



SPARQLMotion™

Tutorial

Version 3.0

May 26, 2011

Revision History

Date	Version	Revision
August 27, 2008	1.0	
March 26, 2009	1.1	Corrected various errata Exercise 2 – added section on Preference for the max. number of triples displayed Added explanations on results in exercises 2 & 3
April 1, 2009	2.0	Updated Web Services exercises and some screen shots to SPARQLMotion 2.0 and TBC-ME 3.0
May 23, 2008	2.0.1	Updated URL for web services from “servlet” to “actions”
June 6, 2009	2.0.2	Various errata, added sections on Help files, URIs and qnames.
May 21, 2011	3.0	Updated to align with 3.5 release

Copyright © 2008-2011 TopQuadrant, Inc. All rights reserved. “SPARQLMotion”, “TopBraid”, “TopBraid Composer”, “TopBraid Suite” and “TopBraid Live” are trademarks of TopQuadrant, Inc.

Table of Contents

TUTORIAL	1
VERSION 3.0	1
1 INTRODUCTION	4
1.1 CONVENTIONS	4
1.2 ASSUMPTIONS	5
1.3 TOPBRAID COMPOSER HELP FILES	5
1.4 COMPLETED EXERCISES FOR THIS TUTORIAL ARE AVAILABLE	5
2 SPARQLMOTION MODULES	6
3 CREATING SIMPLE DATA PROCESSING SEQUENCES	7
3.1 EXERCISE 1	7
3.2 EXERCISE 2	10
3.3 EXERCISE 3	13
4 ITERATIVE DEVELOPMENT OF SPARQLMOTION SCRIPTS	15
4.1 EXERCISE 4	15
4.2 EXERCISE 5	22
5 SPARQLMOTION AND WEB SERVICES	23
5.1 EXERCISE 6	23
6 EXTENDING SPARQLMOTION	26
6.1 EXERCISE 7	26
7 NEXT STEPS	28

1 Introduction

SPARQLMotion™ is a visual scripting language and engine for semantic data processing. It is fully compliant with W3C standard languages SPARQL, RDF, and OWL. SPARQLMotion scripts can be assembled graphically by people who understand data flow processes and can create queries, but who are not necessarily programmers. Script developers can chain together simple processing steps to form sophisticated processing pipelines. Assembled data-processing pipelines are used to merge, search, query, and mash-up data. As a result, disparate services, data sources, and feeds can be quickly tied together to create new applications such as reports, information dashboards, and data exchanges between the backend systems.

SPARQLMotion is supported by all TopBraid Suite products:

- Scripts are developed and tested in [TopBraid Composer - Maestro Edition](#)
- Scripts are deployed on the [TopBraid Live](#) server platform where they can be used as web services, pro-active agents, or be invoked by applications through APIs.
- Applications assembled with TopBraid Ensemble can directly invoke SPARQLMotion scripts as [TopBraid Ensemble](#) components are SPARQLMotion-enabled.

SPARQLMotion scripts are defined in RDF using modules for importing, processing and exporting data. [TopBraid Suite](#) currently provides well over 100 extensible modules that implement a comprehensive range of data integration tasks. In addition to handling data, SPARQLMotion engine can prompt the user for input and generate user interface components such as maps and calendars, or create files such as spreadsheets.

SPARQLMotion scripts can be executed as REST Web Services. REST (Representational State Transfer) is an architectural style that allows services to be exposed and consumed over the Web using only a simple URL. SPARQLMotion modules can consume any available REST service.

In short, SPARQLMotion leverages REST technology to extend SOA to the web making it possible to use the Web as the SOA platform. Using SPARQLMotion with REST services, organizations can easily expose their data and content for use and re-use by the current and future applications.

SPARQLMotion modules are defined as classes in an ontology (such as sparqlmotionlib.rdf). Because scripts are described in RDF they benefit from RDF capabilities including ease of merging of scripts and ability to query the content of each script. End users can extend SPARQLMotion by specializing existing modules as well as adding special purpose modules.

1.1 Conventions

Class, property, module, and individual filenames are written in the Microsoft Sans Serif font like this.

Names for user interface widgets and menu options are written in the Comic Sans MS font like this.

Where exercises require information to be typed, the input is written in the Verdana font like this.

Exercises and required tutorial steps are presented like this:

Exercise N: Accomplish this

1. Do this.
2. Then do this.

3. Now do this.



Tips and suggestions for using TBC and SPARQLMotion are presented like this.



Potential pitfalls and warnings are presented like this.



General notes are presented like this.



Advanced features are presented like this. We recommend that readers skip advanced features when they first follow this guide.

1.2 Assumptions

Users of this tutorial should be familiar with Semantic Web standards and with the TopBraid Composer product. At a minimum, users should understand the RDF data model and be able to create SPARQL queries. A short “Getting started guide” for SPARQLMotion is available at http://www.topquadrant.com/products/SPARQLMotion_docs/SPARQLMotion_guide.html. It can be used in conjunction with this tutorial.

This tutorial requires TopBraid Composer Maestro Edition version 3.5 or higher.

1.3 TopBraid Composer Help Files

TopBraid Composer Help files provide additional information for all topics in this tutorial. To open Help, go to Help > Help Contents.

1.4 Completed Exercises for this Tutorial are Available

For reference, a set of completed exercises for this tutorial are provided for download at: <http://www.topquadrant.com/sparqlmotion/tutorial/SPARQLTutorial.org.zip>

2 SPARQLMotion Modules

SPARQLMotion is defined in several RDF files located in the TopBraid project, SPARQLMotion folder. They are automatically updated in your workspace when a new version of TopBraid is released.

To explore SPARQLMotion modules and their definitions, create any RDF/OWL file that imports `sparqlmotionlib.rdf`. In the **Classes** View, you will see `sm:Modules` (subclass of `spin:Modules`) with three subclasses:

- **sm:ImportModules** - Modules that import information from a data source. Several subclass modules are available for loading files, establishing connections with databases, receiving XML from web sources, taking user input, and so on.
- **sm:ProcessingModules** - Modules that process data. Subclass modules are available for controlling the flow of the script, executing SPARQL queries, running inferencing, converting between different formats, and so on.
- **sm:ExportModules** - Modules that create an output. Subclass modules are available for outputting files, writing to databases, sending e-mail, generating UI components, and so on.

When you create scripts, you will also see functions in the palette. These are subclasses of `spin:Functions` - modules that can be used as a SPARQL function (more information about function modules is provided later in this tutorial).

Because each module is defined as a class, you can explore the modules by selecting the class and looking at its definition in the **Resource Form**. The relevant properties for each class are mentioned as SPIN constraints at the class. Click on the small + button over the constraints' icons to display details about each property. The list of available modules continuously evolves. To see currently available modules, go to the Composer Help file:

Help Contents > TopBraid Composer > Reference > TopBraid SPARQLMotion Module Library

To create additional custom modules, you will need to create a subclass of one of the existing modules. This process is described in detail in the “Creating New SPARQLMotion Modules” section of this document.

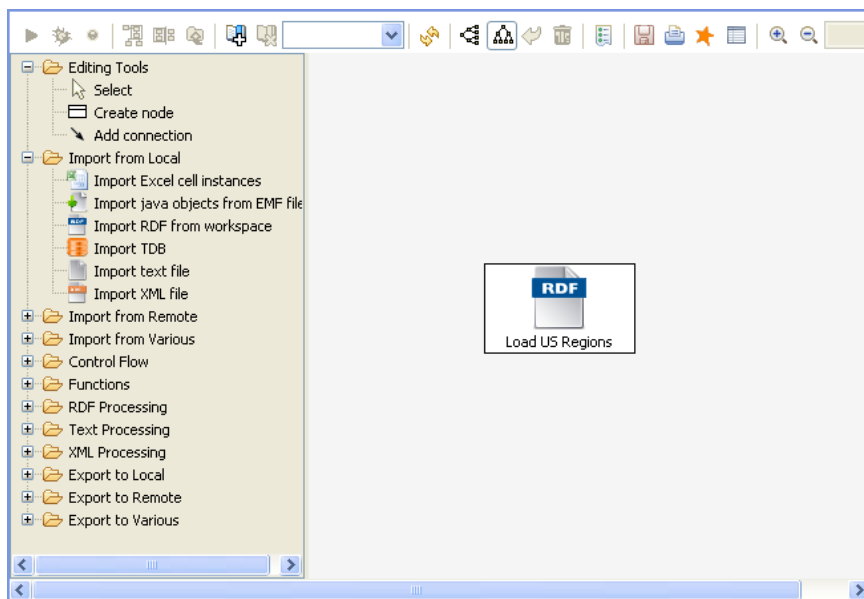
3 Creating Simple Data Processing Sequences

Automating repeatable data-manipulation tasks is a key feature of SPARQLMotion. Our first script will perform a simple data-processing task. We will:

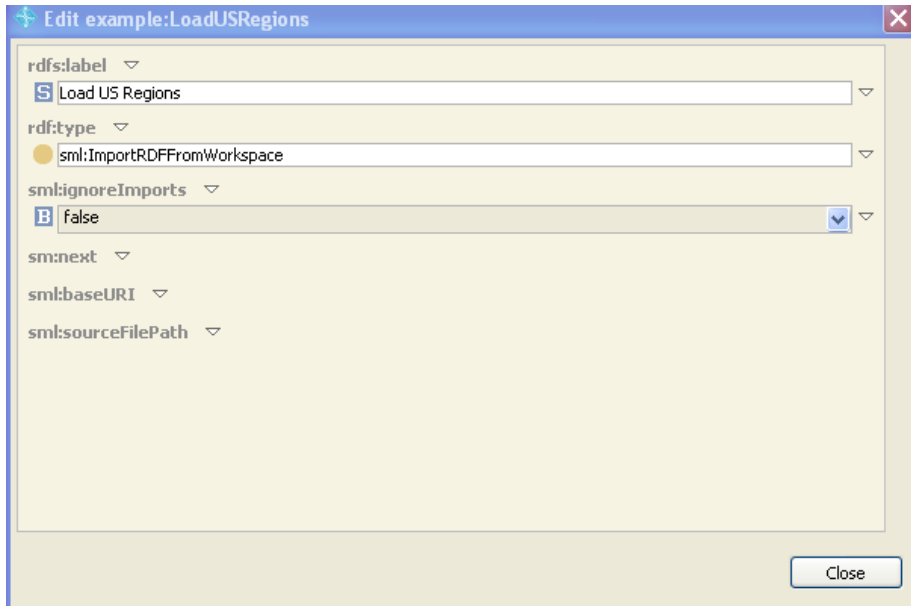
- Load two files
- Merge the files
- Save the merged result as a third file

3.1 Exercise 1

1. Create a new project:
File > New > Project...
Click **Project** (underneath the **General** folder) and click **Next**.
Enter SPARQLMotion-tutorial.com in the text field and click **Finish**. Your new project workspace will appear in the **Navigator** pane on the left of your screen.
2. Place region.owl, US-regions.owl, and US-states.owl into the SPARQL Tutorial project and open each one of them to view their contents. The files are available for download at <http://www.topquadrant.com/sparqlmotion/tutorial/ontologies.zip>.
3. Create a new file:
File > New > RDF/SPARQLMotion File.
Enter exercise1 after the last slash in the **Base URI** field (replacing “unnamed”) and click **Finish**.
This imports the SPARQLMotion models that TopBraid uses to define and manage your scripts.
4. From the **Scripts** menu, select **Create SPARQLMotion Script**. This will display the Graph view canvas with the palette of SPARQLMotion modules.
5. Expand **Import from Local** folder and drag and drop **Import RDF from workspace** module into the canvas. In the **Create** dialog that will appear name the module **LoadUSRegions**.
6. Click on the **Select** arrow (under **Editing Tools**) and double click on **Load US Regions** to see its details



As shown in the following Resource Form, you can provide either a **baseURI** or a **sourceFilePath** for the resource location. For this exercise, we'll provide a **baseURI**:



7. Click on the down-facing triangle next to **sml:baseURI** field and select **Add Empty Row**. Enter <http://www.topbraid.org/owl/geo/US-regions> in the field.
8. Now we'll bring in another OWL file directly from the web. Expand **Import from Remote** to see the available modules. Select **Import RDF from URL**, drag and drop it on the canvas. Name it **LoadCountries** and click **OK**.
9. Click on the **Select** arrow (under **Editing Tools**) and double click **Load countries** to view details. Enter <http://www.topquadrant.com/topbraid/countries> in the **url** field.

Create an instance of the **Export** module to save the loaded triples to a new local file:

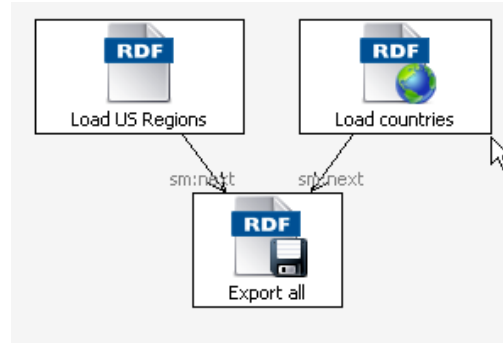
10. Click **Export to Local** in the **Palette** menu to see more modules. Drag-and-drop **Export to RDF file** onto the canvas. Name the new step **ExportAll**. Click **OK**.
11. Let's connect the modules. Click **Load US Regions** to highlight (it will turn yellow). Move the mouse and click the target (**Export all**) to set the connection line.
12. Click on the **Add connection** arrow (under **Editing Tools**) and connect **Load countries** to **Export all**.
13. Double click **Export all**. Enter the **baseURI** (for example, SPARQLMotion-tutorial.com/exercise1) and the **targetFilePath** as **exercise1-output.ttl**. This will create the file in the same project you are currently in.



To create a file in the different project include the project name in the **sml:targetFilePath** field. For example, **/projectX/myfile.n3**.

14. Click on the  **Layout vertically** icon to re-layout after the change.


Your script should now appear as follows:



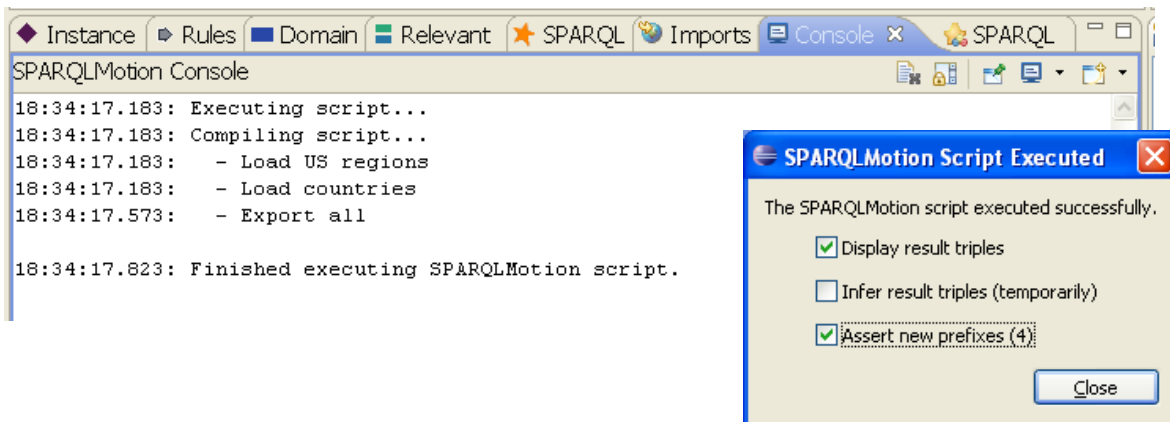
By connecting instances of modules using the next relationship, we specify that all triples created as an output of the execution of the module in the subject of the next triple will be passed as an input to the module in the object of the next triple. In this example, all triples output by **Load US Regions** as it loads the file from the workspace will be passed as input to **Export all**. All triples from **Load countries** will also be passed as input to **Export all**.



Make sure that **Add Connection** is selected before creating another connection. Use **Layout** icons to clean up the display of the script after the changes.

15. To execute the script click **Run** . When the script completes, you will be prompted for some possible actions that the workbench can take based on the new triples. Select **Display result triples**. Also select **Assert new prefixes**; we'll see why this is helpful later in the exercise.

The **Console** view displays the progress of each step as shown below. If there are any errors, they are also reported in the **Console** view (shown below).



```
SPARQLMotion Console
18:34:17.183: Executing script...
18:34:17.183: Compiling script...
18:34:17.183: - Load US regions
18:34:17.183: - Load countries
18:34:17.573: - Export all
18:34:17.823: Finished executing SPARQLMotion script.
```

SPARQLMotion Script Executed

The SPARQLMotion script executed successfully.

- Display result triples
- Infer result triples (temporarily)
- Assert new prefixes (4)

Close



Prefixes, URIs and qnames. RDF resources are represented with full URIs, such as <http://www.topbraid.org/owl/geo/US-States#AmericanSamoa>. Since full URIs are inconvenient and difficult to read, shortcuts, known as prefixes can be used. For example a prefix named 'states' may be defined for <http://www.topbraid.org/owl/geo/US-States#>. When combined with the local name, the URI can be represented as "states:AmericanSamoa". This prefix:name syntax is known as a qname. For more information, see Help > Help Contents > TopBraid Composer > User Interface Overview > Resource Editor > Ontology Overview

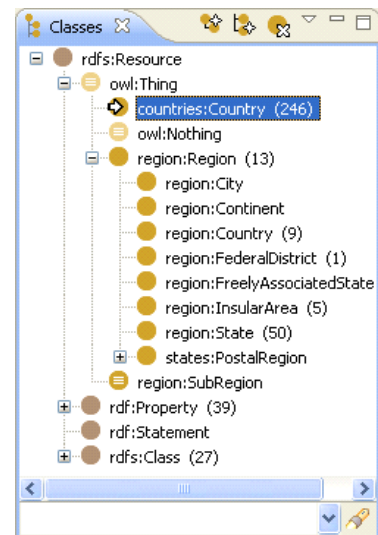
When the script is complete, the **SPARQLMotion Results** view will display all the triples produced by the final step. (In this case, the **Export all** step.) You can browse through the resulting triples

16. Scroll through the **SPARQLMotion Results** view and note that the subjects of the loaded triples use different namespaces as part of the URIs. In fact, they come from more than one ontology, because **US-regions.owl** imports both **regions.owl** and **US-states.owl**. When the file is loaded, all imports specified in the file are loaded as well.

This means that the result file will contain all triples from these four files: **countries**, **US-regions.owl**, **regions.owl**, and **US-states.owl**.

17. Open the output file (e.g. **exercise1-output.ttl**). Look at the class hierarchy. It should look like the figure shown here.

Up to this point, the information has been brought together, but has not been connected. If you look at **US-regions.owl**, you will see that it contains nine countries, all of which are also present in the **countries** ontology, but using different URIs. We would like to define the nine countries such that they refer to the same resource. There are a couple of methods we can use to integrate these models:



- We can connect the countries present in both models using a certain property – **owl:sameAs** would make sense in this case (described in Exercise 2).
- We can remove the country instances brought in by the **US-regions** model and make references directly to the corresponding instances in the **countries** model (described in Exercise 3).



Scripts such as this one are useful for moving RDF content. In this exercise, we created a new file. Alternatively, we could have moved the information into an existing RDF database. If you have access to an RDF database, you can try a variation on this script. You will need a module that establishes a connection to the database; instead of loading RDF file, "load" a database connection. When writing out, replace file export with the **Perform Update** module.

3.2 Exercise 2

In this exercise, we will connect matching countries using the **owl:sameAs** relationship.

1. Open the file `exercise1`, navigate to the namespaces page by clicking on the home button, then on the **Ontology Overview** tab and change `exercise1` to `exercise2` in the **Base URI**.
2. Save it as `exercise2`.
3. Edit the SPARQLMotion script and select **Export all**. Change the **baseURI** to `http://www.sparqlmotion-tutorial.com/exercise2`, and the **targetPath** to `exercise2-output.ttl`.
4. Drag and drop **Apply construct** (under **RDF Processing**) on to the canvas. Name the new step `ConnectCountries`.
5. Right click on the **next** arrows leading to **Export all** and click **Delete**. Redraw the script so that both loads are followed by `Connect countries` which is then followed by **Export all**.
6. Double click on `Connect countries` and enter the following query in the `constructQuery` field:

```

CONSTRUCT {?country1 owl:sameAs ?country2}
WHERE {?country1 a <http://www.topbraid.org/owl/geo/region#Country>.
?country1 <http://www.topbraid.org/owl/geo/US-states#postalAbbreviation> ?value.
?country2 a <http://topbraid.org/countries#Country>.
?country2 <http://topbraid.org/countries#abbreviation> ?value.}

```

Instead of running the entire script, we will now experiment with using the debugging features. You can select a module (single click to highlight) and then click on **Debug** icon. SPARQLMotion will execute all the modules up to and including the selected module.

7. Click **Debug** icon to run the script up to the `Connect countries` step to see what countries are being connected. Click on **Continue**, then close **SPARQLMotion Script Executed** dialog making sure that **Display result triples** is selected.

Subject	[Predicate]	Object
<http://topbraid.org/countries#ER>	owl:sameAs	<http://dbpedia.org/resource/Eritrea>
<http://topbraid.org/countries#BF>	owl:sameAs	<http://dbpedia.org/resource/Burkina_Faso>
<http://topbraid.org/countries#UM>	owl:sameAs	<http://dbpedia.org/resource/United_Microstates_of_Melanesia>
<http://topbraid.org/countries#SI>	owl:sameAs	<http://dbpedia.org/resource/Sierra_Leone>
<http://www.topbraid.org/owl/geo/US-states#MarshallIslands>	owl:sameAs	<http://topbraid.org/countries#MH>
<http://www.topbraid.org/owl/geo/US-states#USVirginIslands>	owl:sameAs	<http://topbraid.org/countries#VI>
<http://www.topbraid.org/owl/geo/US-states#AmericanSamoa>	owl:sameAs	<http://topbraid.org/countries#AS>
<http://www.topbraid.org/owl/geo/US-states#FederatedStatesOfMicronesia>	owl:sameAs	<http://topbraid.org/countries#FM>
<http://www.topbraid.org/owl/geo/US-states#PuertoRico>	owl:sameAs	<http://topbraid.org/countries#PR>
<http://www.topbraid.org/owl/geo/US-states#Guam>	owl:sameAs	<http://topbraid.org/countries#GU>
<http://www.topbraid.org/owl/geo/US-regions#USA>	owl:sameAs	<http://topbraid.org/countries#US>
<http://www.topbraid.org/owl/geo/US-states#NorthernMarianaIsland>	owl:sameAs	<http://topbraid.org/countries#MP>
<http://www.topbraid.org/owl/geo/US-states#Palau>	owl:sameAs	<http://topbraid.org/countries#PW>

8. Navigate to the **SPARQLMotion Results** view to examine the triples. Click the **[Predicate]** column header to sort by predicate and scroll down to `owl:sameAs`. You will see nine triples connecting resources from the two models that we are loading:

If the nine triples do not appear, it may be because Composer limits the maximum number of instances that can be displayed in any of the views. The default 1000, but can be customized through the TopBraid

Composer Preferences. Open Preferences and change the **Max. number of instances to display** to 2000 or greater.

If we run the script to the end (choose **Export all** and click **Run** a file will be created that contains the loaded triples as well as nine constructed triples.

Using the replace property

We may also want to preserve constructed triples in a separate file as a bridge model between **countries** and **US-regions**.

9. Double click on the **Connect countries** step and change the **replace** field entry in its form to **true** and rerun the script. You will see only nine triples in the **SPARQLMotion Results** view, which can now be saved in a separate file.

replace property specifies whether a module will overwrite all triples from its predecessors. If this entry is set to **true** (the default entry is **false**), then the triples from the predecessors shall be overwritten.

10. Create another instance of **Export to RDF file** module and name it **ExportCountryMapping**.

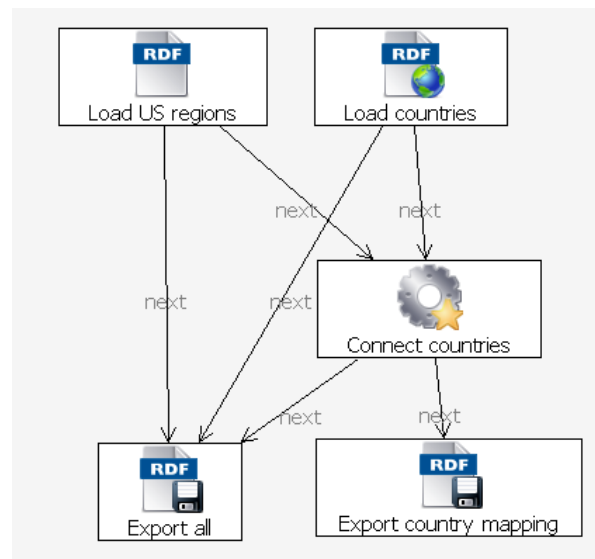
11. Change the **baseURI** to <http://www.sparqlmotion-tutorial.com/exercise2-map> and the **targetPath** to **exercise2-mapping.ttl**.

12. Add another **next** link from **Connect countries** to **Export country mapping**.

13. Add **next** links from **Load US Regions** and **Load countries** to **Export all**. Your script will look similar to the figure on the right.

14. Select **Export country mapping** and click on **Run** icon.

15. Examine the results in **exercise2-mapping.ttl**. Note that when you open this file, Composer displays a warning that there are untyped references to resources. Choose “No” and view the contents of the file through the **Triples View (Window > Show View > Triples)**. Click to select the graph in Triples view, then sort on the predicate column to find nine **owl:sameAs** triples constructed in **Connect countries**.



Select **Scripts -> Execute SPARQLMotion modules...** in the main menu. A dialog will appear showing both **ExportAll** and **ExportCountryMappings** selected. A SPARQLMotion script is identified as an end module (a module that is not part of an iteration loop and is not followed by another module) and all the modules connected to it transitively. Since we have two end modules, TopBraid recognizes two scripts.

Thus far we have been exploring connections between the two definitions of Countries in the **US regions** and **countries** ontologies. We declared that the common country definitions, as determined by common **postalAbbreviation/abbreviation** properties, are the same. This is one way to connect ontologies. The next exercise explores a different way to connect the two ontologies.

3.3 Exercise 3

In this exercise, we will connect US regions directly to countries (the ontology from the “Load countries” module – i.e. <http://www.topbraid.org/countries>) by changing the data so that all of the references to countries refer to the **countries** ontology. This involves creating new triples that use the **countries** definitions and removing existing triples that use the **US regions** definitions.

16. Open the file **exercise2** and save it as **exercise3**. Change the ontology **Base URI** and output files **baseURI** and **targetPath** entries as done in previous exercises.
17. Delete **Export country mapping**; we will produce only one file in this exercise.
18. Double click on **Connect countries** and replace the **constructQuery** entry with the following. This query finds `<?country1>-subject-object` triples from common country resources in the two models and makes new triples that use the **countries** definition (`?country2`).

```
CONSTRUCT {?country2 ?p ?anyobject.}
WHERE {
  ?country1 a <http://www.topbraid.org/owl/geo/region#Country>.
  ?country1 <http://www.topbraid.org/owl/geo/US-states#postalAbbreviation> ?value.
  ?country2 a <http://topbraid.org/countries#Country>.
  ?country2 <http://topbraid.org/countries#abbreviation> ?value.
  ?country1 ?p ?anyobject.}
```

Add a second query to **Connect countries** by creating an empty row for **constructQuery** and inserting the following:

```
CONSTRUCT {?anysubject ?p ?country2.}
WHERE {
  ?country1 a <http://www.topbraid.org/owl/geo/region#Country>.
  ?country1 <http://www.topbraid.org/owl/geo/US-states#postalAbbreviation> ?value.
  ?country2 a <http://topbraid.org/countries#Country>.
  ?country2 <http://topbraid.org/countries#abbreviation> ?value.
  ?anysubject ?p ?country1.}
```

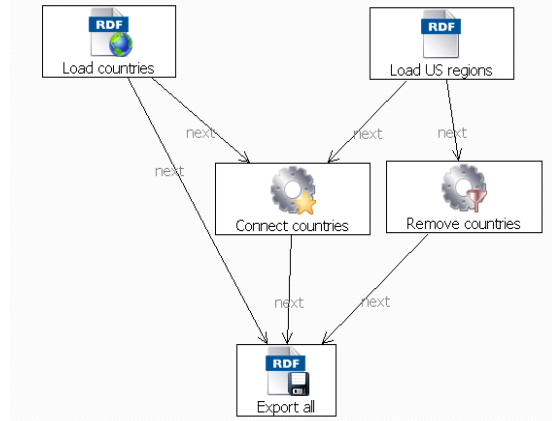
In these queries, we are copying all the triples from the **US-regions** model that have one of the countries as either an object or a subject in order to create the same triples for the corresponding resources in the **countries** model.

19. Run the script up to **Connect countries**. Check **Infer result triples** in the **SPARQLMotion Script Executed** dialog and view SPARQLMotion results. The intermediate results can also be viewed by running a query in the Debugger as it stops at the selected step. There should be 18 instances of **region:Country**. Contrast this to the results of exercise 2 where there were nine instances of **region:Country**. Note that in this exercise the script copied all triples referring to (`?anyobject`) and referenced by (`?anysubject`) the country definitions in the **US-regions** model. Hence entries from each model are found in the output file.

20. Next, we need to remove countries from the US-regions model and any references to them. Drag and drop **Filter by construct** class (under **RDF Processing**). Name the new step **RemoveCountries**.

This module runs the **CONSTRUCT** queries, but instead of adding constructed triples to the output, it removes them from the input. Therefore the output of this module consists of all incoming triples except the triples specified in the **CONSTRUCT** queries.

21. Insert the new module between **Load US regions** and **Export all**. Your script will be similar to the one shown here.



22. Enter the following **CONSTRUCT** queries in the **Remove countries constructQuery** fields:

```
CONSTRUCT {?country ?p ?anyobject.}
WHERE {
?country a <http://www.topbraid.org/owl/geo/region#Country>.
?country ?p ?anyobject.}
```

```
CONSTRUCT {?anysubject ?p ?country.}
WHERE {
?country a <http://www.topbraid.org/owl/geo/region#Country>.
?anysubject ?p ?country.}
```

23. Run the script and examine the results by opening the file created (e.g. **exercise3-output.ttl**).



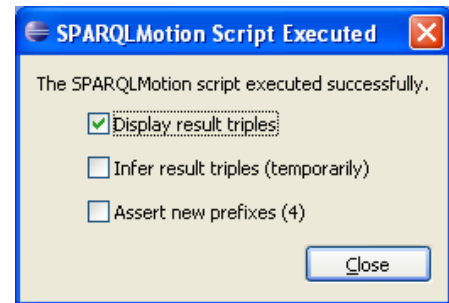
If you navigate away from your script, use **Scripts > Edit/View SPARQLMotion script** to return to the display canvas. Alternatively, navigate to the end module of a script and switch to the **Graph** view.

4 Iterative Development of SPARQLMotion Scripts

At this point in the tutorial, it becomes important to discuss a development methodology for SPARQLMotion. Many modules rely on and require SPARQL queries. Thus, the process of developing scripts is closely connected to the process for developing and testing queries. In order to test the query that will work in a particular step of the script, it must be run against the set of triples that will be passed to that step.

To assist with the development process, TBC-ME provides a convenient feature. You have already noticed the dialog that comes up at the end of each “debug mode” run. See example figure on the right.

If the triples produced by that point are inferred, they become available for testing the queries needed for the subsequent steps. Asserting prefixes eliminates the need to use full URIs in the queries. This means that we can use `region:Country` in the subsequent queries instead of `http://www.topbraid.org/owl/geo/region#Country` since regions file defines the prefix which we can then assert in the script.



The process for developing scripts can be described as follows:

1. Define the first module (this will often be import module or other module not relying on queries).
2. Select the module and click **Debug**.
3. Assert prefixes and infer triples.
4. Run test queries as needed for the next step and browse results.
5. Define the next module using the query(s) that have been tested.
6. Reset inferences.
7. Return to step 2 to continuously develop the script.

We will demonstrate this process in the next exercise.

The exercise will also show how to work with some of the processing modules. In the exercises 2 and 3, we became familiar with two of the RDF Processing modules – **ApplyConstruct** and **FilterByConstruct**. In addition to the RDF Processing modules, SPARQLMotion currently also offers modules for:

- Controlling the flow of the script
- Processing text
- Processing XML

We will demonstrate some of these modules in the exercises that follow.

4.1 Exercise 4

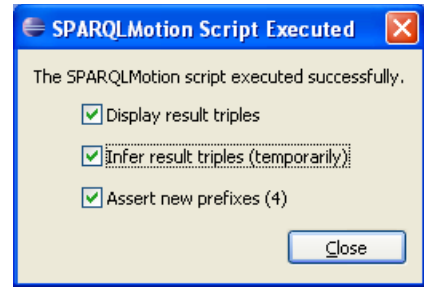
In this exercise we will experiment with the control flow modules. We will iterate over all countries, and for each country get its capital from DBPedia, an RDF store of data gathered from Wikipedia pages. We will then repurpose the DBPedia IDs (URIs) of the capitals to create our own local IDs and connect them to the countries they are capitals of using `region:capital` property. Finally, we will save the result into a file.

1. Create a new SPARQLMotion file - exercise4 file by using **New > RDF/SPARQLMotion file**.
2. Click on **Scripts > Create SPARQLMotion Script** and then create a Load countries module identical to the one created for Exercise 1.
 - Use Import RDF from URL

- url:
http://www.topquadrant.com/topbraid/countries

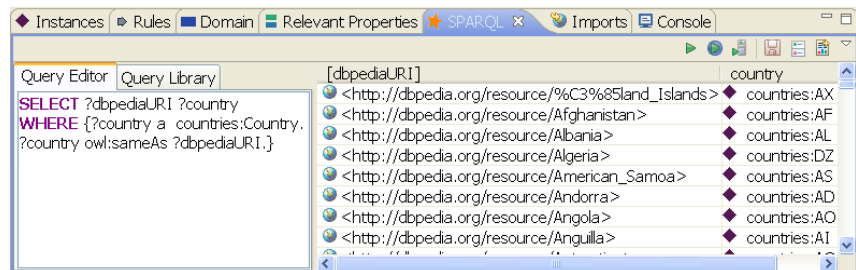
3. Click **Debug** to run the script. Select all check boxes as shown in the dialog and click **Close**.
4. Look in the **Classes View** and note that it now shows class **countries:Country** with 246 instances. Note that this class and instances have been *temporarily* asserted to the script's ontology. These triples will not be saved and will change the next time the script is executed.
5. In the next step of the exercise, we want to get DBPedia URIs for each country and then retrieve from DBPedia the name of the capital. Enter the following query in the **SPARQL View** to test it:

```
SELECT ?dbpediaURI ?country
WHERE {?country a countries:Country.
?country owl:sameAs ?dbpediaURI.}
```



Up until now we have been using full URIs in the queries surrounded by the angle brackets. We can use qnames instead, but to do so, we will need to either import an ontology that declares prefixes for the namespaces or declare the prefixes within the script. Because we have asserted the prefix defined in the countries ontology in step 3, we can now use `country:<local resource name>` in the queries.

The result is an array of `dbpediaURI`, our local country URI for each country.




6. Drag and drop **Iterate over select** on the Graph canvas (under **Control Flow**), name the step **SelectDBPediaURI**. Connect **Load countries** to **Select DBPedia URI** via the next property. This module will repeatedly execute a sub-script for each matching result set of a SPARQL Select query. The start of the sub-script is specified by the **body** property. The **Iterate over select** modules create no new triples, so input to the sub-script is the same as the input to the **Iterate over select** module. In this case, it is all the triples from the country model.
7. Add the query from the **SPARQL View** in the previous step to the `selectQuery` field of `Select dbpedia URI`. For each result pair, the steps pointed to by the `body` field will be executed. In each of those body executions, the variable `?dbpediaURI` will be bound to a value from query used in `Select DBPedia URI`. As a result, it can be used inside SPARQL queries, etc. We will need to take two actions: one to get the URI and a label of capital for each country from the DBPedia and another to prepare this information for saving to the results file. The reason we take two steps is because we want to do some post-processing of the URIs to create our own (non-DBPedia) IDs for the capitals.

Note that variable bindings, such as `?dbpediaURI` and `?country` are passed from one module to the next. However, since **Iterate over select** modules repeat the sub-script for each result set found, the variable bindings in the sub-script can only be used within that sub-script, and not outside of it.



Typically, you can perform most of the processing in a single query and iterating over select is not necessary. However, it becomes useful if, as in this case, you need to take results from one source and use them to get information from another remote source such as a web service. **Iterate over select** modules have `sml:maxThreadCount` field. It is used to allow several iterations to run in parallel. This option can significantly improve performance of the script if a step in the iteration needs to interact with (and wait for response from) remote services. At the end of this exercise, you could optionally set `sml:maxThreadCount` field to 10, re-run the script and see how its performance improves.

8. Drag and drop **Apply construct** on to the graph canvas. Call the new step **GetCapitals**. Connect **Select dbpedia URI** to **Get capitals** using the **body** predicate.
9. Click the **Home**  icon to get to the Ontology home page, then click on the **Overview** tab. Add the **region** prefix for `http://www.topbraid.org/owl/geo/region#` namespace. We can now use `region:<local resource name>` in the queries.
10. Add the following query to the `constructQuery` field of **Get capitals**:

```
CONSTRUCT {
  ?capital a region:Capital .
  ?capital rdfs:label ?name.}
WHERE {
  SERVICE <http://dbpedia.org/sparql> {
    ?dbpediaURI <http://dbpedia.org/ontology/capital> ?capital.
    ?capital rdfs:label ?name.} }
```

This will call the DBPedia SPARQL endpoint (a remote service for DBPedia RDF resources) for each country, find capitals, and return a graph declaring that the matching capital is of type `region:Capital` and it has a label which value is stored in the `?name` variable.

11. Set the **replace** field of **Get capitals** to true so that the temporary triples from DBPedia SPARQL endpoint are not passed to the next step.

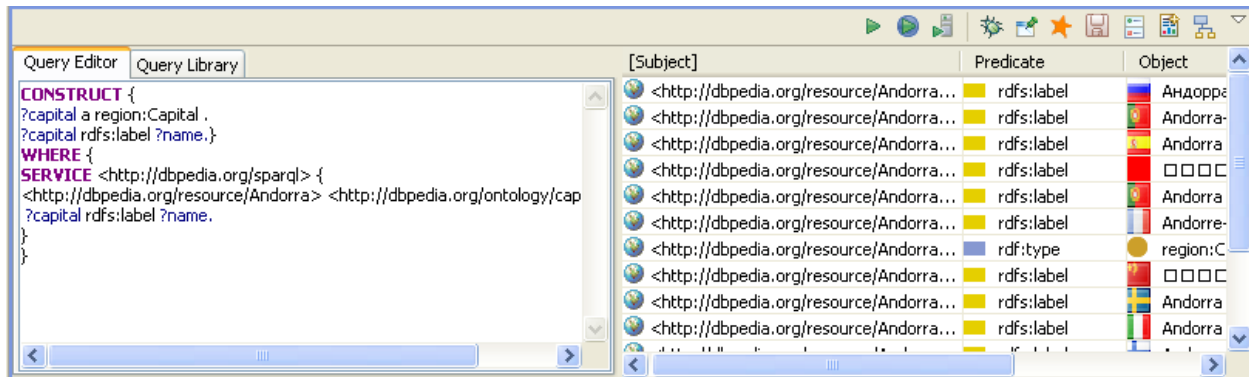
Executing test queries

Following the iterative development methodology, create some test queries using the SPARQL View before adding them to the SPARQLMotion Script.

12. Pick one of the URIs returned from DBPedia when you tested the query in step 5 and create a query to find its label and type information from the DBPedia SPARQL endpoint. For example, `http://dbpedia.org/resource/Andorra`. Use it to test the query, e.g.:


```
CONSTRUCT {
  ?capital a region:Capital .
  ?capital rdfs:label ?name.}
WHERE {
  SERVICE <http://dbpedia.org/sparql> {
    <http://dbpedia.org/resource/Andorra> <http://dbpedia.org/ontology/capital> ?capital.
    ?capital rdfs:label ?name.} }
```

Run the query in the **SPARQL** view and observe the results, which will appear as shown.



When we iteratively execute the query for each country, the URI from DBPedia for each country capital will be bound to the variable `?capital`.

We want to create our own URIs for the capitals. We will build the URIs using the DBPedia URIs.

- In the results part of the **SPARQL** view, select all the new triples created by the previous query using **Ctrl + A** or **Shift + mouse-click**, then click on the  menu icon. Choose **Infer selected constructed triples**.

This temporarily adds the triples from the SPARQL results to the script model. Given that we have also temporarily added triples from the countries ontology, we can test the SPARQL query on data that will be available when running the SPARQLMotion script we are designing.

- Enter the following query in the **SPARQL** view:

```

CONSTRUCT {
?uri a region:City .
?uri rdfs:label ?name .
<http://dbpedia.org/resource/Andorra> region:capital ?uri .
}
WHERE {
<http://dbpedia.org/resource/Andorra_la_Vella> a region:Capital .
BIND (afn:localname(<http://dbpedia.org/resource/Andorra_la_Vella>) AS ?localname) .
BIND (smf:buildURI("http://www.topbraid.org/owl/geo/region#{"?localname}") AS ?uri) .
<http://dbpedia.org/resource/Andorra_la_Vella> rdfs:label ?name .}

```

To create URIs for the capitals that use the `region` namespace, and not the DBPedia namespace, we will use `afn:localname` and `smf:buildURI` property functions.



TopBraid supports all functions implemented by Jena, plus many additional ones. These are described in the [Help > Help Contents > TopBraid Composer > Reference > SPARQL Functions Reference](#), but the best way to see the up to date listing of function is by either looking under `spin:Functions` or by using auto-complete directly in the **SPARQL** view. Type the prefix for one of the function namespaces, for example, `smf`: Then use **CTRL-space**. You will get the auto-complete list. You can navigate through the list and click on each function to see its description.

The `afn:localname` function takes as an input URI of a capital from the DBPedia and returns the part after the “#”. We are calling this variable `?localname`.

The `smf:buildURI` function creates a URI using a specified pattern – in this example we are generating a new URI for the capitals found in DBPedia.

Run the query and observe the results, which will appear as shown.

Apply test queries to SPARQLMotion script

15. Create another **Apply construct** module and name it **MakeLocalCapitals**. Connect **Get capitals** to **Make local capitals** using **next**.

16. Enter the generalization of the query we have just tested in the **construct**

Query field, replacing

`http://dbpedia.org/resource/Andorra` with `?country` and

`http://dbpedia.org/resource/Andorra_la_Vella` with `?capital`:

```

CONSTRUCT {
  ?uri a region:City .
  ?uri rdfs:label ?name .
  ?country region:capital ?uri .
}
WHERE {
  ?capital a region:Capital .
  BIND (afn:localname(?capital) AS ?localname) .
  BIND (smf:buildURI("http://www.topbraid.org/owl/geo/region#{?localname}") AS ?uri) .
  ?capital rdfs:label ?name .}
  
```

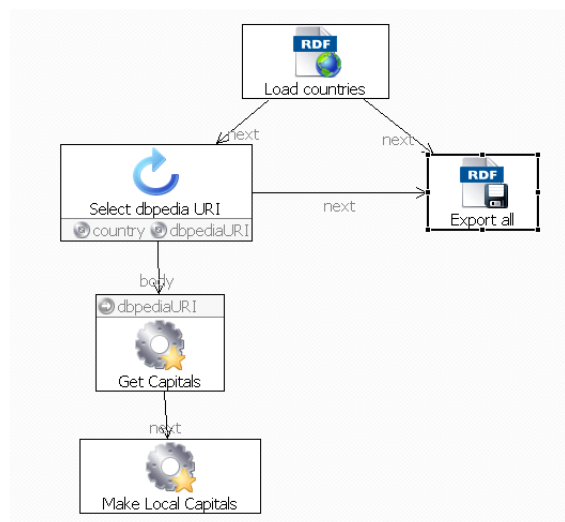
This generates a new URI for each of the capitals found in DBPedia. **Select dbpedia URI** queries the `countries.owl` model for all instances of `countries:Country` and binds `?country` and `?dbpediaURI` for each match found (step 5). For each match, the subscript of the body predicate is executed. First, **Get Capitals** finds the countries bound to `?dbpediaURI` and binds the variable `?capital` to the DBPedia property “capital” (step 10). Second, **Make local capitals** creates a new URI for each `?capital` and adds triples for a label, type, and a `region:capital` property (step 16).

17. After iterating through creating all of the capitals, the script will export the resulting file. Create an **Export to RDF file** step and name it **ExportAll**.

Change the entry in the **baseURI** field to `http://www.sparqlmotion-tutorial.com/exercise4`, and change the **targetPath** field to `exercise4-output.ttl`.

18. Connect **Select dbpedia URI** and **Load countries** to **Export all** using **next**. Your script should now appear as shown on the right.

[Subject]	Predicate	Object
<http://dbpedia.org/resource/Andorra>	region:capital	region:Andorra_la_Vella
region:Andorra_la_Vella	rdfs:label	Andorra la Vella
region:Andorra_la_Vella	rdfs:label	Andorra la Vella
region:Andorra_la_Vella	rdfs:label	Andorra la Vella
region:Andorra_la_Vella	rdfs:label	Andorra la Vella
region:Andorra_la_Vella	rdfs:label	Andorra la Vella
region:Andorra_la_Vella	rdfs:label	Andorre-la-Vieille
region:Andorra_la_Vella	rdfs:label	Andorra-a-Velha
region:Andorra_la_Vella	rdfs:label	Andorra la Vieja
region:Andorra_la_Vella	rdfs:label	Andorra la Vella (paróquia)
region:Andorra_la_Vella	rdfs:label	Andorra la Vella
region:Andorra_la_Vella	rdfs:label	□□□□□□□□□□
region:Andorra_la_Vella	rdfs:label	□□□□
region:Andorra_la_Vella	rdf:type	region:City
region:Andorra_la_Vella	rdfs:label	Андорра-ла-Велья



19. Run the script. Note that the script will call the body subscript (**Get Capitals** followed by **Make Local Capitals**) for each match in the query in **Select dbpedia URI**.

```
<http://topbraid.org/countries>
  a      owl:Ontology ;
  sm:source <http://topbraid.org/examples/countries/countriesGenerator.sms.n3> ;
  owl:imports <http://www.topbraid.org/owl/geo/US-regions> ;
  owl:versionInfo "Created with TopBraid Composer"^^xsd:string .
```



The result variables of the **Select** query in the **Iterate over select** modules will be bound in each iteration of the sub-script loop. In our example, the sub-script has two steps. Once the iterations are completed, control and triples resulting from the iterations will pass to the module pointed to by the **next** predicate. In our example, it is **Export all**.

The results of the **Iterate over select** query are the accumulated triples from the end nodes of the body steps. In our example, these are the triples produced by **Make local capitals** because we are filtering out everything that came in to it with **replace = "true"**. Since the desired result is to add the capital data to the countries model, we will need both sets of triples – those we generated (output of **Select dbpedia URI**) and those that come from the **countries.owl** model (**Load countries**). Therefore we merge these triples by connecting both modules of **Export all** as shown.

Imports for the output model

The ontology created by the script uses the **region** namespace to define various classes and properties used in the output file our script creates. Therefore, the region model's definitions need to be included in the output file, either through an import statement to import it or to physically include the triples in the output file. In this exercise, we will **import** the **region** model. But before we construct the import statement, we need to do some additional housekeeping.

Most ontologies include ontology metadata as shown here (in an N3 format). When reading the model into a SPARQLMotion script and subsequently merging the triples the ontology metadata will be copied, but should be changed to become ontology metadata for the ontology being created, otherwise there will be two **owl:Ontology** declarations.

In our example, the ontology metadata shown is copied into the SPARQLMotion script by **Load countries**. These triples are then passed to **Export all**. We want to remove the ontology metadata statements from **Load countries** and replace it with ontology metadata for the output model we created. This information gets copied into our output ontology from **Load countries**, but we will want to replace this with the ontology definition for the output file.

20. Create a **Filter by construct** module (under **RDF processing**) and name it **RemoveOntologyStatements**. Insert it after **Select dbedia URI** and **Load countries**, but before **Export all**. **FilterByConstruct** will define a set of triples through a **CONSTRUCT** statement and use that to filter matching triples. Create the following construct query:

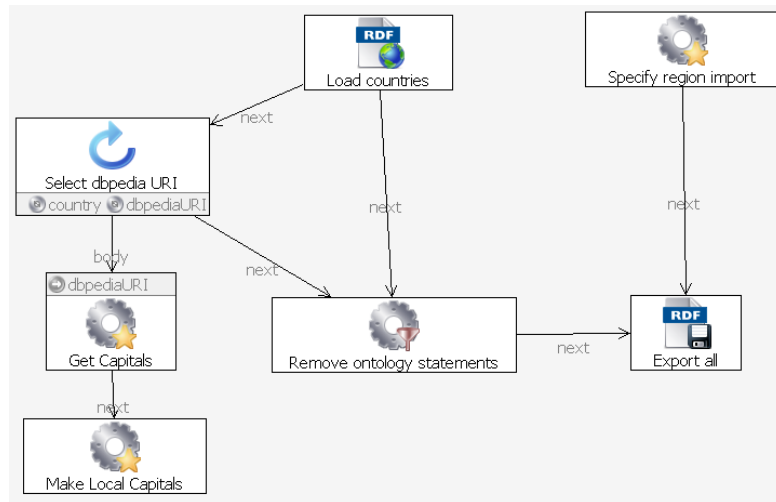
```
CONSTRUCT {
  ?o rdf:type owl:Ontology.
  ?o ?p ?s.}
WHERE {
  ?o rdf:type owl:Ontology.
  ?o ?p ?s.}
```

21. Let's add an import statement. Drag and drop **Apply construct** module and name the new step **SpecifyRegionImport**, and connect it using **next** to **Export all**. Create the following construct query (make sure that the 'exercise4' URI used here – the subject of the owl:imports statement – exactly matches the **base URI** property used in **Export all**) :

```
CONSTRUCT {<http://www.sparqlmotion-tutorial.com/exercise4> owl:imports
<http://www.topbraid.org/owl/geo/region>}
WHERE {}
```

The constructed triple states that the output file will import the contents of region.owl, as specified by its base URI < http://www.topbraid.org/owl/geo/region >. Note that we did not add the **rdf:type owl:Ontology** triple. It will be created automatically. The resulting script should look like as shown.


22. Reset the inferences and run the script.
23. Open **exercise4-output.ttl**. Note that we now have 177 cities (instances of City) connected to the countries that they are capitals of.

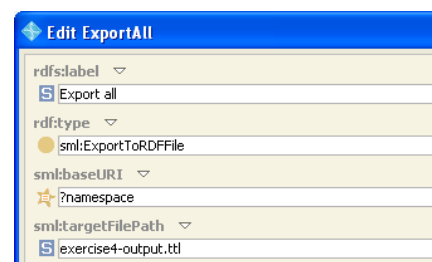


Variable bindings and string templates in SPARQLMotion scripts

The previous script depended on the script having the same value for the output base URI (**Export all**) and the constructed import statement (**Specify region import**). It would be desirable to have this value entered only once. We can accomplish this in a couple of ways. In this exercise, to demonstrate variable bindings, we will use **Bind literal variable** module. In the next exercise we will use a generalization of arguments that allows variable bindings to be passed to the script by user's input or a Web service call.

24. Drag and drop **Bind literal variable** (under **Control Flow**) and connect it to **Specify region import**. Edit the outputVariable to the string "namespace" and the template to the string "http://www.sparqlmotion-tutorial.com/exercise4".

In **Export all** change the baseURI from a string to a reference to ?namespace. Near **sml:baseURI** click on , select "Add SPARQL expression" and use "?namespace" in the field as shown:



25. **Bind literal variable** modules bind a string value to a variable. This will work OK for **Export all**, but when using the value for the import statement, the string will need to be turned into a URI. `smf:buildURI` turns a template into a URI. Change the query in **Specify region import**, to the following:

```
CONSTRUCT {  
  ?namespaceURI owl:imports <http://www.topbraid.org/owl/geo/region> .  
}  
WHERE {  
  BIND (smf:buildURI(?namespace) AS ?namespaceURI) .}
```

26. Change the name for the target file as needed and run the script.

4.2 Exercise 5

After observing the results of the exercise 4, we notice that only about 70% of the countries were matched with their capital cities. It would be helpful to create a report of countries for which we did not find a capital in DBPedia. We will create a tab delimited text file identifying those countries that do not have a connected capital city. We will not create a new script, but continue to work with and extend exercise 4 script.

1. Drag and drop **Create spreadsheet** module (under **Text processing**), call the new step **CountriesWithoutCapital**. Connect it to follow the **Export all** step.
This module creates a tab-separated spreadsheet file, where spreadsheet data is populated from the variables selected by the **SPARQL SELECT** query. We only need one column containing the names of countries that don't have a capital.
2. Enter the following **SELECT** query to create the spreadsheet text:

```
SELECT ?countryName  
WHERE {  
  ?country a <http://topbraid.org/countries#Country>.  
  ?country rdfs:label ?countryName.  
  NOT EXISTS {?country region:capital ?capital}  
}
```
3. The text generated by the **Countries without capital** step is bound to a variable specified in the **outputVariable** field. By default it is called **text** and is inserted automatically. We can now save the text to a file or further process it. To save the text, drag and drop on the graph canvas **Export to text file** module (under **Export to Local**) to create a new step and name it **ExportSpreadsheet**.
4. Make this module the final step of the script. Set its **targetFilePath** field to **countries-without-capitals.txt**. Optionally, set **replace** flag to **true**, if you want to run the script more than once, to overwrite the target file.
5. Run the script and look at the **countries-without-capitals.txt** file.

5 SPARQLMotion and Web Services

The previous exercises used Composer as the interface for controlling SPARQLMotion scripts. In many cases there is a need to run scripts programmatically. SPARQLMotion script can be created and executed as a REST Web Service. Scripts that are web services have to include an “exit point” – a module that can be referenced from the outside by its URI or local name. Several exit point modules are provided under **Control Flow**, including **Return RDF**, **Return XML**, and **Return text**.

To execute the Web Service, invoke the URL with the following format:

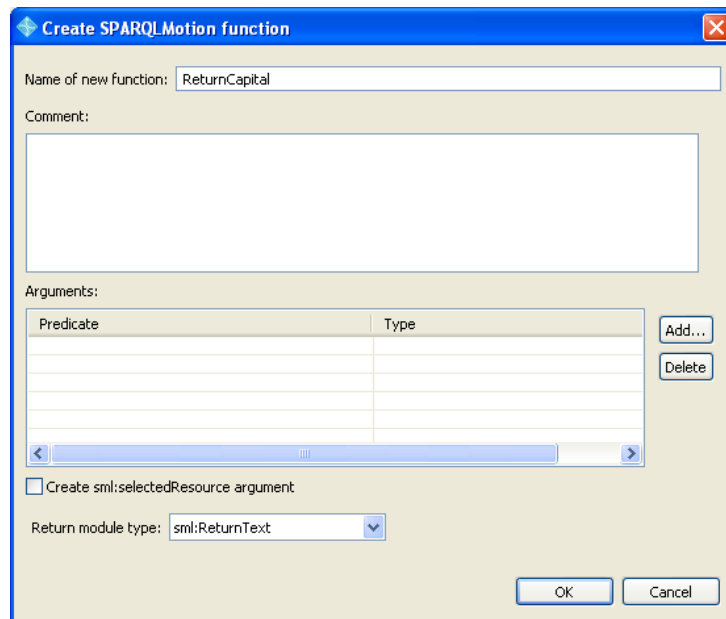
```
http://<server-name>/tbl/actions?action=sparqlmotion&id=<function-name>&<argument-name=argument value>
```

Argument-name is optional and intended for web services that require input variable(s). There can be multiple arguments. When tested using a local installation of TBC Maestro, enter localhost:8083 for the <server-name>.

5.1 Exercise 6

We will create a simple web service that takes a name of the country as an input and returns its capital.

1. Create a new script for exercise 6 by following the steps outline in exercise 1. In the **Create SPARQLMotion File** dialog box, choose the “Script will declare (Web) Services or Functions (.sml extension)” and “Set a default namespace in the new file”.
2. Create a property to be used for the argument. In the **Properties** view, find the property `sp:arg1` (subproperty of `sp:arg`). We can use this generic property or for better understandability, we can create a custom property. Let’s create one as a subproperty of `arg1` and call it `countryName`.
3. Select from the **Scripts** menu **Create SPARQLMotion function ...** You will get a dialog. Call the new function `ReturnCapital`. Select Return module type as `sml:ReturnText`, then click on the **Add** button to specify arguments.
4. In the **Add Argument** dialog, use `countryName` as the **predicate** and set **valueType** to `xsd:string` as shown below. Press **OK**. And press **OK again** in the **Create SPARQLMotion function** dialog.



Add Argument

spl:Argument

comment: a comment describing the argument (xsd:string optional)

hidden: Indicates whether this is a "hidden" argument. Hidden arguments will not be presented to the user in input dialogs but instead always have their default value. (xsd:boolean optional)

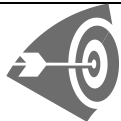
optional: indicates whether the argument is optional (xsd:boolean optional)

predicate: the property holding the values of each function call

valueType: the value type of the argument (rdfs:Class optional)

defaultValue: the default value for the argument (optional)

- You should now see the SPARQLMotion graph canvas with the argument and return module pre-populated. Use **Import RDF from workspace** module to create a step to load the output of exercise 4 file that has both countries and capitals.. Name this step LoadCountriesCapitals.



If you want to avoid having to type in file names when you use **Import RDF from workspace** module, you can instead simply drag and drop a file from your workspace into the graph canvas.

- Drag and drop **Bind by select** (under **Control Flow**) to create a new step and name it GetCapital. Connect Load countries capitals to Get capital. Connect countryName argument to Get capital.
- Enter the following query in Get capital module:

```

SELECT ?capitalLabel
WHERE {
  BIND (smf:setLanguage(?countryName, "en") AS ?countryLabel) .
  ?country rdfs:label ?countryLabel .
  ?country <http://www.topbraid.org/owl/geo/region#capital> ?capital .
  ?capital rdfs:label ?cap .
  FILTER (lang(?cap) = "en") .
  BIND (smf:setLanguage(?cap, "") AS ?capitalLabel) . }

```

This query uses the bound variable ?countryName passed to the service as an argument to find the capital and its label.

We are using smf:setLanguage function to take the input value for the country and create a new string with English identified as the language.

We are also using lang function, one of SPARQL's standard built-in operators. See <http://www.w3.org/TR/rdf-sparql-query/#func-lang> for more details. We have retrieved capital labels in different languages from DBpedia and saved them in the exercise 4 output. We are using lang function in the FILTER to retrieve the capital label in English.

Finally, we used smf:setLanguage to delete the language identifier because we wanted to simply return the string with the name of the capital. We bound the string to the text variable.

8. Drag and drop **Bind literal variable** to create a new step and name it **CreateResponse**. Connect **Get capital** to **Create response**.

9. Enter the following in the **sm:template** field of **Create response**:

{?capitalLabel} is the capital of {?countryName}

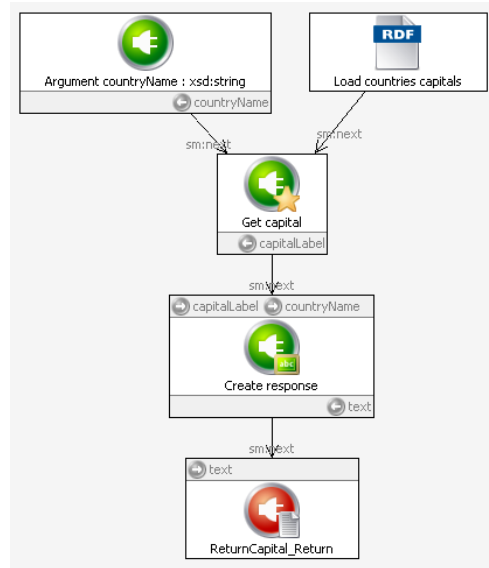
Make sure **sm:outputVariable** has the value: **text**

10. Connect **Create response** to **ReturnCapital_Return**. Your script should look as shown in the diagram.

11. To register the service select **Scripts > Refresh/Display SPARQLMotion functions...** from the menu. This will display all functions currently registered in the workspace in the **Console** view. You should do this each time you modify a service.

12. Test the script using **Run** function and choosing **ReturnCapital_script**. This will prompt the user for a test value for **countryName**. Enter, for example **"Germany"** and **OK**.

13. After the script finishes executing, look at the **Console** view. The result variables will be displayed, including the **"text"** variable in the **ReturnCapital_script** module. For example it will read **"Berlin is the capital of Germany"** per the template we defined in **Create response**.



14. To test the script as a web service, open a browser and enter the following URL:

<http://localhost:8083/tbl/actions?action=sparqlmotion&id=ReturnCapital&countryName=Germany>

TBC-ME's internal localhost server will be called. Result will be displayed in the browser. Note that the query string uses **"action=sparqlmotion"** to call any registered SPARQLMotion script and **"id=ReturnCapital"** to access the SPARQLMotion function we connected to our script.



In addition to being a web service, a SPARQLMotion script can call external REST web services. Calling external web services is done using one of the **imports** from the URL modules, such as **Import RDF from URL** or **Import XML from URL**. Since many web services return XML, **Import XML from URL** will be the most typically used module. It gets XML from a given URL and binds the returned XML document to a specified output variable. For services returning JSON, use **Import XML from URL module**.

6 Extending SPARQLMotion

One of the strengths of SPARQLMotion is that it is based on a highly extensible framework. This means that SPARQLMotion scripts can be enriched with new functionality not built into the default environment. The SPARQLMotion execution engine embedded in the TopBraid Suite can be extended in numerous ways:

1. SPARQLMotion is a model-driven language: the language and the module libraries are themselves defined in OWL ontologies. This, for example, makes it possible to define your own module types by just sub-classing the existing library classes. This works well in those cases where behavior can be derived from variations of existing implementations.
2. SPARQLMotion can be used to define new SPARQL BIND/LET/FILTER functions without Java programming.
3. SPARQLMotion is using the Jena SPARQL engine (ARQ) to execute SPARQL queries. Many SPARQLMotion modules are actually driven by SPARQL queries. Extensions to the SPARQL engine will be available to SPARQLMotion modules at run time. TopBraid includes various such extensions, such as the "tops" property functions (aka magic properties) and the BIND/LET/FILTER functions from the SPARQLMotion Functions Library. Java programmers can add other functions using the Jena API.
4. SPARQLMotion's execution engine API has a plug-in mechanism that enables Java programmers to add custom implementations.

Customization requiring Java programming is outside of the scope of this tutorial.

6.1 Exercise 7

This exercise will demonstrate a simple way to extend SPARQLMotion by creating subclasses of existing modules.

In exercise 1, we loaded triples from a local (workspace) file as well as triples from a web location. We used an existing module `sml:ImportRDFFromURL`. When we created instance of this module, we had to specify a URL. It may be advantageous to create a module which instances will automatically “know” the URL to load RDF from.

This exercise will show how to create such custom modules.

1. Start by cloning the `exercise1` script (by following the process outlined in the beginning of exercise2) to create the `exercise7` script.
2. Create a subclass of `sml:ImportRDFFromURL`, and name it `ImportCountries`.



Constructor properties can be attached to classes and used to hold SPARQL CONSTRUCT expressions. If an instance of the associated class is created, its constructor will be executed and the constructed triples inferred or asserted. The variable `?this` is pre-bound to the instance that was created.

3. Enter the following query into `spin:constructor` field of the new class:

```
CONSTRUCT {?this sml:url "http://www.topquadrant.com/topbraid/countries" }  
WHERE { }
```

This will ensure that every instance of `ImportCountries` class will have `http://www.topquadrant.com/topbraid/countries` in the `sml:url` property.

4. Select **Scripts > Edit/View SPARQLMotion scripts...**
5. Select the `Load countries` step, right click and select **Delete** to remove it from the script.
6. Expand **Export from Remote** folder and notice that you have a new module – **Import countries**, drag and drop it to create a new `LoadCountries` step. Connect it to `Export all`.
7. Double click on `Load countries`. You will see that its url field is automatically populated.
8. Modify `Export all` to create `exercise7-output.ttl` file. Adjust **baseURI** field as needed.
9. Test the script. It should produce the same results as Exercise 1.

The type of customization demonstrated in Exercise 7 can be applied to any module. It does not change the behavior of the module, but sets the default values for the appropriate fields. As mentioned before, completely new custom modules can be created by developing Java plugins.

SPARQLMotion can also be used to define new functions that can then be utilized in SPARQL queries. For more information see `Help > Help Contents TopBraid Composer > SPIN > User-Defined SPIN Functions and Templates`.

7 Next Steps

In this tutorial we have:

- Shown how to create, test, and execute SPARQLMotion scripts.
- Explained the iterative methodology for developing scripts.
- Demonstrated the use of SPARQLMotion to create web services.
- Provided a simple example of customizing available modules.

Only a small subset of the growing number of SPARQLMotion modules was used. We want to encourage the reader to experiment on their own with the modules not covered in the tutorial. As a starting point, additional worked examples are available at <http://www.topquadrant.com/products/SPARQLMotion.html>.

Questions related to the use of SPARQLMotion are welcome at TopBraid User Group Forum at: <http://groups.google.com/group/topbraid-users> .

Finally, TopQuadrant offers trainings for TopBraid Suite products. The classes cover advanced use of TopBraid Composer, including the development of SPARQLMotion scripts, as well as the use of TopBraid Live server for building and deploying semantic web applications.

For more information on the product training, please contact us at trainings@topquadrant.com.