TopBraid Composer™

Getting Started Guide

Version 5.2

September 21, 2016
## Revision History

<table>
<thead>
<tr>
<th>Date</th>
<th>Version</th>
<th>Revision</th>
</tr>
</thead>
<tbody>
<tr>
<td>August 1, 2006</td>
<td>1.0</td>
<td>Initial version</td>
</tr>
<tr>
<td>September 20, 2006</td>
<td>1.0</td>
<td>Correction to Exercise 18</td>
</tr>
<tr>
<td>July 21, 2007</td>
<td>2.0</td>
<td>Update with respect to the latest TopBraid Composer features.</td>
</tr>
<tr>
<td>May 24, 2011</td>
<td>3.0</td>
<td>Update to align with TBC 3.5 release</td>
</tr>
<tr>
<td>July 22, 2014</td>
<td>4.0</td>
<td>Update to align with TBC 4.5 release</td>
</tr>
<tr>
<td>September 21, 2016</td>
<td>5.2</td>
<td>Update exercises to TBC 5.1 release</td>
</tr>
</tbody>
</table>
Table of Contents

1 INTRODUCTION ........................................................................................................................................... 5
  1.1 CONVENTIONS ........................................................................................................................................ 5

2 INSTALLATION AND SET UP ..................................................................................................................... 7
  2.1 SYSTEM REQUIREMENTS .................................................................................................................. 7
  2.2 INSTALLATION .................................................................................................................................... 7
  2.3 BECOMING FAMILIAR WITH TBC .................................................................................................. 8

3 BUILDING YOUR FIRST ONTOLOGY WITH TOPBRAID COMPOSER ........................................... 14
  3.1 CREATE CLASSES ............................................................................................................................ 14
  3.2 CREATE PROPERTIES ..................................................................................................................... 19
  3.3 CREATE INSTANCES ......................................................................................................................... 24
  3.4 EXECUTE SPARQL QUERIES ............................................................................................................ 26
  3.5 EXTEND THE ONTOLOGY ............................................................................................................... 32

4 WORKING WITH IMPORTS AND MULTIPLE ONTOLOGIES .................................................... 34

5 DEFINING CLASSES USING OWL ....................................................................................................... 40
  5.1 RESTRICTION KEY WORDS ............................................................................................................ 42
  5.2 BOOLEAN CLASS CONSTRUCTORS ................................................................................................ 42
  5.3 COMPLEX CLASS EXPRESSIONS .................................................................................................... 42

6 MODELING WITH SHACL .............................................................................................................. 42

APPENDIX A: SEMANTIC WEB STANDARDS ..................................................................................... 43
  A.1 RDF .................................................................................................................................................. 44
  A.2 RDFS ............................................................................................................................................. 45
  A.3 OWL .............................................................................................................................................. 46
  A.4 SPARQL ....................................................................................................................................... 47
  A.5 SHACL ......................................................................................................................................... 48

Figures

Figure 1: Default Views of TopBraid Composer ......................................................................................... 7
Figure 2: Using hyperlinks to navigate ...................................................................................................... 9
Figure 3: Opening multiple ontologies .................................................................................................... 10
Figure 4: Creating a Project ..................................................................................................................... 11
Figure 5: Preferences dialog for classes .................................................................................................. 13
Figure 6: Help facility ............................................................................................................................... 13
Figure 7: Create RDF file dialog ............................................................................................................. 14
Figure 8: Initial screen after creating person ontology and setting the hierarchy start to owl:Thing15
Figure 9: Classes view—menu options ................................................................................................... 16
Figure 10: Creating a class ..................................................................................................................... 16
Figure 11: Initial class hierarchy for the Person Ontology .................................................................... 17
Figure 12: Classes view buttons ............................................................................................................ 18
Figure 13: Buttons and options available for TBC resource view .......................................................... 19
Figure 14: Properties view buttons ........................................................................................................ 20
Figure 15: Create new property ............................................................................................................. 20
Figure 16: Defined firstName property .................................................................................................. 21
Figure 17: Domain view for the Person class ................................................................. 21
Figure 18: Multiple classes as the refs:domain .............................................................. 23
Figure 19: Domain is a union ......................................................................................... 24
Figure 20: Instances View ......................................................................................... 25
Figure 21: Resource Form for William Shakespeare .................................................. 25
Figure 22: Add Statement to Resources ................................................................. 26
Figure 23: SPARQL View ................................................................................. 27
Figure 24: Results of query .............................................................................. 27
Figure 25: SPARQL View .............................................................................. 28
Figure 26: Widget menu .................................................................................. 30
Figure 27: SPARQL Query Library ........................................................................ 31
Figure 28: Imports view buttons ................................................................................ 34
Figure 29: Person ontology with import of topquadrant ontology .................................. 35
Figure 30: Class view ...................................................................................... 35
Figure 31: New shakespeare.ttl file ........................................................................ 37
Figure 32: Moving resources into shakespeare.ttl file ........................................... 38
Figure 33: Confirm move resources dialog ............................................................. 38
Figure 34: Edit resources dialog ........................................................................... 41
Figure 35: Defining cardinality restriction for the Person class ................................... 41
Figure 36: RDF Graph – example 1 ........................................................................ 44
Figure 37: RDF Graph – example 2 ........................................................................ 45
Figure 38: Subclasses of rdf:Property ..................................................................... 47
1 Introduction

This guide introduces TopBraid Composer™—an enterprise-class modeling environment for developing Semantic Web ontologies and building semantic applications. Fully compliant with W3C standards, Composer is the leading industrial-strength RDF editor and OWL ontology editor, as well as the best SPARQL tool on the market.

Throughout this document the terms TopBraid Composer™, TBC and Composer are used interchangeably.

Implemented as an Eclipse plug-in, Composer can be used to develop ontology models, convert data and models to and from RDF/OWL, transform and integrate data source integration, and develop Semantic Web services and applications. This document will cover only a small subset of Composer’s capabilities.

Section 2 of this guide provides information about installing TBC. It shows how to open and view existing ontologies, how to organize TBC workspace and how to create new projects.

Section 3 focuses on building a simple ontology (limited to RDFS vocabulary) and running test queries. It contains an exercise involving RDFS inferencing.

Section 4 explains import features and approaches to working with multiple ontologies.

Section 5 describes how to create key OWL constructs including restrictions.

Section 6 introduces the Shape Constraint Language (SHACL) and provides a link to a separate tutorial.

Appendix A provides additional information on the standards supported by TBC: RDF, RDFS, OWL, SPARQL, and SHACL. Readers who are new to these technologies will benefit from starting with the Appendix prior to moving on to section 2.

Once you master the basic set of features covered here, we invite you to explore a comprehensive Help system shipped with Composer. Many features not covered by this guide are explained in the help files. To access them, select Help > Help Contents menu and then click on TopBraid Composer. You can also find videos and demos on TopQuadrant’s web site.

1.1 Conventions

Class, property and individual names are written in a Times New Roman Bold font like this.

Names for user interface widgets and menu options are presented in an Arial bold font like this.

Where exercises require information to be typed into TBC a Times Courier New font is used 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 building ontologies 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.
2 Installation and set up

2.1 System Requirements
TopBraid Composer is implemented as an Eclipse plug-in. Eclipse is a powerful open-source platform for all kinds of modeling and programming languages. The integration of Composer into this platform means that you can exploit the benefits of an integrated development environment. Eclipse also provides an update mechanism that allows users to conveniently update plug-ins such as TopBraid Composer when a new version becomes available.

System requirements are the same as for the Eclipse 3.6 platform.

2.2 Installation

You will be asked to select the edition you want to download – TopBraid Composer Maestro Edition (TBC-ME), TopBraid Composer Standard Edition (TBC-SE) or TopBraid Composer Free Edition (TBC-FE). For this guide, we are assuming the use of TBC-ME, although all the steps work with TBC-FE as well.

You will also be able to choose the host operating system. After downloading the installation package, run the installer.

When installation completes, launch TopBraid Composer. You will be asked to choose a workspace, which is the Eclipse term for a folder storing a set of projects. Accept the default workspace name (make a note of the name for the upcoming “Organize the Workspace” section) and it will be created for you.

![Figure 1: Default Views of TopBraid Composer](image)
2.3 Becoming familiar with TBC

2.3.1 TBC Views

The TBC user interface is a collection of “views” or application windows. Each view has a tab showing its name and a set of view-specific icons. The following views are shown by default:

- The **Navigator** shows the projects and files in the Eclipse workspace
  - You will see at least 5 projects that are automatically installed with TBC-ME. For now, we will focus on TopBraid project which contains some sample ontologies.
- **Classes, Properties and Associations** views display the hierarchies of the current model
- **Resource Editor** is the main work area. When you open an RDF/OWL graph and navigate around, it will show the currently selected resource:
  - It includes tabs for a **Form, Graph, Source Code** and Browser (only TBC-ME)
  - When selected resource is a class, there is also a **Diagram** tab
  - When selected resource is an ontology, there are tabs for **Statistics, Profile** and **Overview**
- **Imports** shows the imports of the ontology (hierarchically)
- **Instances** shows members of the class selected in the **Classes** view
- **Domain** shows all properties in a domain of the currently selected class (or the most recently selected class, if the current resource is not a class)
- **Relevant Properties** shows a list of properties that are likely to be used for the selected resource
- **SPARQL** provides an interface to create, execute and manage SPARQL queries
- **Basket** can be used as a flexible drag-and-drop area for many purposes
- **Text Search** lets you find resource based on the text matches. It searches across all text properties of a resource
- **Shapes, Target and SHACL Validation** offer support for the new W3C RDF Shapes (SHACL) standard (some of these view were introduced in TBC 5.1 release while others may only be available or have been changed in 5.2 as TBC is tracking standard’s development)

In addition to these, a number of other views are available. To explore and select additional views, select from the top menu **Window > Show view**. You will see quite a number of additional options. All Composer views can be dragged and rearranged as necessary. They can also be resized, maximized and minimized.

To maximize a view, double-click on the tab with the view’s name –it’s header. Then double-click it again to restore the original size. To move a view, click on the tab (the tab color will become blue) and move it to the desired location.

Each view has a number of menu options. These are made available as icons displayed in the view’s header and additional options may be available when clicking on the down facing triangle icon in the header of a view.

We will now open and explore one of the files shipped with TBC.

Exercise 1: Navigating in TBC

1. In the **Navigator** view expand the TopBraid project, then expand the Examples folder and double-click on topquadrant.ttl to open it. The Resource Editor view will now show information about the ontology, while Classes and Properties views will show classes and properties that exist in the ontology.
2. In the **Classes** view, expand **owl:Thing** and double-click on the **schema:Country** class to select it. Information about **schema:Country** will appear in the **Resource Editor** view in the middle of your screen.

   Alternatively, you can use one of the following ways to select a resource:
   
   a. Single-click on an icon in front of **schema:Country** (since this is a class, it has a gold circle icon).
   
   b. Enter **schema:Country** in the toolbar field above the **Resource Editor** view. Here, CTRL+Space will auto-complete a name if you only enter the first few characters.

3. **schema:Country** is a qname\(^1\) or "qualified name" of the selected resource. Click on the underlined **Name** label in the **Resource Editor** view to see the full URI of this resource: [http://schema.org/Country](http://schema.org/Country).

4. Hover over the **schema:AdministrativeArea** in the **rdfs:subClassOf** widget in the **Form** tab of the **Resource Editor** and press CTRL. As shown in the next figure, **schema:AdministrativeArea** will become hyperlinked. Click on it. You should now have **schema:AdministrativeArea** displayed in the form.

There is always one ‘selected’ ontology model, and one ‘selected’ ontology resource (class, property, individual, etc.). The selected resource will always be from the selected model. The Resource Editor displays the currently selected resource, which is also shown in the toolbar.

---

\(^1\) Since URI references can be long, qnames enable the abbreviation of URIs. Qnames consist of a namespace prefix, a colon, and a local, un-prefixed name of a resource.
5. Notice the backward \( \Rightarrow \) and forward \( \Rightarrow \) arrows on Composer's tool bar. They enable quick navigation between current and previously selected resources similar to on a web browser. Click on the backward arrow. Now click on the forward arrow, and note how these let you navigate among resources displayed on the Resource Editor.

6. With the `schema:Country` class selected, click on the Instances view near the bottom of the screen to see a list of all countries.

7. Click on the **Toggle between qnames (with prefixes) and human readable rdfs:labels** icon in the menu area of TopBraid Composer to switch to the label-based display. This displays the more readable rdfs:label values of the resources in the Instances view instead of their qnames. Click again to toggle back to the qname-based display.

**Exercise 2: Switching between ontologies**

1. Go to the Navigator view and open the Kennedys ontology (`kennedys.ttl`). Your screen should look similar to the one shown below.

   ![Figure 3: Opening multiple ontologies](image)

   TopBraid Composer can open a number of different file formats. This includes all common serializations of RDF: RDF/XML (these files typically have .rdf extensions), Turtle (.ttl extensions), and N-triples (.nt extensions). Note, that TBC will not open a file with .owl extension. If you have RDF file with such extension, please rename it as per above.

   TopBraid Composer can also open XML and Excel files (.xls and .xlsx extensions) by converting them to RDF on the fly using TopBraid’s Semantic XML and Semantic Tables technologies. You can try this feature out by copying a spreadsheet into your workspace and double-clicking it to open. TopBraid can open database connection files, which store settings that configure connections to RDF databases and relational databases. Describing how to use TBC with databases or covering all file extensions is out of scope for this tutorial, but you can learn more about it by exploring help files.

2. Note that `kennedys.ttl` is your currently selected ontology, but `topquadrant.ttl` is still open. Switch back to that ontology by clicking on the tab with its name.

3. Close `topquadrant.ttl`. This can be done by either clicking on the “\( x \)” on the tab with its name or by selecting Close from the File menu.

4. Close `kennedys.ttl`. Note that you could have closed all the opened graphs using the Close All option on the File menu.
2.3.2 Create an Eclipse Project

In this exercise you will create your own project.

Exercise 3: Creating a new project

1. Select **File > New > Project...** from the menu.
2. On the next screen expand **General** on the tree display. Select **Project** and click the **Next** button, then type **tutorial.topbraid.com** as a project name. Click **Finish**.

You now can:
- set up folders within a project
- copy your existing files into the project
- create new files

Figure 4: Creating a Project
2.3.3 Organize the workspace

Exercise 4: Create folders in a project

1. In the **Navigator** view, right-click on the project you have just created and select **New □ Folder**
2. Give your new folder a name of your choosing and click **Finish**
3. You will see a new folder appear in the **Navigator** view.
4. Open your operating system’s file management system (such as Windows Explorer) and navigate to the workspace directory that you specified when you started up TopBraid Composer. You will see that a new folder has been created in it.

2.3.4 Copy files into a workspace

Exercise 5: Copy file to a workspace and open it

1. Place a file that you want to add to your workspace in to a copy buffer using any of the copy commands supported by your operating system. You may choose an RDF/OWL file if you have one or choose a file in a format that can be converted by TBC into RDF. A good choice would be an .xls or .xlsx file
2. In the **Navigator** view, right-click on the folder you have just created and select **Paste**.
3. You can now double-click on the file to open it.

---

A workspace corresponds to a folder in your computer’s file system. You can copy the file to any of the workspace folders using your computer’s file management system such as Windows Explorer. However, you will not see the file in the **Navigator** view until you refresh the appropriate folder.

Right click on a folder and select **Refresh**. If you update a file using a different program - for example, if you edit a text file with a text editor outside of the Eclipse environment - TBC will not know that the file version has changed unless you do a refresh.

You can also copy projects into the workspace using the same approach. For example, TopQuadrant provides projects for you to download with worked examples. When you copy a new project into the workspace, TBC needs to know that the workspace has been updated. If after refreshing your workspace you still do not see the new project, do one of the following:

- Right click in the **Navigator** and chose **Import … > General > Existing Projects into Workspace**, and then browse to select the project, or
- Create a new project and give it exactly the same name as the project you have copied, or
- Restart TBC.
2.3.5 Set up preferences

TBC is highly configurable. What is shown in many of the views is governed by the user preferences. Preferences are accessible from the down-facing triangle icon in the Classes and Properties views. On the Preferences dialog box, expand the tree under TopBraid Composer to see available preferences dialogs. The preference dialog for the Classes View is shown in the next diagram.

![Preferences dialog for classes](image)

2.3.6 Explore Help

This tutorial covers only a very small portion of Composer’s features. If you need additional information on any of the features described here or want to learn how to use other capabilities of TBC, you should check out its extensive Help facility.

Help is accessible from Help > Help Contents menu, under TopBraid Composer. Expand the tree under TopBraid Composer to see available help pages. Note also, in the upper-left of the Help window, that Help is searchable.

![Help facility](image)
3 Building Your First Ontology with TopBraid Composer

This chapter describes how to create a very simple ontology about people and family relationships.

Exercise 6: Create a new file

1. In the Navigator view, right-click directly on your new tutorial.topbraid.com project and select New > RDF File.
2. Replace unnamed in the Base URI field with person. Note how this will be automatically entered into the File name field when you do so. The default file extension is ttl, for a turtle file; you can select a different RDF serialization here if you wish.
3. Click Finish.

Figure 7: Create RDF file dialog

3.1 Create classes

When a new file is created, the screen should resemble the screen in the next figure. Depending on your setup, rdfs:Resource may be the root node in the Classes view, but you can change this by clicking on the down-facing triangle icon in the Classes view and selecting Start hierarchy with owl:Thing from the drop down menu. We recommend that you do it for this exercise, so that your view is less cluttered; using rdfs:Resource as the root becomes important when you don’t want to use OWL classes.
OWL classes are interpreted as sets of individual resources. The class owl:Thing represents the set containing all individuals. Because of this all classes are subclasses of owl:Thing.

Let’s add some classes to the ontology.

**Exercise 7: Create classes Person, FemalePerson and MalePerson**

1. Right click on owl:Thing.
2. Select the Create subclass choice from the context menu as shown in the next figure. This creates a new subclass of the selected class (in this case, a subclass of owl:Thing). Alternatively, you can use icon buttons in the banner of the Classes view.
3. The default name shown in Create classes dialog will be **Thing_1**. Rename it to **person:Person**. Check to select **rdfs:label** in the **Annotations Template** portion of the dialog and enter `{name}` in the **Initial Value** field. This tells TBC to generate a human readable label from the class name. Click **OK**. (This is especially useful with more complex, multi-word class names.) The new class will be displayed in the **Form** view.

![Figure 9: Classes view—menu options](image)

![Figure 10: Creating a class](image)
4. The form for each resource already has all fields (properties) that have been defined for the resource; in this case, it shows the rdfs:label and rdfs:subClassOf properties that you set with the Create Classes dialog box. If a field is not present on a form, it can be easily added. For example, drag and drop rdfs:comment from the Properties view into the Annotations section of the Class Form.

5. Type ‘Human being’ in the rdfs:comment widget. Click on the right of the field or simply press Enter. The black frame around the widget needs to disappear.

6. Repeat steps 2 and 3 to add the classes person:FemalePerson and person:MalePerson, ensuring each time that Person is selected before the Create subclass button is pressed so that the classes are created as subclasses of Person. (If you accidentally create person:MalePerson as a subclass of person:FemalePerson, you can always drag it in the classes tree onto the Person node to change its parent. This kind of drag-and-drop editing of class hierarchies is one of the features that makes TBC the most efficient way to edit ontologies.)

7. Double-click on person:MalePerson in the Classes view to show its Class Form. On the Class Form, hover your mouse over the icon to the left of the Person value of the rdfs:subClassOf property and click on the plus sign that appears. This will show the nested Class Form for Person inside the Class Form for person:MalePerson.

The class hierarchy and the Class Form for person:MalePerson should now look like the next figure.

![Figure 11: Initial class hierarchy for the Person Ontology](image)

8. Notice a star in front of the ontology name above the Class Form - *person.ttl. This means that the ontology has been modified, but not saved. Select File > Save or click Ctrl+S.
In Composer, classes have a gold circle icon displayed in front of their names. The selected class (the one currently shown in the form) has an arrow overlaying the gold circle icon in the **Classes** view. Observe that `person:FemalePerson` on the left side of Figure 11 has an arrow in the icon.

**Classes** view has a number of buttons, as explained in the next figure. Hovering over each icon displays a tooltip showing its purpose; most of these features are also available on the context menu displayed when you right-click a class name.

![Figure 12: Classes view buttons](image)

**Figure 12: Classes view buttons**

**Resource** view has a number of buttons and options; most of these are explained in the next figure. You will also see the **Browser** tab that is not shown in the figure below. This tab lets you see various HTML representation of the information about a resource.
3.2 Create Properties

Properties represent connections between an RDF resource and either another resource or a literal. For example, your new MalePerson class has an rdfs:subClassOf connection to the Person class resource and an rdfs:label connection to the “Male person” string literal.

There are two main types of properties in OWL - Object properties and Datatype properties. Object properties link a resource to a resource. Datatype properties link a resource to a literal. Literals are XML Schema Datatype values such as strings and integers.

OWL also has a third type of property – Annotation. Annotation properties are used to store information that is ignored by the reasoning tools. This are typically labels, comments and notes. In exercise 5 we used an annotation property rdfs:comment to add a comment to the Person class.

All the types of OWL properties are subclasses of rdf:Property.

In Composer’s Properties view, properties have rectangular icons displayed in front of their names. Object properties are indicated using blue icons, datatype properties have green icons and annotation properties have yellow icons. RDF properties have navy icons. The property currently shown in the form has an arrow overlaying the rectangular icon.

The Properties view has a number of buttons as explained in the next figure.
Properties may be created using the Create property button in the Properties view shown in the figure above (labeled there as “Add new property”). Right-clicking a property name in the Properties view displays a context menu that lets you create new properties, similar to the way right-clicking a class in the Classes view brings up such a menu. A property created with this approach will become a subproperty of the selected property.

Exercise 8: Create datatype properties called firstName and lastName

1. Click the Create property button. The Create property dialog will appear as shown in the next figure.
2. Select owl:DatatypeProperty. Name the new property person:firstName as shown below. Add xsd:string to the range of this property to show that firstName values will always be of this type. Click OK.
3. Add the **Person** class to the domain of the newly created property to show that a resource with a firstName property is a member of the class **Person**. This can be done in one of the following ways:
   a. Drag **Person** from the **Classes** view and drop it on the **rdfs:domain** property in the form showing the properties of your new firstName property
   b. Click on a **Show widget menu** button next to **rdfs:domain** and select **Add empty row.** (**Show widget menu** button is a down-facing triangle icon located next to each widget on the form as shown in Figure 13.) Type `person:Person` and click **OK**.
   c. Click on a **Show widget menu** button next to **rdfs:domain** and select **Add existing...** Select **Person** from the tree in the dialog and click **OK**.

4. Your screen should now look like the one shown in the next figure.

![Datatype Property Form](image1.png)

**Figure 16:** Defined firstName property

5. Repeat the steps above to create a **person:lastName** property.
6. Select the **Person** class in the **Classes** view.
7. Click on the **Domain** view. Observe (as shown in the next figure) that the newly created properties appear in the view.

![Domain view for the Person class](image2.png)

**Figure 17:** Domain view for the Person class

TopBraid Composer, Copyright 2006-2016 TopQuadrant, Inc. 21 of 48
Exercise 9: Create object properties called hasDaughter, hasSon and hasChild

1. Click the Create property button. The Create property dialog will appear.
2. Select owl:ObjectProperty, because this property will have other objects as values and not string values. Rename the new property to person:hasDaughter and click OK.
3. Set the domain of the newly created property to be Person and range to be FemalePerson.
4. Add another object property called person:hasSon.
5. Set the domain of the newly created property to be Person and range to be MalePerson.
6. Add a third object property called person:hasChild.
7. Make hasChild a parent of hasDaughter and hasSon. It can be done in either of the following ways:
   a. Select hasDaughter. Drag and drop hasChild over rdfs:subPropertyOf field.
   b. In the Properties view select hasSon, and drag and drop it under hasChild.

Domain and range declarations are global. They apply anywhere a property is used. The Semantic Web standards RDFS and OWL define inferences or new statements (triples) that result from existing statements.

Given the definitions we just made, when RDFS inferencing is used, any resource that is a subject in an RDF triple with a predicate hasDaughter will be inferred to be a member of the class Person. Any resource that is an object of such a triple will be inferred to be a member of the class FemalePerson. Because we made hasDaughter a subproperty of hasChild, it will also be inferred that the subject has a hasChild value of the object.
Unlike other data models, such as relational databases, or object oriented modeling, properties are independent of classes and can be re-used in different contexts. This can be done by specifying in a model a property connection to one or more classes. Furthermore, at any point in TBC you can drag and drop a property field into any form even if there are no model definitions that say that this property can be used.

It is possible to specify multiple classes as the domain or range for a property. One can, for example, drag and drop multiple classes over the `rdfs:domain` widget. See example in the figure below.

![Figure 18: Multiple classes as the rdfs:domain](image)

It is important to understand that multiple domain statements are interpreted as intersection. If the domain of a property has two classes `MalePerson` and `FemalePerson`, any instance that is in the domain of the property will be assumed to be of both types.

If you want to say that a domain is a union of both classes (i.e., an instance can be either a `MalePerson` or a `FemalePerson`), you should put `MalePerson` and `FemalePerson` on the same line and type ‘or’ between them as shown in the next figure.
3.3 Create instances

We are now ready to add a few instances to the ontology.

**Exercise 10**: Create instances of the Person class: Susanna_Shakespeare, Judith_Shakespeare, Hamnet_Shakespeare and William_Shakespeare.

1. Select the FemalePerson class.
2. Click on the Instances view. Click the Add new instance button shown in the next figure. When the Create FemalePerson dialog pops up, replace the default name of new instance with person:Susanna_Shakespeare and click OK.
3. Repeat for person:Judith_Shakespeare.
4. Select the MalePerson class and add instances person:Hamnet_Shakespeare and person:William_Shakespeare. At this point your screen should look similar to the one shown in the next figure.
5. Drag the hasSon and hasDaughter widgets onto the William Shakespeare form.

6. Drag Hamnet_Shakespeare over the hasSon widget in the William Shakespeare form.
7. State that Judith and Susanna are William’s daughters. In addition to the drag and drop method used in the previous step, this can be done in a number of ways. Try each to experience different ways of working with TBC:
   a. Use ‘add existing’ menu option:
      - Click on a Show widget menu button next to hasDaughter and select Add existing…
      - Select Judith_Shakespeare and Susanna_Shakespeare from the Add existing screen and click OK.
   b. Type in the required information:
      - Click on a Show widget menu button next to hasDaughter and select Add empty row.
      - Type in Judith_Shakespeare.
      - Repeat for Susanna trying an auto-complete feature. Type in Sus and hold the CTRL key while pressing SPACE.
c. Use a basket:
   - Select the FemalePerson class.
   - Click on the Instances view.
   - Drag Judith_Shakespeare and Susanna_Shakespeare into the Basket view
   - Select the MalePerson class. Click on the Instances view and select William_Shakespeare.
   - Drag Judith_Shakespeare from the Basket view over hasDaughter widget in the William Shakespeare form.
   - Select Susanna_Shakespeare in the Basket view
   - Take the diamond icon to the left of Judith_Shakespeare in William’s form and drop it on to the Basket view.
   - You will see the dialog box shown in the next figure. Click OK. This approach does not save much time in our case, but because you can highlight multiple resources in a basket (using Ctrl Shift keys), it will become very handy when you need to add several statements with the same subject and predicate and different subjects.

![Figure 22: Add Statement to Resources](image)

You can easily change the type of any resource after it has been created. Let’s say, for example, that you have created Susanna_Shakespeare as an instance of a Person. You want to say she is a FemalePerson. Simply select Susanna_Shakespeare and drag FemalePerson over the rdf:type widget.

### 3.4 Execute SPARQL Queries

SPARQL is a standard for querying RDF/OWL data. TBC comes with a built-in query engine for SPARQL. Let’s try some queries now.

**Exercise 11: Run a default query.**

1. Click on the SPARQL view near the bottom of the screen. Its options and layout are explained in the next figure.
2. A starter example query is already in the query panel. It will list all triples that fit the pattern ‘x rdfs:subClassOf y’ and will display x and y. In other words, it will list resources that are subclasses of another class. The result will include resources (subjects) as well as the classes they are subclasses of (object).

3. Run the query by clicking the green arrow (the one without the blue circle behind it) or by pressing Ctrl+Enter with the SPARQL view selected. Observe that results include classes we have just created as well as built-in OWL and RDFS classes.

Let’s write a query to retrieve all people who have daughters. Following the example above, a query to return all parents with their daughters should look like this:

```
SELECT ?subject ?object
WHERE { ?subject rdfs:hasDaughter ?object }
```

Paste or type this query into the SPARQL view to replace the existing query and execute it.

The SPARQL window helps guide the creation of your queries by highlighting SPARQL keywords in purple and adding red lines under property names that it doesn’t recognize. For example, try adding an “X” at the end of rdfs:hasDaughter in your query, and note the red squiggly line that immediately appears under rdfs:hasDaughterX. This shows that rdfs:hasDaughterX is not a property in the existing ontology and would do nothing in the query. (Remove the “X” before continuing.)
Introducing a new prefix is particularly useful if you are working on a project that consists of multiple namespaces and modules.

When you create an RDF file, unchecking “Set a default namespace in the new file” box in the Create dialog will automatically create a prefix for the base URI.

To see how you can define prefixes in Composer, let’s create a prefix for the namespace http://tutorial.topbraid.com/person#.

**Exercise 12: Create a namespace prefix.**

1. Click on the home icon shown under the TBC level menus to display the Resource Form for the ontology itself.
2. Select the Overview tab and click the Add button to the right of the table of Namespace Prefixes.
3. Type person in the Prefix column of the new row. Type http://tutorial.topbraid.com/person# in the Namespace URI column.
4. Press Enter.
5. Click File > Save or Ctrl+S to save your work.

**Exercise 13: Query for all parents of females.**

1. Click on the SPARQL view and change rdfs:hasDaughter to person:hasDaughter.
2. Click the Execute SPARQL button. Your screen should look similar to the one shown in the next figure.

![Figure 25: SPARQL View](image)

3. Change person:hasDaughter to person:hasChild in the query.
4. Run the query. You should receive no results.

Why have we received no results? hasDaughter and hasSon are subproperties of hasChild, so according to RDFS inference rules (explained in Appendix A), the query should have returned William Shakespeare with all his children.

There is a simple explanation. We have been making queries over the asserted (or stated) triples only, but not on the inferred triples. There are no asserted triples with person:hasChild as a predicate.
There are two ways of addressing this problem:

- Directly in the query
- Using an inference engine

5. Change the query to the following, which asks for the subject and object of any triples that have person:hasChild as a predicate or any superproperties of person:hasChild:

   ```sparql
   SELECT ?subject ?object
   WHERE {
     ?p rdfs:subPropertyOf* person:hasChild .
   }
   ```

6. Run the query. You should see three children listed.

In the next exercise we will use an inference engine to run the inferencing and see how it changes query results.

**Exercise 14: Run inferences and query for all parents.**

1. Select **Inference > Run Inferences** menu option. (If running the free version of TBC, then before picking Run Inferences, you may first need to click the home icon, and go to the ontology's Profile tab, and check OWL 2 RL to turn that SPIN library on.)

2. Inferred triples will appear in the **Inferences** view, including three person:hasChild triples for William.

3. In the **SPARQL** view, click on the **Use Currently Configured Inferences** icon, which toggles whether the inferences should be used in the queries. This will enable the inferences to be used in the next query. Then run the following query.

   ```sparql
   SELECT ?subject ?object
   WHERE { ?subject person:hasChild ?object }
   ```

   You should see William Shakespeare with all three of his children. (With the free version, there is no **Use Currently Configured Inferences** button. Once the OWL 2 RL inferencing is turned on as described above, executing a SPARQL query with the green triangle button will use the inferencing.)

4. In the **Instances** view, click on **William_Shakespeare** and observe that his form now includes inferred properties. These are shown in the light blue-gray background.
5. Save the query for future use:
   - Select **person:Person** class in the **Classes** view.
   - Click on **Attach current query to the selected resource** icon in the **SPARQL** view. Press OK on the **Confirm** dialog that appears to ask about importing the SPIN namespace; this makes it possible to attach queries to classes. You will then see that the query has been added to the **spin:query** field on the **Person** Class Form.
   - There are two ways to run saved queries:
     a. Click on the triangular **Show widget menu** button to the right of the **spin:query** field (not the **spin:query** name) in the form. Select **Execute as SPARQL query**.
     b. Select the **Query Library** tab on the **SPARQL** view, where you will see this and other saved queries in the ontology. Make sure that the checkbox near this query is checked and the **Use currently configured inferences** button is selected. Run the query using **Execute SPARQL** button. The **Query Library** tab will appear with the result as shown in the figure:
Figure 27: SPARQL Query Library

TBC will recognize SPARQL syntax in values of a class’s `spin:query` property and provide menu options to execute them.

To find `spin:query` in the Properties view, type `query` in the search box at the bottom of the Properties view, then click on the search icon next to it. The same approach lets you search for class names in the Classes view.

You can also associate rules and constraints with ontology resources. To learn more about this, take a look at the SPIN tutorial available at: http://www.topquadrant.com/spin/tutorial/index.html
6. Export your query results:
   - In the SPARQL view click on Export results to file icon.
   - In the dialog that shows up, give a name to the file about to be created and select its format before clicking the Save button.

### 3.5 Extend the ontology

In this section we will add two more object properties and observe the inferences they entail.

**Exercise 15: Create an object property called hasSpouse**

1. Click the Create property button in the Properties view. The Create property dialog will appear.
2. Expand the tree under owl:ObjectProperty. Select owl:SymmetricProperty.
3. Rename the new property to person:hasSpouse and click OK.
4. On the new property’s Property Form, set its rdfs:domain and rdfs:range to be person:Person.

Prior to this exercise, only the RDFS vocabulary has been used to define properties of the person ontology. (See Appendix A for more information on RDFS and OWL.) This is the first time an OWL construct is being used.

---

**A symmetric property (identified here as p) entails the following inferences:**

1. If there is triple a p b, it will be inferred that b p a. If our example had included the resource Anne_Hathaway, then if William_Shakespeare hasSpouse Anne_Hathaway, it would be inferred that Anne_Hathaway hasSpouse William_Shakespeare because hasSpouse was defined as a symmetric property.
2. If p rdfs:domain a then p rdfs:domain b. Similarly, if p rdfs:range a then p rdfs:range b.

---

Even though we have defined classes as OWL classes and properties as OWL properties, we have not done any modeling that required OWL expressivity. All the exercises thus far could have been accomplished without using any of the OWL statements. For example, instead of creating subclasses of owl:Thing, we could have created subclasses of rdfs:Resource, which are declared as RDFS classes.

Composer can be configured as an RDFS-only editor by ‘hiding’ all of OWL constructs. This can be done by going to Preferences and then making appropriate selections under TopBraid Composer > Classes View and Properties View.
Most inferences will only appear in Composer if you explicitly run the inferencing. However, there are a limited number of trivial inferences that Composer performs interactively or ‘just in time’. These are:

- Coordination of inverses. If it is stated that property $p$ owl:inverseOf $q$, TBC will infer that property $q$ owl:inverseOf property $p$.
- Coordination of domains and ranges of inverse properties. If it is stated that $p$ rdfs:domain $a$ and $p$ owl:inverseOf $q$, TBC will infer that $q$ rdfs:range $a$

Because TBC maintains these automatic inferences we recommend that if you use inverse properties, you should specify domains and ranges only for the properties ‘going in one direction’ and let TBC maintain the domains and ranges for their inverses.

Given that you can always query with SPARQL in any direction, we generally discourage users from specifying inverse properties.

**Exercise 16: Create object property called hasFamilyMember**

1. Click the Create property button. Create property dialog will appear.
2. Select owl:ObjectProperty. Rename the new property person:hasFamilyMember and click OK.
3. Make hasChild and hasSpouse subproperties of hasFamilyMember by dragging them onto that property in the Properties view.
4. Select File > Save to save your work.

When subproperties have their domains and ranges defined, it is usually unnecessary and even not advisable to define domain and ranges for the parent property. Another ontology design pattern (less commonly used, but applicable in some cases) is to define the domain and range for a parent property and leave out domain and range definitions for subproperties.
4 Working with Imports and multiple ontologies

In this chapter, we will combine the topquadrant.ttl and person.ttl models, connecting them to describe where people were born in.

The topquadrant.ttl ontology is accessible under TopBraid > Examples > topquadrant.ttl

Rather than replicate this information in person.ttl, we will show how to import an ontology and reuse it.

OWL ontologies may import one or more other OWL ontologies. Once imported, not only can classes, properties and individuals be referenced by the importing ontology, the axioms and facts that are contained in the ontology being imported are actually included in the importing ontology. OWL allows ontology imports to be cyclic so that, for example, the topquadrant ontology may import the person ontology, and the person ontology may import the topquadrant ontology. For our exercise we have decided to import the topquadrant ontology into person ontology.

Notice the distinction between referring to classes, properties and individuals in another ontology using namespaces and completely importing an ontology.

Exercise 17: Import topquadrant ontology and make changes

1. In the Navigator view, open person.ttl file.
2. In the Classes view menu, select Start hierarchy with owl:Thing.
3. Click on the Imports view and drag and drop topquadrant.ttl from the Navigator into the Imports view.

Another way to specify imports is by pressing the Import local file... button shown in the next figure:

![Figure 28: Imports view buttons](image)

When the Import local file dialog pops up, expand the workspace folders until you've located the person countries, select it and click OK.
4. Your screen should now look similar to the one shown in the next figure. Note that some classes and properties (for example, `owl:Thing > schema:Country`) are displayed using ‘washed out’ icons and fonts. These are the resources that come from imported model.

![Figure 29: Person ontology with import of the topquadrant ontology](image)

5. Create an object property called `person:bornIn`. Set its domain to `Person` (from the imported Person model) and its range to `Country` (from the topquadrant model). You have just created a bridge between two models!

6. Save your changes using **File > Save**.

7. When dealing with multiple imported ontologies, it can be useful to view classes and properties sorted by namespace. At the bottom left of the **Classes** view, click on **Group by namespaces** icon to organize the tree view by namespaces. With this presentation, the rdfs:subClassOf hierarchy is not shown. Click it again to restore the class tree view. Note that a similar feature is available with **Properties** view.

8. Select the `Person` class and modify its name to be `person:HumanBeing`. Press ENTER.

9. Close person ontology by using **File > Close**.

To import RDF graphs located on the web, use **Import from URL** button.
10. A dialog will pop-up offering to save the changes. Since we do not want to save the most recent change, click on **Deselect All** then click **OK**.

When working with multiple ontologies, it is important to know where the new statements and/or changes to the old statements are saved. Composer follows these rules:

- New statements are added into the currently selected ontology. The property `person:bornIn` was added to `person.ttl`.
- Data on forms for imported resources can be edited. Any changes to the existing statements are written into the ontology they come from. For example, if `topquadrant.ttl` would import `person.ttl` and while viewing the `topquadrant.ttl` we were to modify the fact that `FemalePerson` is a subclass of `Person` (essentially remove the triple that says `person:Person rdfs:subClassOf person:FemalePerson`), the change would go into the `person.ttl` file.
- If we were to say that domain of `hasSon` is no longer a `Person`, but `Parent` (a new class we can define for this purpose), the location where the new triple will go depends on how the change is made:
  - If we were to overtype `Person` with `Parent`, the change would be saved in the `person.ttl` as an update to the previously existing triple in that file.
  - If we were to delete the entry about the domain and then add a new one, the deletion would be done in `person.ttl` and the new triple would be saved in the `topquadrant.ttl` (if we made such change from `topquadrant.ttl` and `topquadrant.ttl` would import `person.ttl`).
- When we changed the URI of the `person:Person` class to `person:HumanBeing`, the change was made to the `Person` class definition in `person.ttl`. TBC resolved and updated all the references to this class in `person.ttl`. It also scanned to see if there are any other ontologies that import it.

When working with modular imported ontologies, it is, therefore possible to intentionally or accidentally make changes to imported files. If imported files come from the Web, such changes will be lost when you close the model. With local files they can be saved.

Composer keeps a log of all changes – accessible from the **Change History** view. Unsaved changes can be rolled back using **Edit > Undo**.

When working with the local models that belong to other parties, a good practice is to lock them (make them read only) to prevent accidental updates. This can be done by selecting the file in the **Navigator** view and clicking the **🔒** button.

The person ontology has some general (schema level) information about people. It also has some very specific information about the Shakespeare family. Since we are interested in the general cases of relationships between people and their travel interests, it makes sense to separate information about the Shakespeare family into a file of its own.

**Exercise 18: Move resources between ontologies**

1. In the **Navigator** view, create a new RDF file in your tutorial.topbraid.com project, calling it `shakespeare.ttl`. Invent a base URI of your choice.
2. Import `person.ttl` into `shakespeare.ttl` by dragging it from the **Navigator** view onto the **Imports** view. Your screen should now look similar to the one shown in the next figure.
3. Select the **File > Save** menu option. Notice that star in front of the file name disappear.
4. Open person.ttl.
5. Select the **MalePerson** class in the **Classes** view and click on the **Instances** view.
6. Drag **William_Shakespeare** and **Hamnet_Shakespeare** from the **Instances** view into the **Basket** view.
7. Select the **FemalePerson** class.
8. Drag **Judith_Shakespeare** and **Susanna_Shakespeare** into the **Basket**.
9. Select all four resources in the **Basket** and drag and drop them over the Shakespeare.ttl file in the **Navigator** as shown in the next figure.
10. A **Confirm move resources** dialog will pop up. Click **Yes**.

11. Observe that the **Person** classes in person.ttl no longer have instances associated with them.

12. Switch to the shakespeare.ttl file and locate the William Shakespeare resource. Observe that the connections between him and his children are still in place.
Notice that in the previous exercise all four individuals were moved at once. Let’s examine what would happen if we were to move person:William_Shakespeare first and then move person:Susanna_Shakespeare:

- Saying ‘yes’ to the ‘should the namespace be adjusted’ question would change the record id of person:William_Shakespeare to shakespeare:William_Shakespeare
- The record id for person:Susanna_Shakespeare would stay the same. Therefore, the Shakespeare.ttl file would now have the following triple: shakespeare:William_Shakespeare person:hasDaughter person:Susanna_Shakespeare
- If we were now to move Susanna and say ‘yes’ to the namespace adjustment, her record id would change to shakespeare:Susanna_Shakespeare and connection between her and William would be lost.

If you can move all resources at once, Composer will maintain relationships between connected resources as their URIs are modified. Say ‘yes’ to the question about adjusting the namespace.

If you cannot move all the connected resources at once, for all the moves except for the very first one, say ‘no’ to the adjusting the namespace question and modify the resource IDs so that they include correct namespaces manually after the move.

As demonstrated in exercise 17, if you change the URI of a resource, TBC will check to see if there are any files that import this model and may therefore be impacted by renaming. The dialog box will list the (potentially) impacted models. Under your direction, TBC will propagate the change to all affected files.

For more granular control over moving operation use the Triples view accessible by selecting Window > Show View > Triples.
5 Defining Classes using OWL

As noted before, all RDFS declarations about properties are global. If it is stated that Person is in the domain of the hasChild property, this declaration remains true everywhere that the hasChild property is used. In other words, it defines the property hasChild and not the class Person.

Let’s consider the following example:
- We have already said that Person is bornIn Country by using the rdfs:domain statement.
- We now want to say that at there can be only one country a person is born in.

This is where OWL restrictions come in. Unlike domains and ranges, restrictions define classes. They are used to restrict the individuals that belong to a class. OWL supports the following restrictions:
- Quantifier Restrictions – allValuesFrom\(^2\) and someValuesFrom\(^3\)
- Cardinality Restrictions – minCardinality, cardinality and maxCardinality
- hasValue Restrictions

Restriction can be declared using either rdfs:subClassOf or owl:equivalentClass statements. The difference is the inferences that can result from the declarations. For example:

- Saying that US Citizen is a subclass of all things for which the value of the nationality property equals (hasValue) ‘USA’, means that:
  - if it is known that an individual is US Citizen, it can be inferred that his nationality is ‘USA’
- Saying that US Citizen is equivalent to all things for which the value of nationality property equals (hasValue) ‘USA’, means that:
  - if it is known that an individual is US Citizen, it can be inferred that his nationality is ‘USA’ AND
  - if it is known that an individual’s nationality is ‘USA’, it can be inferred that he is a US Citizen.

Let’s use the example introduced in the beginning of the section and create a restriction.

Exercise 19: Create someValuesFrom restriction using the Edit Restriction dialog

1. On the Class Form for person:Person, near the owl:equivalentClass widget, click on the Show widget menu button (shown in Figure 18) and select Create restriction... The Edit Restriction dialog will pop up.
2. At the dialog, select bornIn property from the On Property tree and the cardinality (exactly) radio button from the Restriction Type options. In the Filler field, enter the value 1 schema:Country to show that person can be born in exactly one country. Click on OK. Before doing so, the dialog will look as in the following figure:

\(^2\) Also called universal quantifiers
\(^3\) Also called existential quantifiers
Your screen should now look like the one shown in the next figure. Note that restriction is specified in words using a de-facto standard notation called Manchester OWL syntax after the University of Manchester where it was developed.

As an alternative to using dialogs, you can enter restrictions by adding an empty row in the owl:equivalentClass or rdfs:subClassOf widgets and directly typing it in using Manchester syntax.

The sections that follow explain the most commonly used Manchester syntax keywords.

---

4 http://www.w3.org/2007/OWL/wiki/ManchesterSyntax
5.1 Restriction Key Words

<table>
<thead>
<tr>
<th>OWL</th>
<th>DL Symbol</th>
<th>Manchester Syntax Keyword</th>
<th>Example</th>
<th>Inferences</th>
</tr>
</thead>
<tbody>
<tr>
<td>someValuesFrom</td>
<td>some</td>
<td>hasChild some Man</td>
<td></td>
<td>When used with owl:equivalent class, enables inferences about a type of a subject of a triple</td>
</tr>
<tr>
<td>allValuesFrom</td>
<td>all</td>
<td>hasSibling all Woman</td>
<td></td>
<td>Enables inferences about a type of an object of a triple.</td>
</tr>
<tr>
<td>hasValue</td>
<td>has</td>
<td>hasCountryOfOrigin has England</td>
<td></td>
<td>When used with owl:equivalent class, enables inferences about a type of a subject of a triple</td>
</tr>
<tr>
<td>minCardinality</td>
<td>min</td>
<td>hasChild min 3</td>
<td></td>
<td>When used with owl:equivalent class, enables inferences about a type of a subject of a triple</td>
</tr>
<tr>
<td>Cardinality</td>
<td>exactly</td>
<td>hasChild exactly 3</td>
<td></td>
<td>When used under the open world assumptions, does not result in any classification inferences</td>
</tr>
<tr>
<td>maxCardinality</td>
<td>max</td>
<td>hasChild max 3</td>
<td></td>
<td>When used under the open world assumptions, does not result in any classification inferences</td>
</tr>
</tbody>
</table>

Note that OWL allows hasValue restrictions to have a datatype literal as filler. Examples for the syntax for these in TopBraid Composer are as follows:

- "value" for xsd:string literals
- 42 for xsd:int literals
- 4.2 for xsd:float literals
- true or false for xsd:boolean literals

5.2 Boolean Class Constructors

<table>
<thead>
<tr>
<th>OWL</th>
<th>DL Symbol</th>
<th>TopBraid Composer Syntax Keyword</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>intersectionOf</td>
<td>and</td>
<td></td>
<td>Doctor and Female</td>
</tr>
<tr>
<td>unionOf</td>
<td>or</td>
<td></td>
<td>Man or Woman</td>
</tr>
<tr>
<td>complementOf</td>
<td>not</td>
<td></td>
<td>not Child</td>
</tr>
</tbody>
</table>

5.3 Complex Class Expressions

Complex class expressions can be constructed using the above boolean constructors and restrictions. For example,

Person and hasChild some (Person and (hasChild all Man) and (hasChild some Person))

describes the set of people who have at least one child that has some children that are only men (i.e., grandparents that only have grandsons). Note that brackets should be used to disambiguate the meaning of the expression.
6 Modeling with SHACL

SHACL (Shapes Constraint Language) is a data modeling language slated to become an official W3C Recommendation in early 2017. As more people work with data coming from a variety of sources, especially for data integration projects, SHACL gives them a way to describe the “shapes” of the data that they’re working with so that applications can take better advantage of that data. In addition to describing which properties go with which classes (for example to drive user input forms), SHACL lets you define constraints on data that, when used by your applications, can make it easier to improve the quality of your data with standardized models instead of procedural code.

TopQuadrant is an active participant in developing SHACL standard and is supporting SHACL in all products. Unlike OWL, SHACL (as SPARQL does as well) takes a "closed world" approach to interpreting models. This is necessary in order for the data validation to produce expected results. SHACL shapes and constraints could be used in combination with OWL ontologies or you could decide not to use OWL and model entirely in SHACL. SPIN was a key input to SHACL and, for data constraints, SHACL can be seen as SPIN 2.0. SPIN, however, has a broader scope than SHACL. For example, SHACL does not yet support rules for inferring new facts. SHACL and SPIN can be used together to cover the entire scope of modeling and application needs.

For a beginner’s tutorial of how to use TBC to model with SHACL, see http://www.topquadrant.com/technology/shacl/tutorial/.
Appendix A: Semantic Web Standards

A.1 RDF

The Resource Description Framework (RDF) is a language for representing information about resources that reside in and can be accessed via the internet infrastructure. Each RDF resource has a unique ID that is based on standard Web identifiers called Uniform Resource Identifiers (URIs) (URIs are unique symbolic strings similar to URLs, but they don't have to resolve to actual locations.)

RDF has an XML serialization and many who are familiar with XML will think of RDF primarily in terms of that syntax. This is not a correct understanding of RDF. RDF should be understood in terms of its data model. RDF data can be represented (serialized) in XML, but it also has a number of other serialization syntaxes, such as N3 and Turtle.

RDF statements are often called 'triples' because they consist of three parts: a subject (rdf:subject), predicate (rdf:predicate) and object (rdf:object). Each entry is an RDF resource. Statements that contain resources with the same IDs are merged. For example, the following three triples can be brought together:

- flights:FlightDL210 flights:hasDestination geo:London
- geo:London rdf:type geo:City
- :John :livesIn geo:London

The resulting in the graph shown below.

![Figure 36: RDF Graph—example 1](image)

RDF data should be thought of as a graph of nodes and arcs where nodes are subjects and objects and arcs are predicates. A triple statement itself can be given a resource ID using instances of the rdf:Statement class. This makes it possible in RDF to say things about statements.

The predicate of one statement can be a subject or an object of another statement as shown in the graph below where livesIn, a predicate in one of the statements of the first graph (above), is a subject of the livesIn rdf:type owl:ObjectProperty statement in the second graph (below).
Figure 37: RDF Graph - example 2

You can generate diagrams like these using the Graph Panel in the Composer's Resource Editor View.

The prefixes in front of resource IDs (for example, rdf: or owl:) shown in the graph figure represent namespaces. There exists a comprehensive (and somewhat arcane) set of syntax rules for using XML namespaces in RDF. Composer implements these rules for you. The only thing you need to do as a user is to provide a namespace for each ontology or RDF dataset. Additional information about naming conventions for namespaces is available in the help panel within TBC titled “Naming Conventions and Namespaces.” (You can find this either in the Help facility’s Table of Contents or by searching the Help for this title.)

While RDF includes an `rdf:type` property, the RDF data model does not provide a way to express custom schemas for RDF data—that is, there is no notion of classes or class definitions in RDF. Schemas are provided by the languages that build on and are layered on top of RDF—RDFS and OWL. RDFS introduces classes which are critical to specifying a schema. A class is a set of individuals whose membership in the class is determined either by an explicit `rdf:type` statement—that the individual is of the type of the class, or whose membership can be inferred from other statements.

All built-in RDF language constructs are available for your use in Composer. These can be seen in the Classes View and the Properties View. Use the Classes View Preferences and the Properties View Preferences to configure which RDF elements should be visible.

### A.2 RDFS

RDFS is a schema language for RDF. RDFS is a rather small vocabulary. Among other things, it defines the following resources that are commonly used in ontology development:

- `rdfs:Class`
- `rdfs:subClassOf`
- `rdfs:subPropertyOf`
- `rdfs:domain`
- `rdfs:range`
- `rdfs:Resource`
RDFS also defines four annotation properties as well as a few other archaic elements such as `rdfs:Container` that are rarely used.

All resources in the RDFS vocabulary are available in TopBraid Composer. Its classes and properties are shown in the Classes View and Properties Views.

Unlike XML Schema, RDFS can not be used for validation. It is used to infer additional information based on the ontology schema and given (asserted) statements. An RDFS ontology can never be semantically invalid.

Syntactical errors are possible (and even likely) when creating RDF in a text editor. Using ontology development tools such as TopBraid Composer ensures that there are no syntactic errors.

RDFS and OWL have formally defined semantics. These semantics are defined in terms of inferences entailed by each RDFS or OWL statement. By entailed we mean inferences that result from an asserted statement in RDF. RDFS definitions are as follows:

- `rdfs:subClassOf`
  
  \[\text{If } x \text{ rdf:type } rdfs:Class1 \text{ and } rdfs:Class1 \text{ rdfs:subClassOf } rdfs:Class2, \text{then } x \text{ rdf:type } rdfs:Class2\]

- `rdfs:subPropertyOf`
  
  \[\text{If } x \text{ property1 } y \text{ and } \text{property1 rdfs:subpropertyOf property2, then } x \text{ property2 } y\]

- `rdfs:domain`
  
  \[\text{If } \text{Class1 rdfs:domain property1 and } x \text{ property1 } y, \text{then } x \text{ rdf:type Class1}\]

- `rdfs:range`
  
  \[\text{If } \text{Class2 rdfs:range property1 and } x \text{ property1 } y, \text{then } y \text{ rdf:type Class2}\]

**A.3 OWL**

The OWL Web Ontology Language is a W3C standard language for defining and instantiating Web Ontologies. The OWL vocabulary is defined on top of the RDFS vocabulary:

- `owl:Class` is a subclass of `rdfs:Class`.
- While RDFS properties are instances `rdf:Property`, OWL introduces `owl:ObjectProperty` and `owl:DatatypeProperty` which are both subclasses of `rdf:Property`. 
- OWL Object properties link two resources within the ontology. For example, the property `livesIn` might link John (an instance of the `Person` class) with London (an instance of the `City` class).
- OWL Datatype properties link a resource to an XML Schema Datatype value or an RDF literal. For example, the property `hasTelephoneNumber` might link John to the string literal “44-10-1233400”.

Additionally, OWL offers other ways to describe properties, including:

- Functional (any instance using this property can have only one distinct value for the property)
- Inverse Functional
- Transitive (if `partOf` is an owl:TransitiveProperty and `x partOf y and x partOf z`, then `x partOf z`)
- Symmetric (if `siblingOf` is an owl:SymmetricProperty and `x siblingOf y`, then `y siblingOf x`)

All OWL language constructs are available in TopBraid Composer. The screenshot below shows built-in OWL classes for properties.

![Available types:](image)

**Figure 38: Subclasses of rdf:Property**

In addition to global restrictions (property descriptions), OWL provides a vocabulary for creating local restrictions specific to a class. OWL is a rich language. Covering all its syntax is beyond the scope of this document.

OWL formal semantics specify how to derive logical consequences (inferences) of the statements made in the ontology, i.e. facts not literally present in the ontology, but entailed by the semantics. Composer provides support for generating OWL inferences. This is done using one of the built-in inference engines that can be configured for use with Composer.

### A.4 SPARQL

SPARQL is a W3C standard recommendation to query RDF graphs. It comes with a notation similar to the relational database query language SQL, but focuses on triple matching.

SPARQL provides five different query variations for different purposes.
**SELECT query** - Used to return results in a table format

**CONSTRUCT query** - Used to return RDF triples

**ASK query** - Used to provide a simple True/False result for a query

**DESCRIBE query** - Used to extract an RDF graph from the SPARQL endpoint, the contents of which is left to the endpoint to decide based on what the maintainer deems as useful information.

**UPDATE query** - Used to insert or delete RDF triples into a graph

Each of these query forms takes a WHERE block to restrict the query. Please check the SPARQL specification online for a precise language description:

http://www.w3.org/TR/rdf-sparql-query/

### A.5 SHACL

SHACL Shapes Constraint Language, a W3C standard language for validating RDF graphs against a set of conditions. These conditions are provided as shapes and other constructs expressed in the form of an RDF graph. Shapes may be used for a variety of purposes beside validation, including user interface building, code generation and data integration. As a modeling language, SHACL is an alternative to OWL. It is based on the closed world semantics.

SHACL can be considered SPIN 2.0 – results of standardization of SPIN. At the time of writing of this document, SHACL is still at the working draft status.