are unique to mutants or diseases: e.g. **oncogenesis** is not a valid GO term, as "causing cancer" is the result of reprogrammed, not normal cells and thus it is not the normal function of a gene.

Attributes of sequence such as "intron" or "exon" parameters belong in a separate sequence ontology;

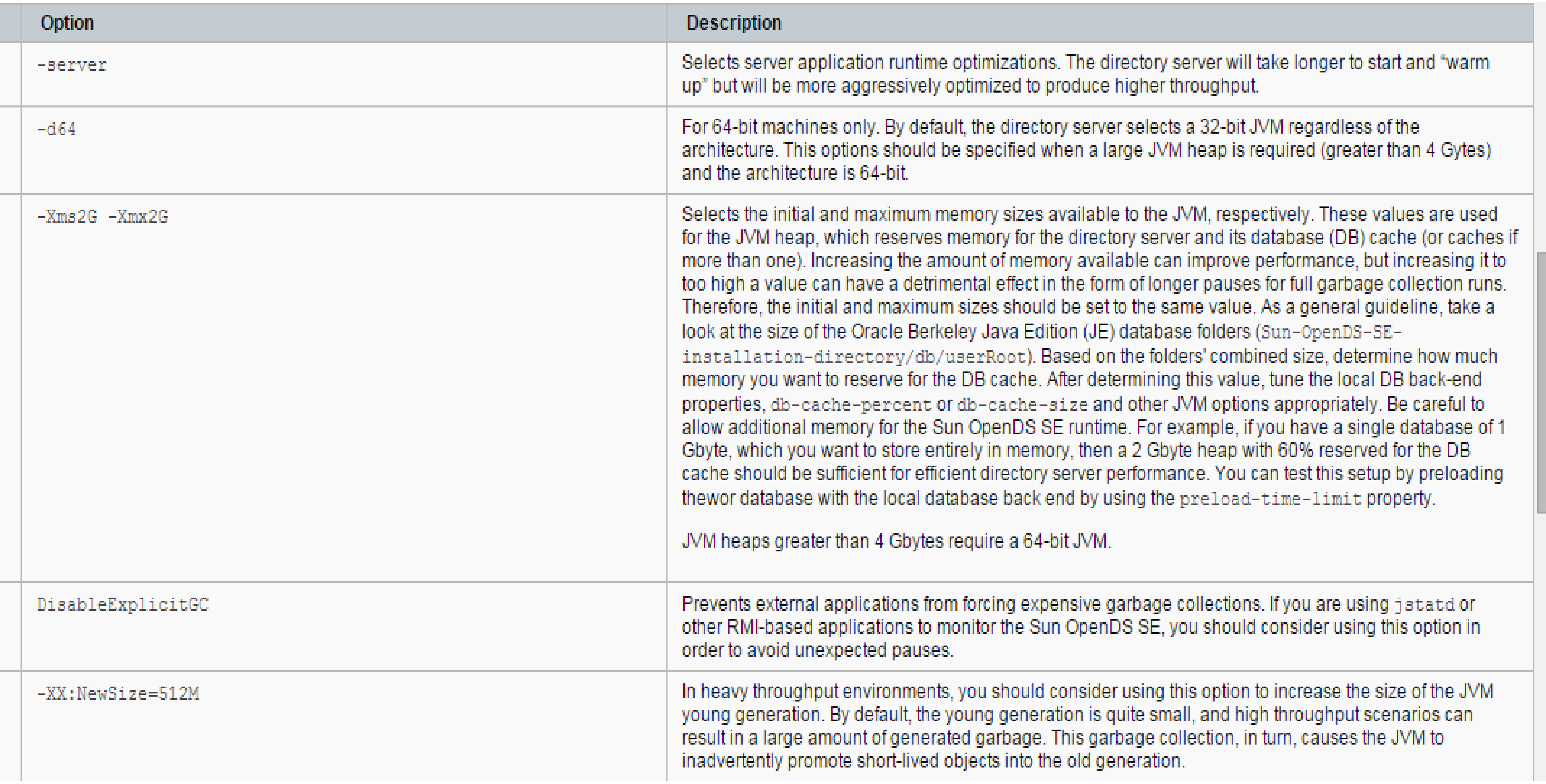
Protein domains or structural features.

Protein-protein interactions.

Environment, evolution and expression.

Anatomical or histological features above the level of cellular components, including cell types.

rJava: Since GO is very large we have to change some parameters of the JVM



Option	Description
`-server`	Selects server application runtime optimizations. The directory server will take longer to start and "warm up" but will be more aggressively optimized to produce higher throughput.
`-d64`	For 64-bit machines only. By default, the directory server selects a 32-bit JVM regardless of the architecture. This options should be specified when a large JVM heap is required (greater than 4 Gytes) and the architecture is 64-bit.
`-Xms2G -Xmx2G`	Selects the initial and maximum memory sizes available to the JVM, respectively. These values are used for the JVM heap, which reserves memory for the directory server and its database (DB) cache (or caches if more than one). Increasing the amount of memory available can improve performance, but increasing it to too high a value can have a detrimental effect in the form of longer pauses for full garbage collection runs. Therefore, the initial and maximum sizes should be set to the same value. As a general guideline, take a look at the size of the Oracle Berkeley Java Edition (JE) database folders (`Sun-OpenDS-SE-installation-directory/db/userRoot`). Based on the folders' combined size, determine how much memory you want to reserve for the DB cache. After determining this value, tune the local DB back-end properties, `db-cache-percent` or `db-cache-size` and other JVM options appropriately. Be careful to allow additional memory for the Sun OpenDS SE runtime. For example, if you have a single database of 1 Gbyte, which you want to store entirely in memory, then a 2 Gbyte heap with 60% reserved for the DB cache should be sufficient for efficient directory server performance. You can test this setup by preloading thewor database with the local database back end by using the `preload-time-limit` property. JVM heaps greater than 4 Gbytes require a 64-bit JVM.
`DisableExplicitGC`	Prevents external applications from forcing expensive garbage collections. If you are using `jstatd` or other RMI-based applications to monitor the Sun OpenDS SE, you should consider using this option in order to avoid unexpected pauses.
`-XX:NewSize=512M`	In heavy throughput environments, you should consider using this option to increase the size of the JVM young generation. By default, the young generation is quite small, and high throughput scenarios can result in a large amount of generated garbage. This garbage collection, in turn, causes the JVM to inadvertently promote short-lived objects into the old generation.

JVM heaps greater than 4 Gbytes require a 64-bit JVM.
http://docs.oracle.com/cd/E19450-01/820-6168/configuring-default-jvm.html

```r
#Since GO is very large we have to change some parameters of the JVM.
install.packages("rJava") # if not present already

if (Sys.getenv("JAVA_HOME")!="")
  Sys.setenv(JAVA_HOME="")

library(rJava)

options( java.parameters = "-Xmx4g" )
.jinit()

#To check the result:
.jcall(.jnew("java/lang/Runtime"), "J", "maxMemory")


#Now it is possible to work with large ontologies like GO
```

```r
source("http://bioconductor.org/biocLite.R")
biocLite("ontoCAT")
```

```r
# Get GO ontology
library(ontoCAT)

# Obtain GO ontology
go <- getOntology("http://www.geneontology.org/ontology/obo_format_1_2/gene_ontology_ext.obo")
```

```
INFO [ReasonedFileOntologyService] Classified the ontology
http://www.geneontology.org/ontology/obo_format_1_2/gene_ontology_ext.obo
```

The ontoCAT package:

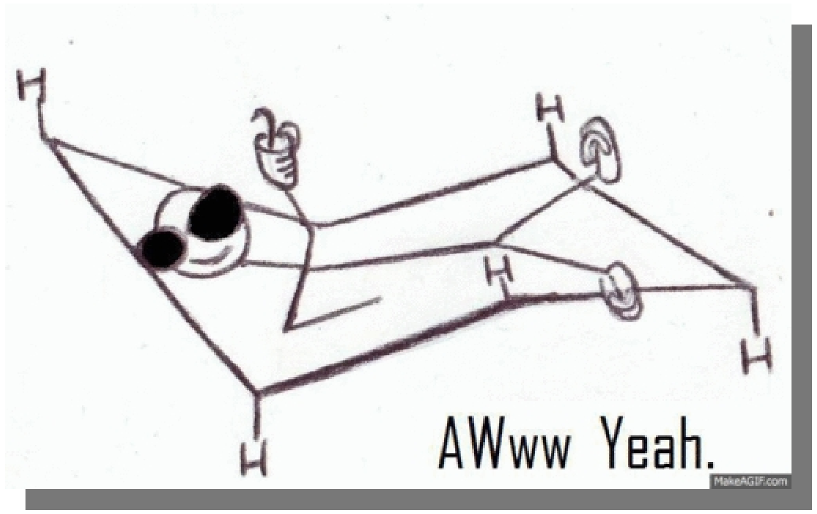
• **gives unified, format-independent access to ontology terms and the ontology hierarchy represented in OWL and OBO formats**;

• **provides basic methods for ontology traversal**, such as searching for terms, listing a specific term's relations, showing paths to the term from the root element of the ontology, showing flattened-tree representations of the ontology hierarchy; and

• **supports working with groups of ontologies and with major public ontology repositories:** searching for terms across ontologies, listing available ontologies and loading ontologies for further analysis as necessary.

The ontoCAT package can load an ontology in OWL or OBO format from a local file or on-the-fly from a URI. Reasoning over ontologies and extracting relationships is supported by using **HermiT** (Motik et al., 2009) reasoner. OBO ontologies are translated by OWL API (Horridge and Bechhofer, 2009) into valid OWL format that can be reasoned over. Ontologies can also be loaded from ontology repositories.

Some features seem to not work properly…
but R packages are open source and we are informatics.



[1] Accessing biological data in R with semantic web Technologies , Egon L. Willighagen - 2014

[2] Package ‘rrdf’ and ‘rrdflib’, Egon Willighagen – 2013

[3] Precise generation of systems biology models from KEGG pathways - Clemens Wrzodek et all – 2013

[4] OpenTox Predictive Toxicology Framework: toxicological ontology and semantic media wiki-based OpenToxipedia - Olga Tcheremenskaia et all – 2011

[5] The UniProt Consortium UniProt: a hub for protein information - Nucleic Acids Res. – 2015

[6] ontoCAT: an R package for ontology traversal and search - Natalja Kurbatova et all. - 2009