

# SPARQL2ws: Manual

## 1 Introduction

SPARQL2ws is a piece of software that integrates with the widely used Sesame<sup>1</sup> framework for semantic web applications. Its development was part of a thesis written at the WeST institute of the University of Koblenz-Landau. The key component of SPARQL2ws is WSSail. The purpose of this storage and inference layer (Sail) is to allow semantic web applications to query legacy information systems (accessible via SOAP or RESTful web services) by means of SPARQL. The WSSail acts as a wrapper for such information systems so that it can be queried the same way as Sesame-Sails in general. The WSSail therefore translates submitted SPARQL queries into a sequence of web service calls. Subsequently received data is then analysed and transformed such that it fits the respective query. In order to do that the right way the WSSail has to be configured appropriately with the used web services in mind.

## 2 Installing SPARQL2ws

To make use of SPARQL2ws, you may build your own custom application using only those classes from both SPARQL2ws and the Sesame framework needed. The more obvious approach however is to integrate an existing build of SPARQL2ws with an up and running Sesame server. In the following, this is illustrated for a Sesame system running ontop of Apache Tomcat. Before continuing you may want to make yourself familiar with the setup of Tomcat and Sesame<sup>2</sup>.

### Prerequisites

- Sesame 2, up and running ontop of Tomcat
- copy of Saxon-HE 9.2<sup>3</sup>
- working implementation of JAX-RS<sup>4</sup>

A package of all libraries needed can be obtained here.

---

<sup>1</sup><http://www.openrdf.org/>

<sup>2</sup><http://www.openrdf.org/doc/sesame2/users/ch06.html>

<sup>3</sup><http://saxon.sourceforge.net/#F9.2HE>

<sup>4</sup><https://jersey.dev.java.net/>

To use SPARQL2ws inside a Sesame system running on top of a Tomcat server, do the following:

1. unzip the package - you'll get 2 folders: **LIB** and **TEMPLATES**
2. copy all jar-files inside **LIB** to  
`%SERVER_HOME%/webapps/openrdf-sesame/WEB-INF/lib`
3. copy **LIB/SPARQL2ws-1.0.jar** to the lib-folder of your Sesame-Console
4. copy all ttl-files inside **TEMPLATES** to the **template**-folder inside the working directory of the Sesame-Console (this directory will be created after running Sesame-Console for the first time); you may also create your own repository template at this point
5. start your Tomcat server and run Sesame-Console
6. inside the console enter: `'connect http://my-host.org/openrdf-sesame.'`
7. when successfully connected, create a Sail-Repository fitting one of the templates by entering: `'create [ttl-name].'` ... then follow the command prompt
8. if the repository has been established, it should be available for querying

### 3 Configuring the WSSail

All settings necessary to enable the WSSail are passed on to the Sail in the form of a configuration file. Such a configuration file itself resembles a RDF graph, that has been serialised using the Turtle format.

When creating a new WSSail repository, the instantiated Sail needs to know where to find its configuration file. Therefore a string containing an URL or filesystem location has to be passed at this point. This can be done by declaring this location inside the respective repository template. Alternatively the template can be set up such that the location may be passed by answering the prompt of the Sesame console.

There are a number of concerns a configuration of WSSail needs to take care of. For one, it needs to sufficiently describe all parts of the web service to be used. That is, terms of how to access the web service and its methods (e.g. what parameters certain methods expect, etc.). Two, it needs to introduce a domain model consisting of resources and their properties. These properties are then being used inside of actual queries on a number of the model's resources. And three, it needs to map these properties onto certain methods. At this point, there are two different kinds of properties to be distinguished: detail properties and matching properties. Query patterns containing detail properties link a subject variable to an object variable, assuming that a number of instances for the subject have already been generated (thus being used to find object instances). Matching properties however are used in triple patterns that link a subject variable to an object instance. The value of the object in such a pattern is used to assign exactly one parameter of a search method, thus partially instantiating a call of such a method. Responses to a call like that are then being used to find solutions of the subject variable.

```

@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix rep: <http://www.openrdf.org/config/repository#>.
@prefix sr: <http://www.openrdf.org/config/repository/sail#>.
@prefix sail: <http://www.openrdf.org/config/sail#>.
@prefix ms: <http://www.openrdf.org/config/sail/memory#>.
@prefix ws: <http://example.org/schema/SPARQL2ws#>. (A)

[] a rep:Repository ;
  rep:repositoryID "{%Repository ID|WSStore%}";
  rdfs:label "{%Repository title|Web Service Store%}";
  rep:repositoryImpl [
    rep:repositoryType "openrdf:SailRepository";
    sr:sailImpl [
      sail:sailType "isweb:WSSail"; (B)
      ws:config-location "{%Config Location(file / HTTP)|
                          http://example.org/webservice.ttl%}"; (C)
      sail:delegate [ sail:sailType "openrdf:MemoryStore" ]
    ]
  ]
].

```

Figure 1: Example of a template for a WSSail repository. (A): namespace declaration, specific to the web service in use; (B): the repository's sail must be of type "*isweb:WSSail*"; (C): declaration of the configuration files location (a string of the form "*{%. . |[default value]%}*" is interpreted as an input request on the prompt when creating the repository).

```

> connect http://localhost:8080/openrdf-sesame.
Disconnecting from default data directory
Connected to http://localhost:8080/openrdf-sesame
> create WSStore.
Please specify values for the following variables:
Repository ID [WSStore]: wsRepository
Repository title [Web Service Store]:
Config Location(file / HTTP) [http://example.com/webservice.ttl]:
Repository created
> exit.

```

Figure 2: Example illustrating a typical console session: a new WSSail repository *ws-Repository* is being created with the above template and a configuration located at default location *http://example.com/webservice.ttl*.

```

@prefix c: <http://demetrius.isweb.uni-koblenz.de/schema/SPARQL2ws#>.
@prefix i: <http://www.youtube.com/api/>.
@prefix yt: <http://www.youtube.com/schema#>.

```

---

```

c:schema c:resource yt:video

yt:video
  c:nameSpace "www://example.org/video/";
  c:matchingGroup yt:VIDEOMATCH;
  c:detailProperty yt:has-date, yt:has-title.

yt:VIDEOMATCH
  c:xpath "/fn:replace(atm:id, '^tag:youtube.com,2008:video:', '')".
  c:method i:VIDEOSEARCH;
  c:parameter [ c:matchingProperty yt:QUERY; c:qualifiedName "q"],
              [ c:matchingProperty yt:LOCATION; c:qualifiedName "location"],
              [ c:matchingProperty yt:RADIUS; c:qualifiedName "radius"];

yt:has-date
  c:xpath "/fn:replace(atm:published, 'T.*Z$', '')".
  c:method i:VIDEOSEARCH;

yt:has-title
  c:xpath "/atm:title".
  c:method i:VIDEOSEARCH;

```

---

```

c:interface
  c:method i:VIDEOSEARCH, i:VIDEOINFO.
  c:requestFormat c:REST;
  c:xmlNamespace [ c:prefix "atm";
                  c:nameSpace "http://www.w3.org/2005/Atom"
                ].

  i:VIDEOSEARCH
    c:baseXPath "//atm:entry";
    c:url [ c:template "http://gdata.youtube.com/.../videos"; c:style c:QUERY ];
    c:staticParameter [ c:qualifiedName "v"; c:value "2" ],
                      [ c:qualifiedName "max-results"; c:value "50" ].

  i:VIDEOINFO c:url [
    c:template "http://gdata.youtube.com/.../videos/{video_id}?v=2";
    c:style c:SUBSTITUTE
  ].

```

Figure 3: Example of a WSSail configuration enabling the Youtube API for querying.

## 4 Configuration - Syntax

The following fragments are graph patterns, that are allowed to be used in concrete configurations of a WSSail. The syntactic form of these fragments is based on Turtle, since currently only Turtle files can be parsed. Be aware that braces ("{" , "}") enclose those subexpressions that are further quantified ("(\*)": zero or more, "(?)": optional) and are not part of the actual syntax itself.

### 4.1 Interface Description

#### Interface Settings

(graph pattern:)

```
c:interface c:requestFormat Format
{ c:interface c:method Method. }(*)
{ c:interface c:apiKey [
    c:qualifiedName AKVar;
    c:value APIKey ]. }(?)
{ c:interface c:retryCount Retries. }(?)
```

- Communication format Format (possible values c:REST, c:SOAP).
- Methods provided by the interface, identified by Method.
- Optional API key assignment with variable identifier AKVar and value APIKey.
- Maximum possible number of connection retries Retries.

#### Further Interface Settings

```
{ c:interface c:soapSettings [
    c:endpoint Endpoint;
    c:requestName RequestName;
    c:requestPrefix RequestPrefix;
    c:requestNameSpace RequestNamespace ]. }(?)
{ c:interface c:xmlNamespace [
    c:prefix Prefix;
    c:nameSpace XMLns ]. }(*)
{ c:interface c:httpStatusHandling [
    c:statusCode Code;
    c:action Action ]. }(*)
```

- Possibly necessary SOAP settings with endpoint address Endpoint, XML name RequestName and XML namespace RequestNamespace with prefix RequestPrefix for SOAP requests.
- Optional namespace declaration (prefix Prefix, XML-namespace XMLns) for XQuery processing of responses.

- Optional mapping of HTTP statuscode *Code* onto an action identified by *Action* - by now *Action* may only be *c:CALL\_ESCAPE*, which signals a complete connection abort.

## Method

```
{ Method c:url [
    c:template Url;
    c:style UrlStyle ].}(?)
{ Method c:methodName [
    c:qualifiedName NameVar;
    c:value Name ]. }(?)
{ Method c:pagination [
    c:qualifiedName PageVar;
    c:pages Pages].}(?)
{ Method c:staticParameter [
    c:qualifiedName ParamName;
    c:value ParamValue ].}(*)
{ Method c:baseXPath BaseXPath. }(?)
```

- Possibly necessary URL *Url* in case of a RESTful web service. String *Url* may contain a variable part (enclosed in "{"","}") that is to be replaced by certain values for call instances. In this case the style of this URL *UrlStyle* has to be set to *c:SUBSTITUTE*. Otherwise all parameter assignments are encoded inside the query string of call URLs (in which case URL style has to be *c:QUERY*).
- Optional setting for concurrent calls requesting a range of numbered documents (pagination): declaration of identifier *PageVar* of the page parameter and the number of pages to be requested, *Pages*.
- Optional static parameter constantly set for every single call of *Method*: declaration of the page-parameter's identifier *ParamName* and its value *ParamValue*.
- Possibly necessary base XPath for search methods, declared by *BaseXPath*. This may be necessary in case there exists a 1:N relationship between elements of a search result and one of their attributes you're interested in. When processing an element in the list via XQuery, a base XPath is then used to constrain the extent of a join. This prevents global joins on the document as a whole, which would yield unintended results.

## 4.2 Schema Mapping

### Resource Mapping

```
c:schema c:resource Resource.
Resource c:nameSpace WSNamespace.
{ Resource c:detailProperty DProperty. }(*)
{ Resource c:matchingGroup MGroup. }(*)
```

- Declaration of URI Resource identifying a type of resource of the domain model.
- Web service specific namespace WSNamespace corresponding to Resource; WSNamespace will be put in front of all IDs identifying instances of Resource, that are extracted from response data.
- ID of (optional) detail property DProperty associated with Resource
- ID of "matching group" MGroup associated with Resource. Matching groups are used for collation of matching properties, that each instantiate one parameter of the same search method. Such a search method has to be capable of finding instances of Resource.

### Matching Properties

```

MGroup c:method Method; c:xpath XPath.
{ MGroup c:parameter [
    c:qualifiedName ParamName;
    c:matchingProperty MProperty; c:isURI B
]. }(*)

```

- Method Method associated with matching group MGroup and used to retrieve data containing instances of the very resource having this group of matching properties.
- Obligatory declaration of XPath XPath needed for extraction of such instances from XML documents requested through calls of Method.
- Particular matching property MProperty belonging to MGroup:
  - Identifier ParamName identifying a certain parameter of Method. This parameter is to be set using objects linked by MProperty inside a query.
  - Arbitrary object B. If B exists inside the subgraph, the objects linked by MProperty are meant to be URIs - only their local names are being processed further.

### Detail Properties

```

DProperty c:method Method; c:xpath XPath.
{ DProperty c:qualifiedName ParamName. }(?)
{ DProperty c:setResource Resource. }(?)

```

- Method Method mapped onto detail property type DProperty. Calls of Method are used to retrieve data containing solutions of the very query variables linked by DProperty.
- Obligatory declaration of XPath XPath needed for extraction of such solutions from XML documents requested through calls of Method.

- Optional identifier *ParamName* identifying a certain parameter of *Method*. This parameter will be set using instances of the very resource having detail property *DProperty*.
- Optional declaration of resource type *Resource*, in case the detail object to be instantiated is of this resource type.

## 5 Querying the WSSail

A typical query passed on to a WSSail is as follows:

```

PREFIX yt:<http://www.youtube.com/schema#>
SELECT * WHERE {
  ?video yt:QUERY 'berlin'.
  ?video yt:LOCATION '52.52,13.46'.
  ?video yt:RADIUS '15km'.

  ?video yt:has-title ?title.
  ?video yt:has-date ?date.
}

```

Figure 4: Example query applied to a WSSail repository, that has been set up with a configuration similar to the one in 3.

There are a few constraints that one has to be aware of:

- all triple patterns share the same subject variable
- all matching properties instantiate a call of the same search method
- there may only be one search method implied per query
- due to the way queries are parsed, matching properties inside the query expression have to come first

If such a query has been successfully parsed, a set of web service calls will be instantiated as specified inside the respective configuration. The corresponding responses are then parsed and analysed for certain parts of data that are thought of as instances of certain resources or attributes. This analysis and the subsequential extraction is based on XQuery using the XPath declarations mapped onto such resources inside the configuration.

Results of a query are represented as pairs of variable identifiers and values. The identifiers are those used inside the query. Identifiers of resource variables are completed by namespace URIs, also declared inside the configuration in association with these resources. The values are those data extracted from call responses that fit as solutions to particular variables.

<b>video</b>	<b>title</b>	<b>date</b>
<www://example.org/mpkrdU0L8K8>	German New Yorker in Berlin	2010-06-24
<www://example.org/Bw_A8C2F1v4>	I Heart Sharks - Wolves	2010-06-26
	Live 25.6.2010 Berlin Magnet Club	
<www://example.org/1_eCVhCGYwE>	The opening of the Wall at Berlin Bornholmer Strasse 1989	2007-08-05
<www://example.org/G7Ly2AY-TUc>	Berlin Block Tetris	2009-09-24
<www://example.org/lkTMMJZI65M>	Berlin in 3D for Google Earth	2007-03-09
<www://example.org/ciXarXYjP-g>	Crossing the Berlin border on the S-Bahn (late '80s)	2006-07-12
<www://example.org/AouNl1ts20E>	Berlin - Checkpoint Charlie	2007-04-15
<www://example.org/dW81gl_LS-U>	BERLIN - Serafinale	2006-10-31
<www://example.org/_ab7Dksqfnw>	Berlin Riot 1999 (Atari Teenage Riot LIVE)	2006-04-22
...		

Figure 5: Example query applied to a WSSail repository set up with configuration similar to the one in 3.