



TopBraid Ensemble Application Development and Reference Guide

release 3.4

November 28, 2010

Contents

Chapter 1: TopBraid Ensemble Overview.....	9
1.1 Conventions.....	10
1.2 Getting Started with TopBraid Ensemble.....	10
1.3 Deploying an Application to TBL Enterprise Server.....	10
1.4 TBE Components and Applications.....	11
1.5 Displaying Labels in TopBraid Ensemble.....	13
1.6 TopBraid Suite Dictionaries and Preferred Labels.....	13
1.7 Autocomplete in TopBraid Ensemble.....	15
 Chapter 2: Developing a TBE Application.....	 17
2.1 Creating a TBE Application.....	18
2.2 Selecting and Configuring Components.....	20
2.2.1 Components and Layout.....	21
2.2.2 Event Wiring.....	25
2.3 Creating Popups.....	27
2.4 Pages.....	28
2.5 Modifying and Resaving an Application.....	30
2.6 Configuring a Simple Application: SKOS Editor.....	31
2.7 Configuring Custom Forms and Grids.....	35
2.8 Configuring Startup Events.....	36
2.9 Saving TBE Applications.....	36
 Chapter 3: TopBraid Ensemble User Interface Components.....	 39
3.1 Event Button.....	40
3.1.1 Attributes and Events.....	40
Attributes.....	40
Post events.....	41
Listen events.....	41
3.2 Info Box.....	42
3.2.1 Attributes and Events.....	42
Attributes.....	42
Listen events.....	43
3.3 SPARQL Editor.....	43
3.3.1 Attributes and Events.....	43
Post events.....	43
Listen events.....	43
3.4 Basket.....	44
3.4.1 Attributes and Events.....	44
Attributes.....	44
Post events.....	44
Listen events.....	44
Buttons.....	44

- Drag events..... 45
 - Drop events..... 45
- 3.5 Form..... 45
 - 3.5.1 Attributes and Events..... 46
 - Attributes..... 46
 - Post events..... 46
 - Listen events..... 47
 - Buttons..... 48
 - Drop events..... 48
- 3.6 History Form..... 48
 - 3.6.1 Attributes and Events..... 49
 - Post events..... 49
 - Listen events..... 49
 - Buttons..... 50
- 3.7 Graph Editor and Query..... 50
 - 3.7.1 Attributes and Events..... 51
 - Attributes..... 51
 - Post events..... 51
 - Drop events..... 52
- 3.8 Results Grid..... 52
 - 3.8.1 Attributes and Events..... 52
 - Attributes..... 52
 - Post events..... 55
 - Listen events..... 55
 - Buttons..... 56
 - Drag events..... 57
 - Drop events..... 57
- 3.9 Tree..... 57
 - 3.9.1 Attributes and Events..... 58
 - Attributes..... 58
 - Post events..... 59
 - Listen events..... 59
 - Buttons..... 59
 - Drag events..... 60
 - Drop events..... 60
- 3.10 Search Form..... 60
 - 3.10.1 Attributes and Events..... 61
 - Attributes..... 61
 - Post events..... 61
 - Listen events..... 62
 - Buttons..... 62
 - Drop events..... 62
- 3.11 White Space..... 62
 - 3.11.1 Attributes and Events..... 62
- 3.12 Yahoo Map..... 63
 - 3.12.1 Attributes and Events..... 64
 - Attributes..... 64
 - Post events..... 64
 - Listen events..... 64
- 3.13 Search Field..... 65

3.13.1 Attributes and Events.....	65
Attributes.....	65
Post events.....	65
Listen events.....	65
Chapter 4: TopBraid Ensemble Event Rules.....	93
4.1 Deep Linking Component.....	94
4.2 SPARQL Rule.....	94
4.3 SPARQLMotion Relay.....	95
Chapter 5: TBE Architecture.....	97
5.1 Customizing Component Interactions: the TBE Event Bus (DAG).....	98
Chapter 6: Integrating SPARQLMotion Scripts with TBE Applications.....	101
6.1 Defining SPARQLMotion Scripts.....	102
6.2 Using Ensemble Events to Invoke SPARQLMotion Scripts.....	106
6.3 Invoking SPARQLMotion Scripts on Ensemble Components.....	109
6.4 Using the SPARQLMotion Debugger in TBL Personal Server.....	111
Chapter 7: Building Custom Flex Components.....	113
Chapter 8: Component Configuration Fields.....	115
8.1 Common Events.....	116
8.1.1 All Components.....	116
Listen events.....	116
8.2 User Interface.....	116
8.2.1 Basket.....	116
Attributes.....	116
Post events.....	116
Listen events.....	116
Buttons.....	117
Drag events.....	117
Drop events.....	117
8.2.2 Event Button.....	117
Attributes.....	117
Post events.....	118
Listen events.....	118
8.2.3 Search Form.....	119
Attributes.....	119
Post events.....	119
Listen events.....	119
Buttons.....	120
Drop events.....	120
8.2.4 History Form.....	120
Post events.....	120
Listen events.....	120

Buttons.....	120
8.2.5 Form.....	120
Attributes.....	120
Post events.....	121
Listen events.....	122
Buttons.....	122
Drop events.....	123
8.2.6 Yahoo Map.....	123
Attributes.....	123
Post events.....	123
Listen events.....	123
8.2.7 Results Grid.....	124
Attributes.....	124
Post events.....	126
Listen events.....	127
Buttons.....	128
Drag events.....	128
Drop events.....	128
8.2.8 Info Box.....	128
Attributes.....	128
Listen events.....	129
8.2.9 Search Field.....	129
Attributes.....	129
Post events.....	129
Listen events.....	129
8.2.10 SPARQL Editor.....	130
Post events.....	130
Listen events.....	130
8.2.11 TopBraid Ensemble.....	130
Attributes.....	130
Post events.....	132
Listen events.....	132
Buttons.....	132
8.2.12 Tree.....	133
Attributes.....	133
Post events.....	134
Listen events.....	134
Buttons.....	135
Drag events.....	135
Drop events.....	135
8.2.13 Graph Editor and Query.....	135
Attributes.....	135
Post events.....	135
Drop events.....	136
8.2.14 White Space.....	136
8.3 Pages and Popups.....	136
8.3.1 TopBraid Page.....	136
Attributes.....	136
Post events.....	136
8.3.2 Popup Window.....	136

Attributes.....	136
Post events.....	137
Listen events.....	137
8.4 Event Rules.....	138
8.4.1 Relay.....	138
Attributes.....	138
Post events.....	138
Listen events.....	138
8.4.2 Deep Linking Component.....	138
Attributes.....	138
8.4.3 SPARQL Rule.....	138
Attributes.....	138
8.4.4 SPARQLMotion Relay.....	139
Attributes.....	139
8.5 Listen Events Summary.....	139
8.6 Post Events Summary.....	142
8.7 Drag Events Summary.....	146
8.8 Drop Events Summary.....	146

Chapter 1

TopBraid Ensemble Overview

Topics:

- [Conventions](#)
- [Getting Started with TopBraid Ensemble](#)
- [Deploying an Application to TBL Enterprise Server](#)
- [TBE Components and Applications](#)
- [Displaying Labels in TopBraid Ensemble](#)
- [TopBraid Suite Dictionaries and Preferred Labels](#)
- [Autocomplete in TopBraid Ensemble](#)

TopBraid Ensemble (TBE) is a Flex-based web application for assembling and displaying TopBraid applications. The user interface runs on web browsers with Adobe Flash installed. When developing applications, the data behind the application is hosted by the TopBraid Live Personal Server included with TopBraid Composer Maestro Edition (TBC-ME); applications can then be deployed for multi-user use on a server running the TopBraid Live Enterprise Server product. Because TBE applications run on Flash, they will have a consistent user interface and appearance in all browsers, making applications easier to deploy to a wide variety of users.

TBE is based on a model-view-controller (MVC) architecture that lets developers quickly configure applications around semantic web models. You can develop web applications by:

- Assembling pre-packaged graphical user interface components such as search forms and tree-based displays into desired configurations that let the user view or edit data
- Adding custom business actions through SPARQLMotion scripts
- Building and using new custom components

Once the application developer designs forms and event logic around a data model and deploys this TBE application on a server running the Enterprise edition of TopBraid Live, the developer's end users can use this application with any web browser that supports Adobe Flash.

This guide provides an overview of working with the TBE to assemble applications.

The TopBraid Live Administration Guide provides additional information that will be valuable to people creating TBE applications, particularly in the book's Administration section, which covers the management of application-related files once they are deployed on a TBL Enterprise Server.

1.1 Conventions

Names for user interface widgets and menu options are presented in a style **like this**.

Where exercises require information to be typed into TBC a monospaced font is used **like this**.

Exercises and required tutorial steps are presented like this:

1. Execute the first step.
2. Then, execute the second step.
3. Here is the third step to carry out.



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.

1.2 Getting Started with TopBraid Ensemble

There is no setup necessary for starting TopBraid Ensemble, other than installing and running either TopBraid Composer-Maestro Edition (Personal Server) or TopBraid Live (Enterprise Server). Unless otherwise stated, this document assumes the reader has a current version of TBC-ME running. To open any data graph (file in workspace) in the TBC-ME workspace do the following:

- In a browser, enter the URL: `http://localhost:8083/tbl`. This will open the TBL Console (Personal Server).
- Click on "Default Application". A page will appear that lists all projects in the TBC-ME server running in the background.
- Click on the "Show" button and choose a link to open one of the data graphs.

When opening a data graphs in TBL Personal server, temporary files will open in TBC-ME. These files represent session data used by TBL. The files should be closed through the TopBraid Live Server Administration page by choosing "Clear All Sessions". Later sections of this document will provide details about these files and session data.

The Default Application is designed to display any valid ontology in the workspace. Some of the available components and event configuration have been used to allow the user to view and navigate the ontology. The remainder of this document addresses how Ensemble applications can be created through Ensemble configurations and server-side scripting.

1.3 Deploying an Application to TBL Enterprise Server

TopBraid Ensemble applications are developed on Composer (TBC-ME), taking full advantage of TopBraid Composer editing and design features for ontology editing, SPARQL support, inference, SPARQLMotion scripts,

and SPIN. TBC-ME includes the TBL Personal Server to develop and test Ensemble applications on a local machine. Much of the TopBraid Live infrastructure is common between Personal and Enterprise servers, ensuring that applications that work on Personal Server will work in the same manner on Enterprise Server. Once applications are developed, the project containing the Ensemble application definition and associated files are deployed to a TBL Enterprise Server for access by remote users. There are two ways to deploy an application to a TBL Enterprise Server:

- In TBC-ME, choose the project to be deployed in the Navigator view. Choose "Export... > Deploy Project To TopBraid Live Server". Enter the name of the server and enter the user name and password. The contents of the project will be uploaded to the server and a confirmation page will open in TBC-ME. The project will be ready for immediate use, including registrations of scripts, etc.
- Use a standard zip archive program to archive the project into a zip file. Open the console page for the destination TBL Enterprise Server in a browser. Choose "Project Upload", specify the name of zip file created for the project and "submit". The zip file will be uploaded to the server and deployed as a project for immediate use.

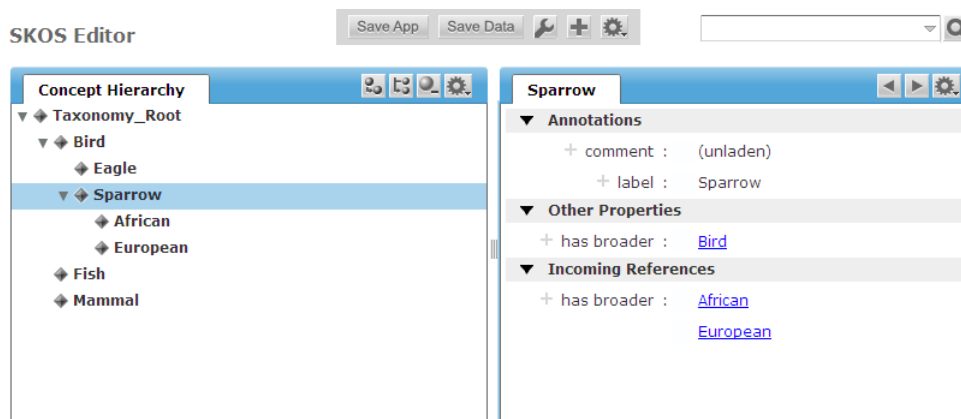
One-step deployment is a key feature of the fully integrated TopBraid Suite. There are few difference between Enterprise and Server installations, ensuring easy transition from development and testing to production. One minor difference is that TBL Enterprise Server is less tolerant of base URI conflicts than Composer. When deploying, error message will appear if the base URI of a file is the same as a file in the Live workspace or the project itself has base URI conflicts. For more on base URI conflicts, see [What does "Opening non-primary file" mean?](#)

Deploying to a TBL Enterprise Server installation may include additional considerations for multi-user applications, such as access control and authentication. These are described in detail in the TopBraid Live Installation Guide, available with purchase of TopBraid Live Enterprise Server (see http://www.topquadrant.com/products/TB_Live.html and contact sales@topquadrant.com for more information).

1.4 TBE Components and Applications

The user interface component types offered in TBS include trees, grids, forms and maps. A developer can arrange these across multiple pages that function like a tabbed interface to reduce the clutter than can result from squeezing too much onto a single screen.

For example, the following shows a form from a TBE application with a tree component and a form component, arranged to let an end user edit the entries stored in a simple thesaurus:



Each UI component can:

- Contain one or more widgets that can display or provide an edit mode for information from the connected information source
- Include its own menu actions and icons

- Broadcast events to other components and listen for and react to events broadcast by other components

TBE also provides non-UI, or invisible components, which work behind the scenes to perform data transformation and to relay events.



The information as associated with each component is stored in a ‘manifest file’ for each Component type.

A developer designs one or more input screens with TBE components and specifies the data graph that it will work with. As we’ll see, these applications can call SPARQLMotion scripts, combining sophisticated back-end triples processing with quick development of application user interfaces to create powerful, intuitive semantic web applications.

Creating an application consists of four basic steps:

1. Select whether you want to begin with a New Application or a Default Application
2. Identify the data to use with your application
3. Select and configure the components to go on your screens
4. Test your application

Starting with a New Application means starting with a blank form and then adding and configuring the components that you need. This is especially useful for experienced TBE developers creating simple applications. New TBE developers often find it easier to begin with a Default Application, which includes several components arranged in a default layout that you can modify by deleting, adding, moving, and reconfiguring the components.

The following shows a Default Application before any modifications have been made, with `kennedyfamily.owl` specified as the data to use with it:

Name	Comment	Year of birth	Year of death
Rory Kennedy		1968	
Alfred Tucker		1967	
Victoria Lawford		1958	
Robin Lawford		1961	
Carolyn Bessette		1966	1999

A configured TBE application specifies what components are shown when it is launched, how components are arranged, and how they interact with each other. Each application also defines what data to use with it. Some applications, such as the default apps and the new app, are configured to work with any data. Other applications are configured to work with specific datasets.

When creating a TBE application, components can be rearranged, resized, removed and added. For example, you can have two Results Grid components on the screen instead of one, like the default applications have. You can also customize the menu options for each component and the way the components interact with each other.

TopBraid Ensemble's Default configurations are general TBE applications included in your installation. These configurations include most of the built-in components, laid out and wired for interaction with each other so that when an event such as a resource selection happens in one component, information displayed in some of the other components changes. For example, when you select a class in the tree, the members of the class appear in the grid, and when you select a resource in the grid, it populates the form with information about that resource.

You can learn a lot about the potential power of these forms by studying how the components in the default configurations generate and listen to the events that make this kind of component interaction possible.

1.5 Displaying Labels in TopBraid Ensemble

TopBraid Ensemble uses labels to display names of resources. Ensemble displays all resources using a string label property, which avoids problems with characters that are not legal in URI fragments. Names are displayed similar to TopBraid Composer with the "Labels Display Mode" toggled to "Human-readable labels". For more information, see Composer's Help files at Help > User Interface Overview, Labels Display Mode.

A resource's `rdfs:label` property and subproperties of `rdfs:label` are used to display labels in Ensemble. If a `rdfs:label` property is not defined for a resource the local name (URI fragment) of the resource will be displayed. For example, if the resource's URI is "`http://topbraid.org/examples/kennedys#GeorgetownUniversity`", the string "`GeorgetownUniversity`" will be displayed as the resource's label.

Labels are rendered by the TopBraid Suite Dictionary, as described in the next section.

1.6 TopBraid Suite Dictionaries and Preferred Labels

Following W3C recommendations, `rdfs:label` is used as the standard string representation for displaying a resource using TopBraid Live. The TopBraid Suite dictionary will find the preferred label for each resource in a data graph and store the labels in a Lucene index for efficient lookup and label processing. TopBraid Ensemble uses the dictionary to:

1. find a preferred label for a resource to be displayed in the user interface
2. facilitate quick display of resources based on their labels (autocomplete functionality).

The labels contained in the dictionary are used whenever a resource is displayed in a TopBraid Ensemble component. There are a number of issues to consider when finding preferred labels for resources. It is not uncommon for resources to have multiple labels. Furthermore, different ontologies may use different properties for the labels. For example, RDFS defines `rdfs:label` for this purpose. SKOS offers several label properties, all of which are defined as subproperties of `rdfs:label`. Ontology developers may have their own properties they want to be used as a label, and can extend the TBS dictionary to use these by declaring the property to be a subproperty of `rdfs:label`. By considering `rdfs:label` and all subproperties of `rdfs:label` as preferred labels, TopBraid Suite has flexible support for defining how resource names are displayed in user interfaces.

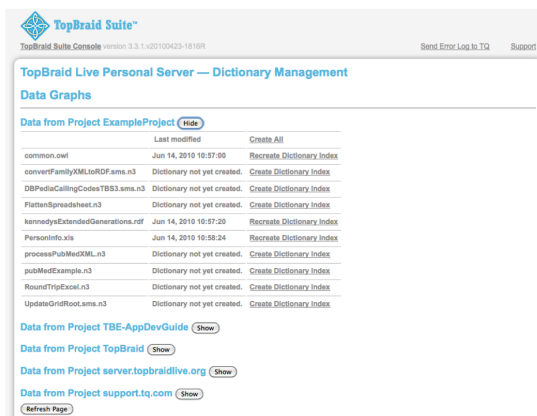
To find the *preferred label* the dictionary considers, for each resource, properties that may hold label values as well as well as the local language preferences. There are four parts to finding preferred labels for storage in the dictionary, which are applied in the following order:

1. Property precedence:
 - a. `skos:prefLabel` and all subproperties of `skos:prefLabel` have the first precedence.
 - b. `rdfs:label` and all subproperties of `rdfs:label` have the second precedence.

- c. Within a tree of subproperties, precedence is based on the level - children take precedence over parents.
2. Other label considerations:
 - a. If no labels are found, resource's localname is used (the fragment part of the resource's URI).
 - b. skos:altLabel is always a non-preferred label, making it searchable on autocomplete, but never shown as the label for a resource in the UI components.
3. Language tag precedence:
 - a. Labels with Language Tag that matches the browser's language preference.
 - b. Labels without a Language Tag.
 - c. Labels with an English language tag (de facto universal language).
 - d. If labels do not fall within the cases above, use the Property precedence described above.
4. Label conflicts:
 - a. When displaying names for resources in UI components, label conflicts are allowed. I.e. two resources with the label "Mary Kennedy" will appear using the same string, "Mary Kennedy" for the name in a Tree component. This is in contrast with TopBraid Composer's "Labels Display Mode" toggled to "Human-readable labels", where all labels are uniquely defined, including resolving label conflicts (as done in Autocomplete).
 - b. When displaying names in an Autocomplete context, unique labels are rendered for each resource, as described in [Autocomplete in TopBraid Ensemble](#).

Creating and re-creating dictionaries. For each data graph (base URI) there is one dictionary implemented as a Lucene index. The dictionary for a data graph includes dictionaries for all local imported graphs. Web imports are not stored in the index. Labels for web imports are resolved 'on the fly' when the resource is selected.

TBS console provides a Dictionary Management page under TopBraid Live Server Administration listing all dictionaries and the timestamp when they were created/updated. See the screenshot below. The page also offers an administration function for creating and rebuilding dictionaries. This functionality is available in both TBL Enterprise and Personal Servers. Dictionaries are stored in a folder named .TBIndices for each project in the workspace. Dictionaries can be removed by deleting this folder.



Dictionaries are created when a project is uploaded to the TopBraid Live server. If a dictionary does not exist for a given data graph, its creation will be triggered when a user starts a TBE application working with the graph. Dictionaries can also be created manually in the Dictionary Management page.

When uploading large data sets to TBL some time should be allowed for dictionary creation. TBE applications will work during the initial dictionary building process, but performance may suffer. A similar best practice applies to

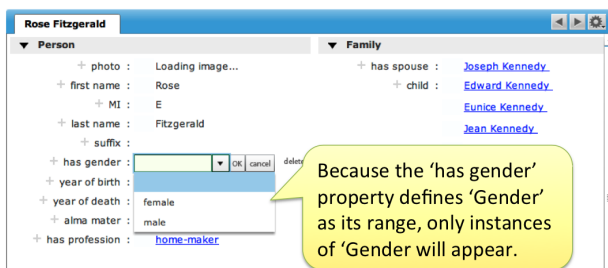
working with TBE-ME. If you have a large data graph, prior to using it with TBE create a dictionary by navigating to the Dictionary Management page and clicking on either a 'Create Dictionary Index' link next to the data graph or on a 'Create All' link for the entire project.

When users perform edit operations using TBE applications, dictionaries are incrementally updated to reflect added and deleted information. The PerformUpdate module in SPARQLMotion scripts will also incrementally update the dictionary for each resource inserted into or deleted from the <http://tb-session> session. Note that a PerformUpdate module adding a large number of triples will temporarily impact TBL performance.

1.7 Autocomplete in TopBraid Ensemble

When adding or editing a resource value for a triple, TopBraid Suite will show suggestions of existing valid resources for the triple based on the characters typed into a text area. The labels listed as suggestions are generated by the TBS dictionary.

RDFS restrictions are applied when rendering the list of autocomplete text choices. If the triple's property has a range, autocomplete suggestions will be limited to resources from that type. For example, in the Kennedy ontology (in TBC-ME workspace under TopBraid/Examples/kennedys.rdf), when entering or modifying a person's "child", TBE will only show resources that are of the type "Person" because the rdfs:range of the "child" property is "Person". The following screen show show an example where the range of "has gender" is instance of the class "Gender"



Autocomplete matches can be of 3 types:

1. If an autocomplete result matches the resource's preferred label, then the preferred label is shown as rendered by the TBs dictionary.
2. Autocomplete will search all labels in the dictionary for a match. If the matching label is not the preferred label for the resource, the match will display the preferred label, appended with the label that matched the characters entered in the autocomplete (notice that the preferred label might not match those characters). For example, if the resource <http://fifa.com/worldcup> has the following defined labels:

```
http://fifa.com/worldcup rdfs:label "World Cup"
http://fifa.com/worldcup skos:prefLabel "FIFA Copa do Mundo"
```

...the label "FIFA Copa do Mundo" will be the preferred label. If a user types "world", autocomplete will display the match as "FIFA Copa do Mundo (World Cup)".

3. When two or more resource labels resolve to the same string, qnames or URIs of the resources are appended to the end of the label in parenthesis. For example, suppose two resources exists in a model with the same label:

```
Mary Kennedy (kennedys:MaryCKennedy)
Mary Kennedy (kennedys:MaryKKennedy)
```

These resources would appear in Ensemble as:

```
"Mary Kennedy [http://topbraid.org/examples/kennedys#MaryCKennedy]"
```

```
"Mary Kennedy [http://topbraid.org/examples/kennedys#MaryKKennedy]"
```

If a prefix is defined for the namespace, the qname (namespace prefix + local name) will be displayed. For example, if the prefix "kennedys:" is defined for "http://topbraid.org/examples/kennedys#", the resource will be displayed as:

```
"Mary Kennedy [kennedys:MaryCKennedy]"
```

```
"Mary Kennedy [kennedys:MaryKKennedy]"
```

Chapter

2

Developing a TBE Application

Topics:

- [Creating a TBE Application](#)
- [Selecting and Configuring Components](#)
- [Creating Popups](#)
- [Pages](#)
- [Modifying and Resaving an Application](#)
- [Configuring a Simple Application: SKOS Editor](#)
- [Configuring Custom Forms and Grids](#)
- [Configuring Startup Events](#)
- [Saving TBE Applications](#)

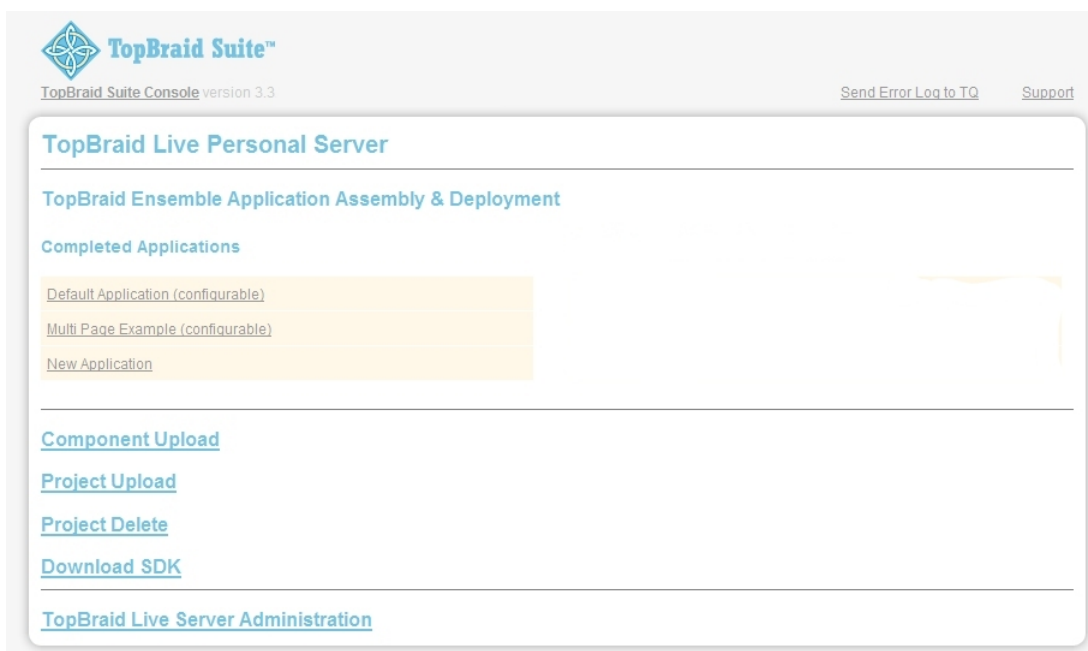
The following sections provide a brief walkthrough of the tools and process for developing an application with TBE.

2.1 Creating a TBE Application

To get started with developing and configuring a TBE application, open a browser window with either of the following URLs:

- <http://localhost:8083/tbl> when running the TopBraid Live Personal Server included with TBC Maestro Edition
- <http://<HOSTNAME>/tbl/server/tbl> when with TBL Enterprise Edition

This opens the TopBraid Suite Console. See the TopBraid Suite Console chapter for more on the menu choices available on this screen.

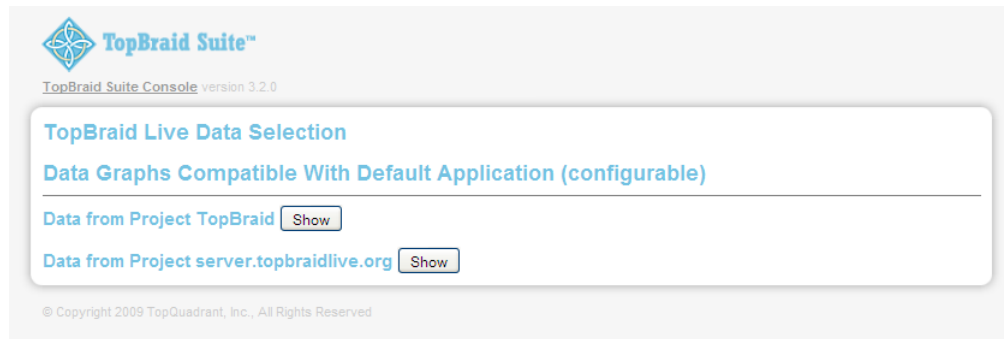


TopBraid Suite Console

Start by clicking on one of the available applications. This will provide the starting point for your new application. A number of factors may be considered when deciding which application to start with:

- When using New Application as a starting point, your first step will be to add a component. This may be the best starting point if your application has only one or two components
- If your application will involve multiple components, using one of the Default TBE configurations as a starting point offers several advantages, because a copy of each of the available components is already included. If you need most of these components, starting with a Default Application may save some time. The TBE components in each Default Application are already wired together. We recommend examining the Default Applications to better understand how to connect components.

After you select one of the applications, the next screen shows the available projects in the workspace. By clicking on a "Show" button next to the project name, you will see all available data graphs (ontologies and RDF data sets) in each project that are compatible with the selected application.



Selecting the data graph for an application

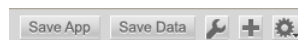
All data graphs are compatible with both the general TBE application and with the blank application.



You can select one data graph for your application to work with. If you want your application to work with multiple ontologies or datasets, either merge all of your data and ontologies into a single model and select it from the list or create a model that imports all the models and data and select that from the list.

To select one of the data graphs displayed with a Show button, click on its name. Selecting a data graph will launch your new application. Note that the data graph your application works with will be automatically loaded into the TopBraid Live server, whether you are using the Personal edition included with TBC-ME or the Enterprise edition purchased separately.

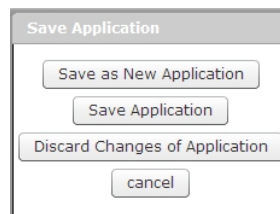
Once an application is opened, note the configuration menu bar on the top of the screen.



The icons from the left to right let you do the following:

1. Save an application (the combination of components, with their configurations, currently on your screen)
2. Save the data from your model—for example, if your application's end user edited data displayed on a form or grid component on your application's screen
3. Configuring details about the application and its current components such as their labels and their events that they listen for and post
4. Add new components, pages, event rules, and popups to the application
5. Show the SPARQLMotion scripts for the selected resource

When you click the Save App button, TBE offers you these choices:



If you select Save As New Application, TBE asks for the following additional information:

Save Application As

Optional: Select graph(s) which will be required for this application.

http://topbraid.org/examples/kennedys
 http://www.topbraidcomposer.org/owl/2006/07/tbcgeo.owl
 http://www.w3.org/2003/01/geo/wgs84_pos
 http://purl.org/dc/elements/1.1/

Please enter application name and base URI.

Application Name:

Application Base URI:

Save To:

☐ Overwrite?

TBE will not allow you to save a modified version of one of the default applications, so enter a new Application Name and Application Base URI for your application.



If you try save an application with the same name and Base URI as an existing one, you will get an error message if you don't check the Overwrite check box.

If you leave all the choices under "Select graph(s) which will be required for this application" unselected, then when you next open this application, it will ask which graphs to use, like it did when you first started creating the application. Otherwise, it will assume that the selected graphs are the ones to use.



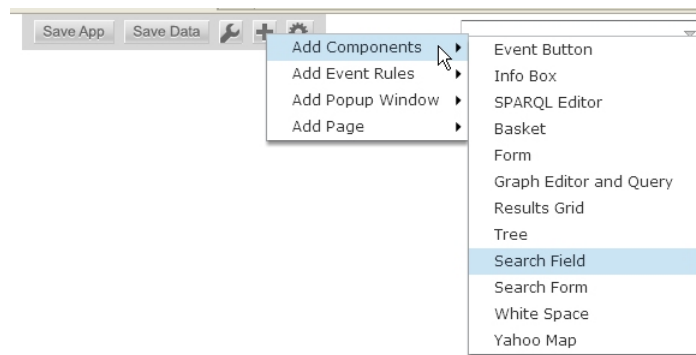
The base URI is different from the URL used to access the application; we recommend including "(configurable)" in the application name and URI until you are ready to deploy.

In the "Save to" field, you can pick a project in which to save your application. Saving it to the same project as the data files and any SPARQLMotion files used by the application will make it easier to package up the complete application for deployment on a TopBraid Live Enterprise server.


Newly created applications will appear in the TopBraid Suite Administration Console under the "Applications Under Development" section on the right side of the TopBraid Suite Console after they have been saved using Save App. Newly created applications are stored with a URL of the form `http://server.topbraidlive.org/dynamic/user-applications/<Chosen App Name>.n3`

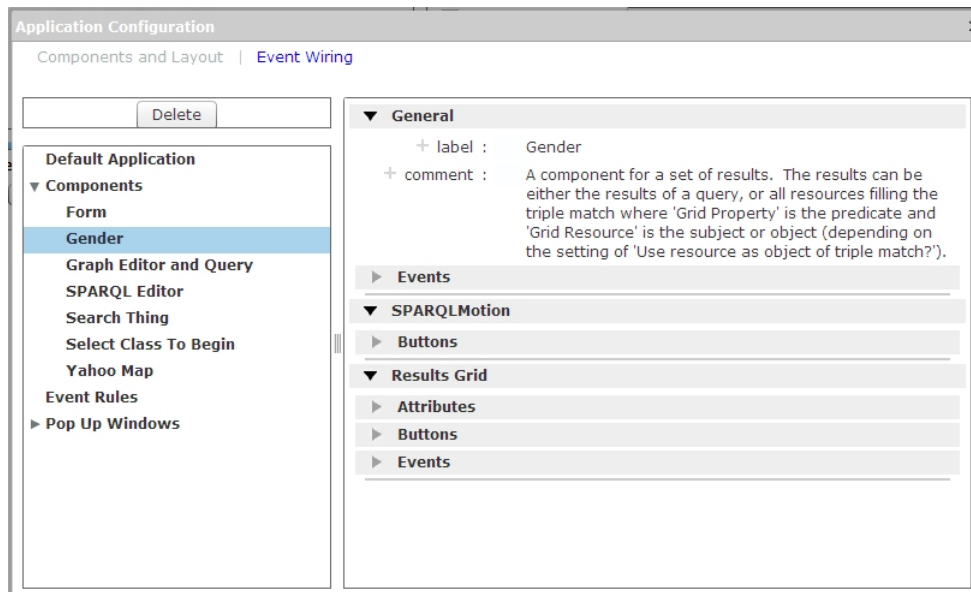
2.2 Selecting and Configuring Components

To add a new component to the screen select Add Components from the + icon. This displays a cascade menu listing available components. This includes prepackaged components as well as any custom component uploaded using the Administration Console.



When a screen contains two or more components, you can move them by dragging and dropping their tabs. You can toggle the display of a panel's tab with a Control-click next to the tab. Components can be resized by dragging the little dividers on each side of them, and these two can be hidden by Control-clicking on them.

Click the "Show/Hide Configuration"  icon to open a configuration console for the application as shown in the following:



Application Configuration Console

The console offers a choice of two screens:

- **Components and Layout** lets you configure the attributes of the application itself, the application's components, the events that each component listens for and posts, and the pop up windows.
- **Event Wiring** lets you manage the events themselves.

The following describes these two screens in further detail.

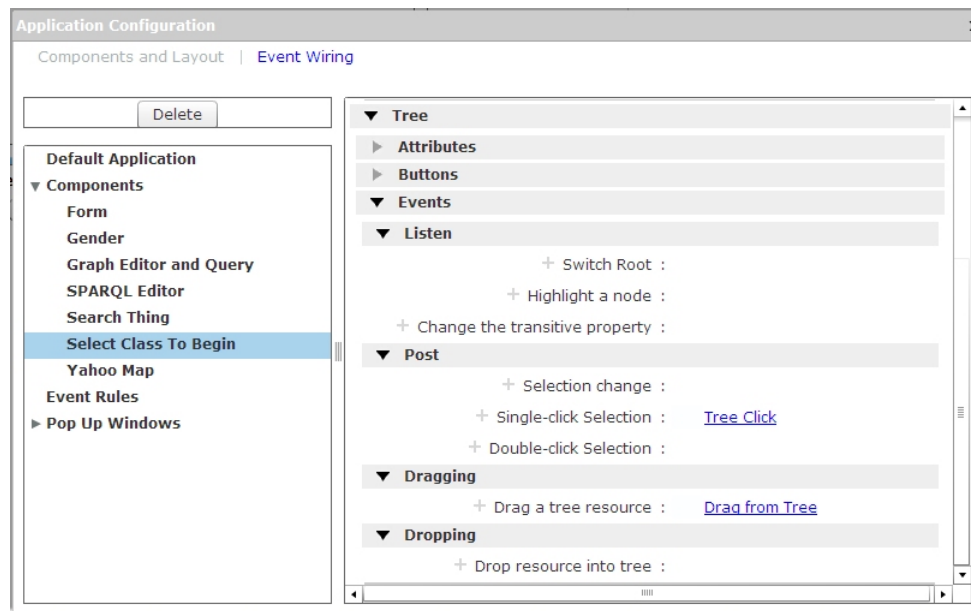
2.2.1 Components and Layout

The Navigator on the left side of the Components and Layout screen provides access to the configuration forms for the application and the individual components with a tree-based display. Configuration forms are displayed on the right.

Once you select a component on the left, the Delete button in the upper-left becomes active, if you want to delete the component. If you click this, a message box will ask you to confirm.

You'll also see the configuration form for that component on the right, with two or three main sections, depending on the nature of the component:

- A General section contains options common to all components, such as the label, or name, of the component. It also includes an Attributes subsection that lets you set attributes of the component such as whether it can be closed by your application's users.
- A SPARQLMotion section lets you configure the buttons that drive the component's use of SPARQLMotion scripts, if that component can use these scripts.
- A component-specific section such as Results Grid in the illustration above contains options specific to the use of the particular component. This section has three categories:
 - **Attributes** configures attributes of a component. Each component will have different attributes. For example, attributes of the Tree component let you configure the property used to display the tree, the root node of the tree, and more.
 - **Buttons** configures pre-defined buttons appearing in the top-right of the component.
 - **Events** configures events. A component can be configured to listen to an event or post an event. Events are resources that are defined to carry information across an event bus. Dragging and Dropping events are also available. The following illustration shows an example:

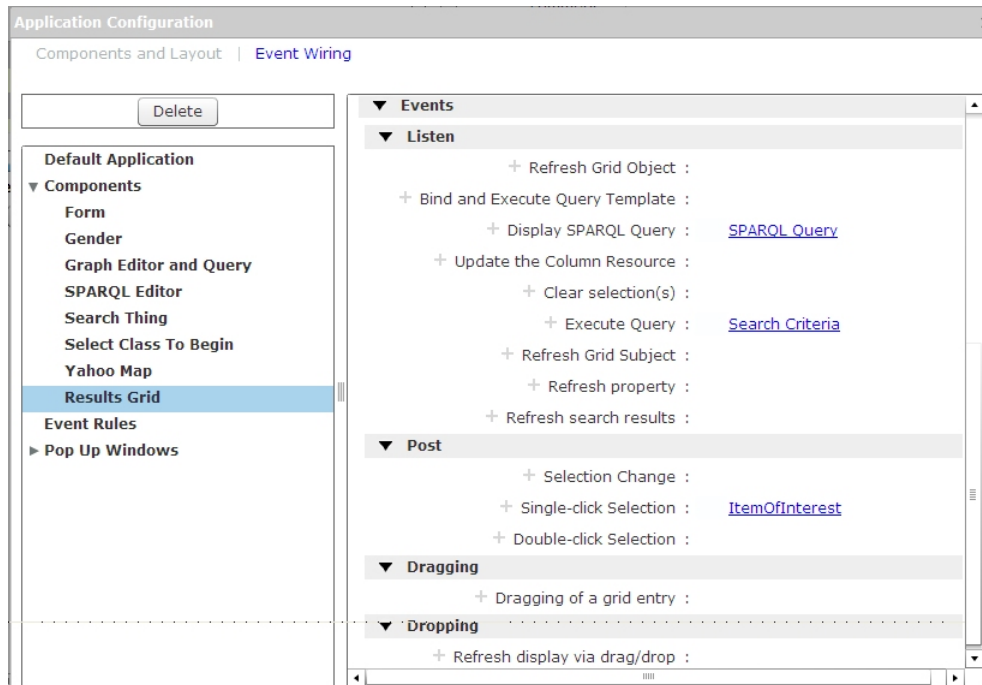


Event Section of a Component Configuration Form for a Tree



This screenshot shows a tree component that currently has none of its Listen or Dropping events set, one Post event set, and one Dragging event set.

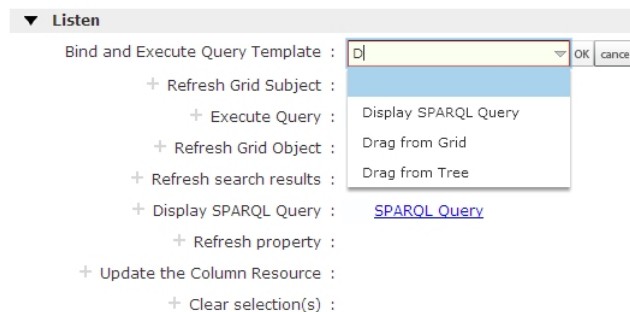
Components listen for posted events. For example, the next illustration shows how the Results Grid listens for two different events. When it hears a ClassOfInterest event, it will refresh its display with the associated data. We saw above that the Tree component posts the Tree Click event when one of its nodes receives a single click, but other components can send the ClassOfInterest event to be picked up by the Results Grid component as well.

This Results Grid is also configured to post an ItemOfInterest event when the application's end user clicks one of its items. As you can see from the other available Post items, you can also configure the Results Grid it to inform other components when one of its items is double-clicked or when the selection changes.

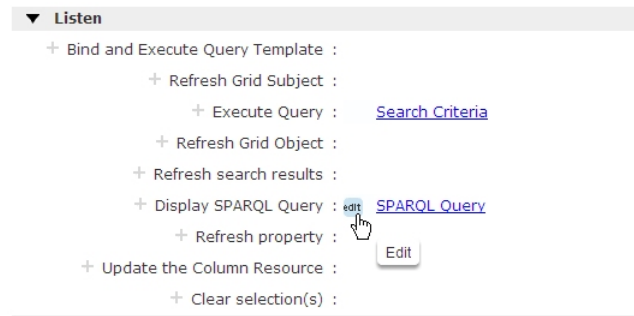


Event Section of a Component Configuration Form for a Results Grid

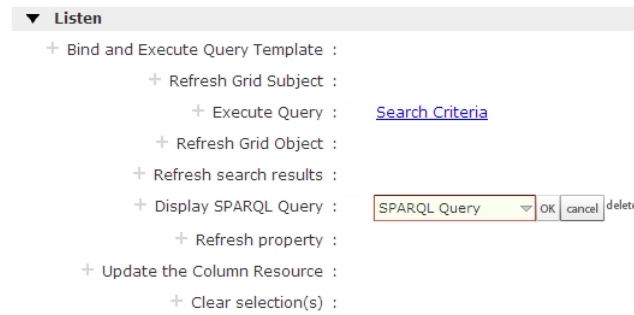
To identify a new event to listen for, click the "Add new row"  icon accessible from the menu to the left of the field label. This displays a context menu offering the choice of defining a new event with "Create New" or picking an existing event with "Add existing entry". This creates a new resource. To select an existing event, click the "Add existing entry"  icon. This creates a new row. When you start typing in the row, the TBE autocomplete feature provide a drop down list of the available events to choose from:



To edit an existing entry, mouse over the area directly to the left of the field. An edit menu will appear as shown here:



When you click it, the current value will be moved into a field where you can edit it by hand or use the drop down list to select an existing item name, as shown here:

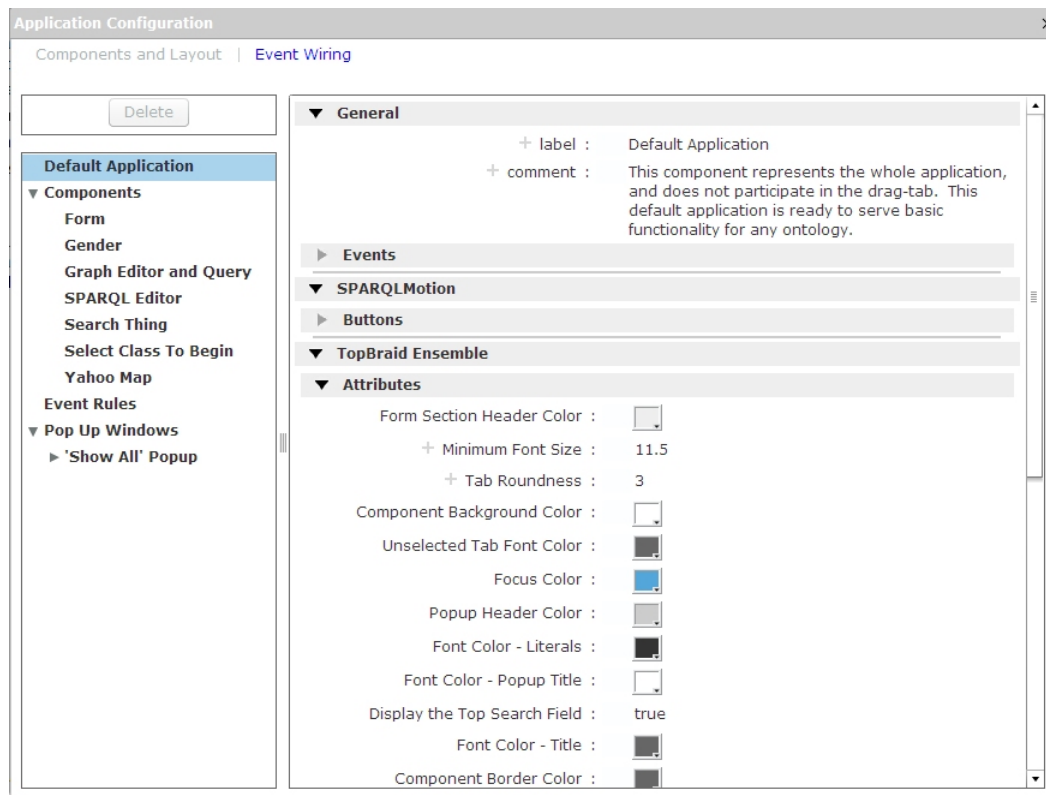


When you have finished editing the value or made your selection from the list, click the OK button. Or, to delete the entry so that no event is assigned to take this action (in this case, to refresh the display), click the delete button to the right of the value.

In addition to the components that have user interfaces, such as trees and forms, TBE also provides invisible relay components. Relays are used to transform events by running queries or SPARQLMotion scripts.

Relay components, which define how an event can be processed and relayed to another event, are shown under Event Rules in the Application Configuration Console Navigator. To create a new relay, select Add Event Rules and then SPARQL Rule from the + menu. When you next see the Application Configuration screen, your new rule will be at the end of the Event Rules list on the navigator, ready for you to configure it.

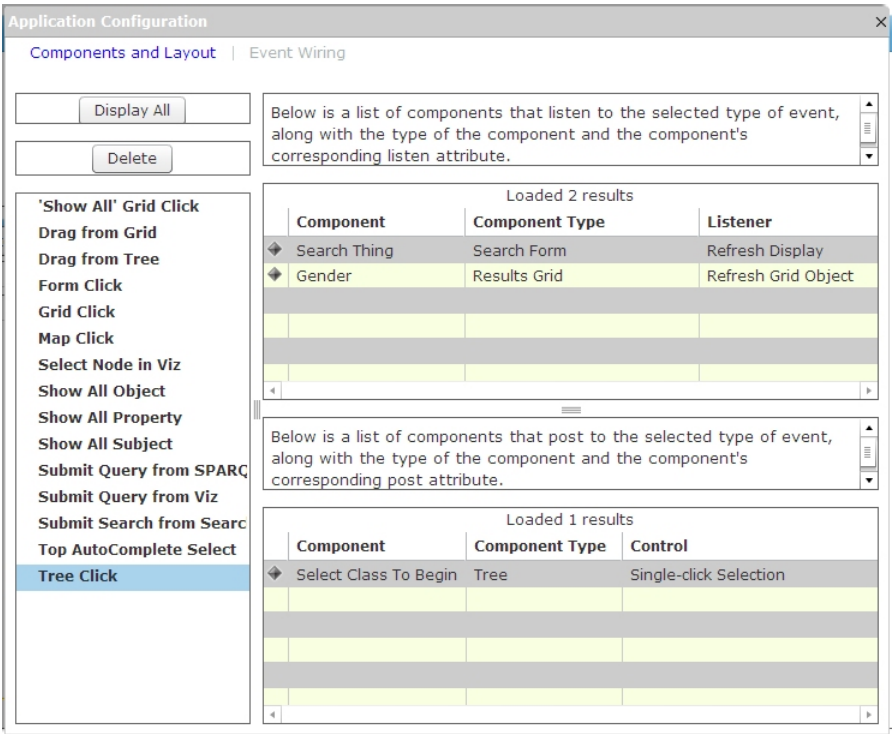
To configure the look and feel of the application, select the Application itself in the Navigator and use the Attribute subsection under TopBraid Ensemble. Many options are available. Some are shown in the next screenshot.



Application Configuration Form

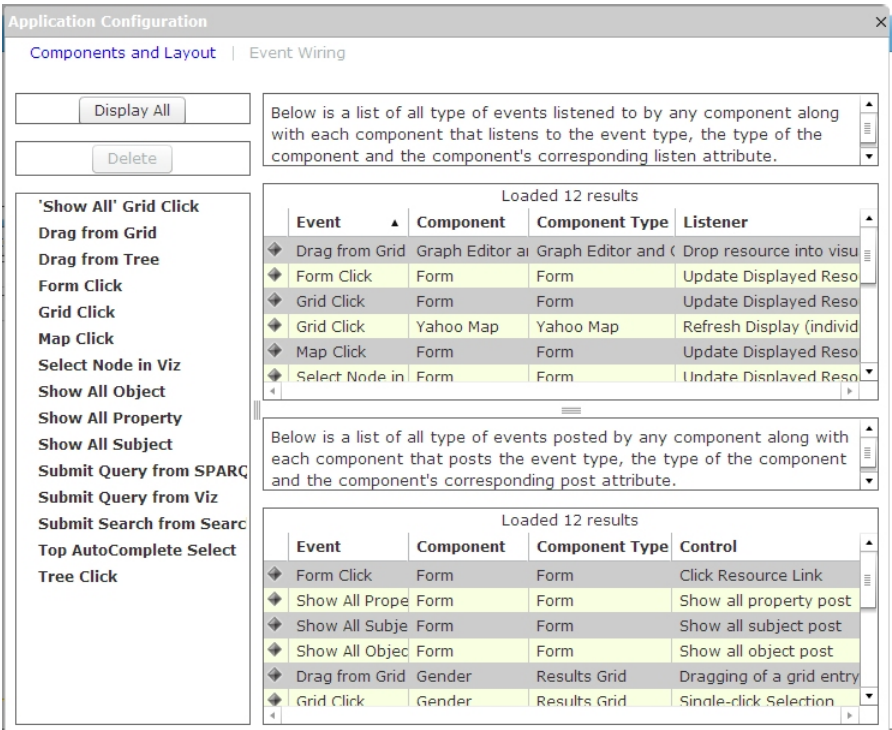
2.2.2 Event Wiring

As a TBE application grows in complexity, you may lose track of which events are being posted or listened for by the various components. The Application Configuration Console's Event Wiring screen makes it easy to manage these events. While the Components and Layout screen lets you configure which events each screen component posts and listens for, the Event Wiring screen lets you see, for each event, which components (and which of the component's actions) are using it. For example, the following illustration shows that a Search Form component called Search Thing is listening for the Tree Click event and will Refresh Display when it "hears" it, and that the Gender component will Refresh Grid Object upon hearing the same event. The only component to post this event is a Tree component called Select Class to Begin, and it will do so when the application's user single-clicks a selection.



The Delete button tells TBE to delete the currently selected event.

Clicking the Display All button in the upper left lists all the events that are listened for and posted, along with the component names, types, and the component actions that are associated with those events:



Clicking a column heading in one of the panels (for example, clicking on the words "Event" or "Component") will sort that panel's entries by that value.

2.3 Creating Popups

In addition to creating forms for your end users to interact with your data models, TBE lets you create popup windows that can serve as dialog boxes.



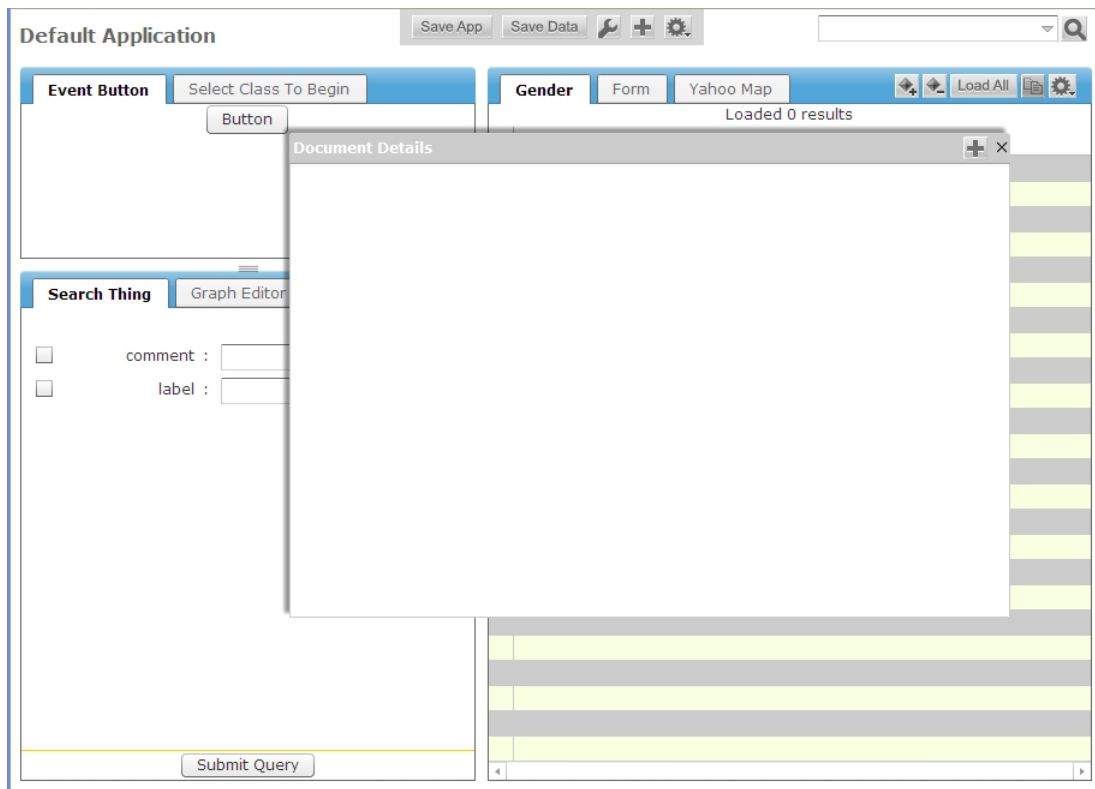
An alternative to TBE popup windows is to display information created as HTML in a browser window created as part of a SPARQLMotion script. See the [Defining SPARQLMotion Scripts](#) section of this book for more information.

When you first create a new popup window, you will not see it right away, because you must first configure the event that makes it appear. If you have not yet configured this event, you can create a temporary event button to do so.

For example, let's say you're creating an application that tracks data about a collection of documents and you're going to create a new popup window called "DisplayDocDetails":

1. Create your new popup window by selecting Add Popup Window from the + menu and then Popup Window from the cascade menu that appears.
2. Create your temporary testing button by Selecting Add Components from the + menu and then Event Button from the cascade menu that appears.
3. Click the wrench button or press F2 to display the Application Configuration console.
4. On the Components and Layout screen, select your new Event Button under Components on the left, and then under Events on the right, define the new event for this button to post by clicking the + to the left of "Click Event" and pick "Create New" from the context menu.
5. Call your new event DisplayDocDetails and click OK.
6. Next, you want to configure your new Popup Window to listen for this DisplayDocDetails event and open up when it hears the event. Select the popup window from the Popup Window section of the Application Configuration console's navigation pane, and then, under Events/Listen, click the + next to "Open the popup".
7. Select "Add Existing Entry," and then in the drop-down combo box, select the DisplayDocDetails event and click the OK button.
8. This would be a good time to change the label of your new popup window to "Doc Details" or "Document Details." Names like this will make it easier to keep track of your application components as your application grows in complexity.

Now, when you close the Application Configuration console and click your new button, your new popup window will appear.

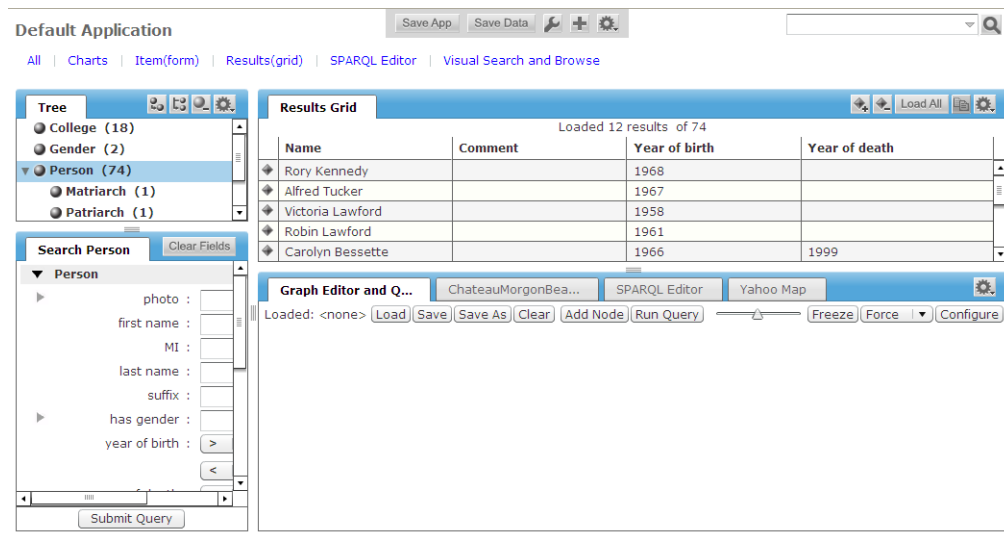


A popup window serves as a container for components. Therefore the popup will appear empty. Clicking the plus sign in the upper-right of the popup window lets you add components and pages to the popup. (A good start may be to add and configure the component that closes the popup—for example, a new button.) As with the Listen event you’ve already configured for this popup, you’ll configure these components on the main Application Configuration console.

You can always change your application so that a different action displays your popup (for example, the selection of a document name on a results grid or tree if your model stores data about documents), and you can change the posted event of your testing button to open other new popups as you create them. If this testing button plays no role in your finished application, you can delete it when you’re finished.

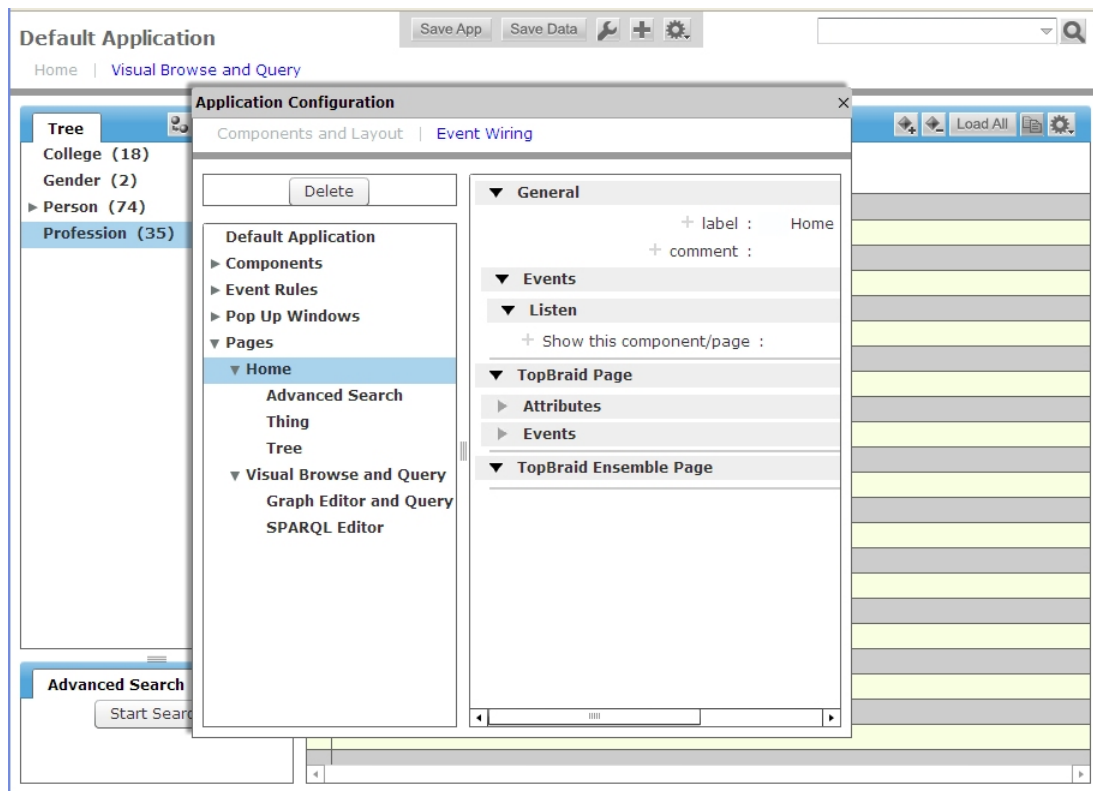
2.4 Pages

Page-based input forms let you spread application components across multiple pages. The following shows a sample application, which has six pages:



Your application's users can switch between pages by clicking on page names at the top of the screen. Each page is a separate window, a container for components, that is configured independently but shares events. Components on any page can be configured to post events that display a particular component in a page, making the pages even more interactive.

Add a new page by selecting Add Page from the + menu. (If you don't see this selection on the + menu, set "Multi-paged Application" to "true" in the TopBraid Ensemble Attributes section of the TBE Application Configuration Console.) Add components to a page the same way you would to a single-page TBE application: by selecting "Add Component" from the + menu while viewing it. On the Application Configuration console, a page's components will appear under the header for that page. For example, the following shows that the page titled "Home" has three components:



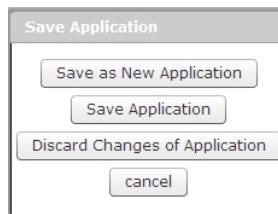
A new page's default name will be the word "Page" followed by a number, but you can modify its label and other properties of the page on the Application Configuration screen as shown above.

The illustration above also shows the "Show this component/page" listen event, which is where you name the events that trigger the display of this particular page. For example, if the "Show this component/page" value for a Product Page named DisplayProductData as the event to trigger the display of this page, and a button on another page had DisplayProductData as the click even to post, then clicking that button on the other page would make the Product Page the currently displayed page for the application.

One page attribute not shown above is "Page Index," which lets you set the ordering of the page names across the top of your multi-page application.

2.5 Modifying and Resaving an Application

To continue customizing after the first save, simply re-launch the saved application. After making modifications click on Save App, which displays this dialog box:



To save the application with the same name, click the "Save Application" button and then, on the Save Application dialog box, click the Overwrite checkbox before clicking Save to save the configured application with the same name.

When saving an application, you can change what data graphs it works with. The "Save Application As" dialog box displays the data graphs that the application is currently working with. You can deselect the current data graph when saving. Note that the de-selection broadens the data graphs your application can work with. For example, an application shown in the next screenshot is compatible with any data graph that imports the Kennedy family ontology, the Dublin Core ontology, the W3C geospatial ontology, and a TopBraid Composer extension to the W3C geospatial ontology. If you deselect all of them, you are saying that your application can work with any data graph. If you deselect the two geospatial ontologies and the Dublin Core ontology, you are saying that your application can work with any data graph that imports the Kennedy family ontology.

When you go back to the console and select the saved application, all the data graphs present in the workspace that are compatible with the application will be shown as available for selection.

It is recommended that the application is saved in the project that will be deployed to TopBraid Live Enterprise Server. Use the "Save To" choice list to select the project you are working with.

When an application is completed, it can be promoted to the "Completed Application" section of the console by moving it from the user-applications directory (folder) to the server.topbraidlive.org project's system-applications directory.

Note that when configuring or running the application using TopBraid Live Personal Server included with Composer Maestro Edition, TBC-ME will display TBLnnnn.n3 files, which are used to track your application's configuration, as you modify the application in your browser. Before quitting TBC-ME, to clear these files from TBE, go to the TopBraid Live Server Administration screen from the TopBraid Suite Console (in a typical configuration, this will be at <http://localhost:8083/tbl/admin/>) and click the **Clear all [n] Sessions** button.

2.6 Configuring a Simple Application: SKOS Editor

This section shows you how to create a simple TBE application that lets end users edit a thesaurus that uses the SKOS ontology. The application will have only two components:

- A tree displaying resources connected by skos:broader property.

- A form showing details of each resource. When the end-user clicks a node of the tree or a class name on the form, the form will refresh to display all the details about that class.

Step 1: Create Ontology

In TBC, create a new RDF/OWL file called `thes1` and import `skos-core.rdf` from the Common folder of the TopBraid library into your new model. (You can do this at the "Create RDF/OWL File" dialog box by checking "SKOS" in the Initial imports list.) Create an instance of `skos:Concept` (a subclass of `owl:Thing`) in this model and call it `Taxonomy Root`. Remember which project it's stored in; we will need it as a root resource for the tree created on your TBE form.

Save your work.

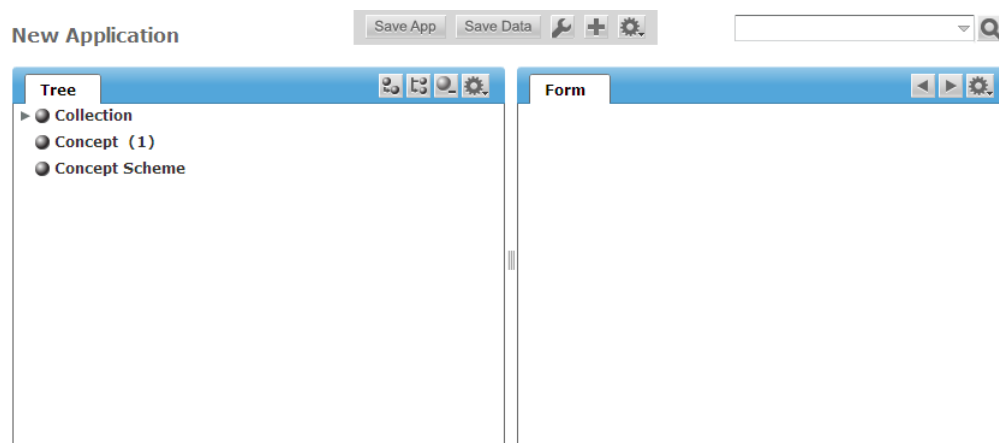
Step 2: Create New Application

In the TBS Administration Console, select "New Application".

On the "Data Graphs Compatible With New Application" screen, select the `thes` model that you just created from the project where it was stored.

Step 3: Select Components

In TBE, click the plus sign icon and select Add Components to add each of two new components to the application: Tree and Form. Drag their tabs to lay them out side by side as shown in the following:



SKOS Editor component layout



Note how the default configuration of the tree component is to show the class structure of the data model; you'll be reconfiguring it to show the tree of taxonomy terms being edited by your new application's end user.

Step 4: Configure Application

Launch the configuration console by clicking the wrench icon or pressing F2. You began with a copy of "New Application"; Click on "New Application" at the top of the navigation pane on the left, and then under General on the right change the application label from "New Application" to "SKOS Editor".



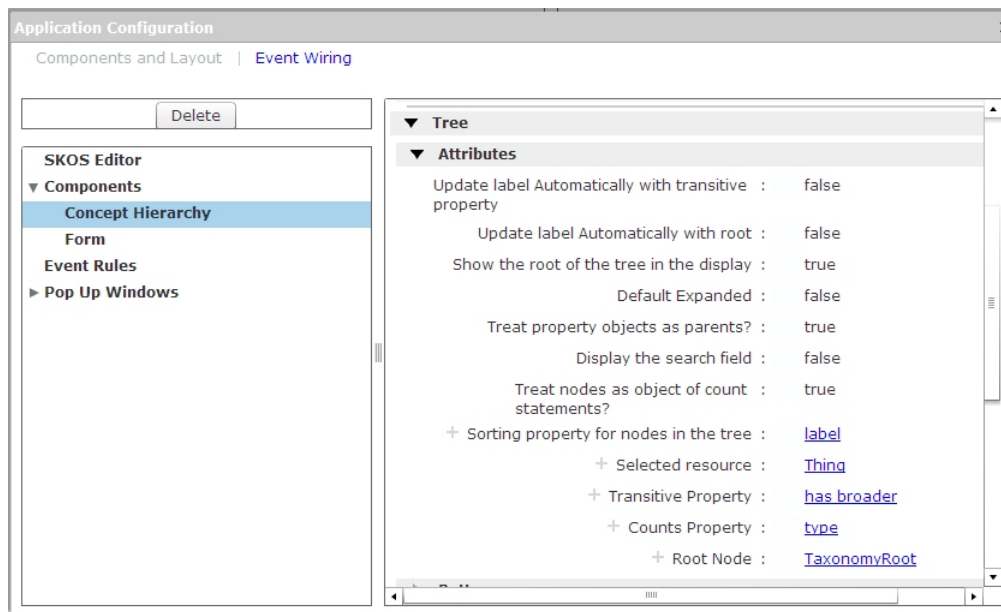
Hover to the left of the existing value and click the "edit" button that appears there to change the value.

Step 5: Configure Tree

Now click on the Tree component on the left side of the Application Configuration console (you may need to first click on the Components node triangle on the tree to expand it into a list of available components) and make the following modifications:

- Rename the Tree's label to read Concept Hierarchy.
- Under Tree/Attributes, change the "Show the root of the tree in the display" setting to true.
- Change the Root Node to the Taxonomy Root (the new resource you created in TBC).
- Change the Transitive Property that identifies tree node child/parent relationships to "has broader" (the label assigned to the skos:broader property).
- Under Events, create a new Post Event for Single-click Selection and call it Tree Selection.

Your configuration should look similar to the one shown here:



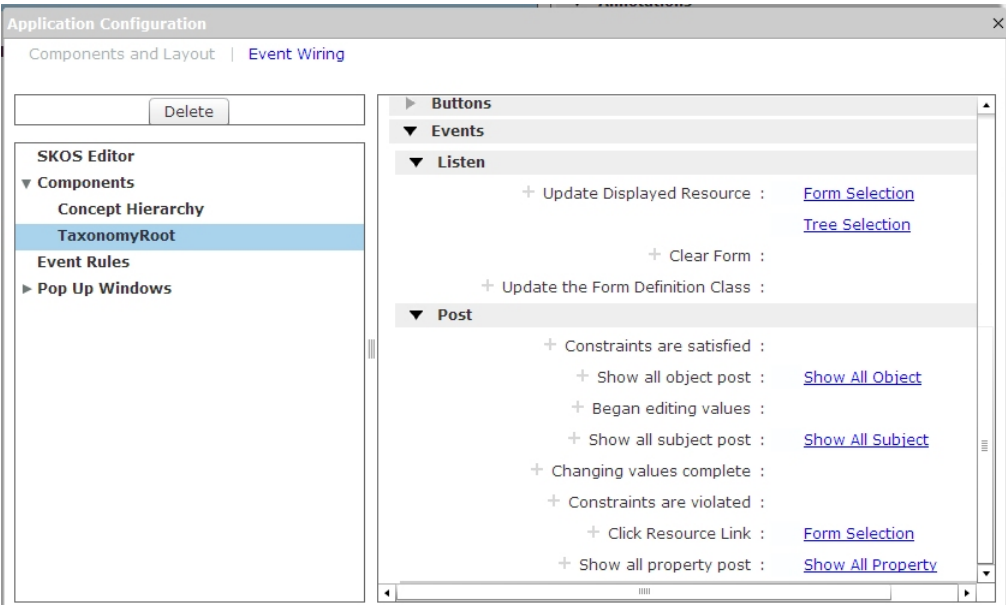
SKOS Editor – tree configuration

Step 6: Configure Form

Now select the Form component on the left of the Application Configuration console and make the following modifications:

- Wire the Form to listen to the Tree Selection event. Click on Add existing entry next to Update Displayed Resource (Listen Event) and start typing "Tree". When "Tree Selection" shows up in the drop down, select it and click OK.
- Create a new Post Event for the Click Resource Link. Call it Form Selection.
- The form is already configured to refresh when the end user clicks on a tree node, which sends a Tree Selection event. You also want the form to refresh if the end user clicks on the name of a concept on the currently displayed form. For example, if the form for the concept Sparrow shows that Sparrow's broader term is Bird, and the end user clicks on Bird, you want it to display data for the concept Bird. Click on Add existing entry next to the Update Displayed Resource even and start typing Form. When "Form Selection" shows up in the drop down list, select it. It will join Tree Selection as the second value for the Tree's Update Displayed Resource listen event.

Your configuration should look similar to the one shown here:

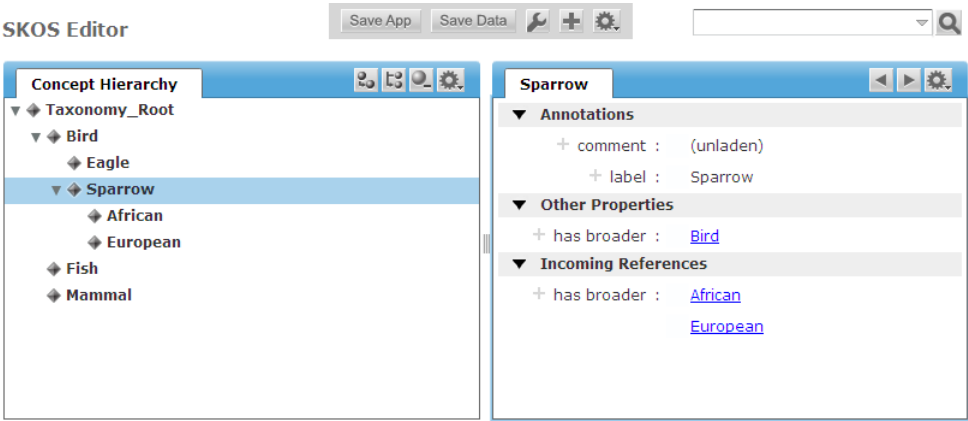


SKOS Editor – form configuration

Step 7: Test and Save

Your new application is done.

You can close the configuration console by clicking the X at the upper right and test your new application by creating new nodes in the tree with the buttons in the upper-right of the tree component and entering additional information in the forms as shown here:



SKOS Editor – assembled application with data entered

As you add and select nodes in the Concept Hierarchy, the tab title and many of the fields in the form on the right will update automatically to provide information about the selected tree node.




To improve the application, you may want to consider customizing forms in TBC. This is explained in the Configuring Custom Forms section.

Follow the instructions in the Creating a TBE Application section above to save the application.

2.7 Configuring Custom Forms and Grids

One of the strengths of TopBraid Suite is the ability for developers to customize how information is displayed to end users by designing customized forms for specific classes. Unlike other Ensemble customizations, form and grid configurations are created in Composer. For more information on Form layout, see the Composer Help page at Help > User Interface Overview > Resource Editor > Form Layout Pane. For more information on grid layouts, see the Instance view in Composer, Help > User Interface Overview > Instances View

The procedure for updating forms from Composer to the Live server is as follows:

1. Assemble an application using Ensemble as you normally would. Any form information that was already stored with your data models is available in your application.
2. If you wish to modify the forms customizations, you can do so using the TopBraid Composer tool supplied with TopBraid Suite (for example, by selecting "Edit form of resource's type" from the form's  menu). These forms customizations can be previewed in Composer using the Form Layout tab when a class is selected in the Class view. Of the three choices on that menu, if the only form that is defined is General, TBC will use it and TBE will use it for both edit and search form layouts. If TBE edit form is defined, TBC will ignore it and will use General if such form exist or default form presentation if it does not, and TBE will use the form defined for edit form presentation. If TBE search form is defined, TBC will ignore it and will use General if such a form exists or the default form presentation if it does not, and TBE will use the form defined for search form presentation.
3. After making the changes, go to the TopBraid Suite Console and select "TopBraid Live Server Administration"
4. Select "Clear all n sessions" (where "n" is the number of sessions currently running) near the top of the page. Make sure that you saved any application changes prior to doing this.
5. Re-launch the application from the TopBraid Suite console.

Updated forms should now be available in your TBE applications.



If you customize forms or grids for a graph stored in a file named `myApp.n3`, the customization information will be stored in a file named `myApp.n3.tbc`. This file will be part of the same project, and should be distributed with the project if that project is deployed on another server. The only information in

When configuring each form property in TBC, the Preferred Widget Type field lets you specify the type of input field used on the TBE form. The following are the choices that may be available, depending on the range of the property being configured for your form:

Label	Description	Type Range
Text area	A multi-line text area.	string
Radio buttons	Radio buttons to select one of our several enumerated values. For datatype properties, this assumes that the range is a datatype enumeration (owl:oneOf). For object properties, it will enumerate all instances of the range class, or use those defined by an OWL enumerated class.	Resource, Literal
Rich text area	A multi-line area of text including HTML markup, ideally rendered as WYSIWYG.	Literal

Label	Description	Type Range
Check box	A check box to select a boolean value.	boolean
List box	A list displaying multiple values to choose from.	Resource, Literal
Table	A tabular display of resources in one or more columns.	Resource
Text input	A single line of text.	Resource, Literal
Combo box	A combo box to pick one out of several enumerated values.	Resource, Literal



The only one of these values that will carry over to TBE forms is Rich text area.

If you select a widget type of <Default>, a widget type will be assigned based on the property's range. For example, if the range is a date, a date widget will appear; if it is a dateTime, a date time widget will appear; if it is a Boolean value, a true/false combo box will appear; if it is a string or number (int, float, double), then a simple text input field will appear, with validation when the user attempts to commit the value.

For more information, see the "Form Layout Panel" page in Composer's Help pages (Help > User Interface Overview > Resource Editor > Form Layout Panel).

The same approach applies in customizing columns displayed in the Results Grid. Select the "Grid Column" class in TBC and adjust the columns using Instances View. Then follow steps 3 through 5 above.

2.8 Configuring Startup Events

To configure your TBE application to execute specific events upon startup, use a SPARQLMotion Relay. See [SPARQLMotion Relay](#) on page 95 for more information.

2.9 Saving TBE Applications

When an application is ready to be saved so it can be used by end users:

1. Click "Save App" and name the application so that "(configurable)" is no longer part of the name.
2. If you want to deliver your new application to users who shouldn't be able to modify the application itself, you can remove the three buttons that are used for configuration (the "Add components" button, the "Show/hide configuration" button and the "Save App" button) from your delivered application. Do this by using TopBraid Composer to open the application file, which can be found in the user-applications folder of the relevant project.

Once you've opened this file, click the "home" button and select **InstallConfigButtons** or **RemoveConfigButtons** from the the TopBraid Composer **Resource** menu to control whether the TBE application includes those buttons. Remember to save the file before opening it in TBE again.

If you have been developing application locally and need to promote it to a server, you will need to deploy the project containing the application file to the server. To accomplish this, use either "Export... > Deploy Project to

TopBraid Live Server" in TBC-ME or "Project Upload" in the TopBraid Live Console. For more, see [Deploying an Application to TBL Enterprise Server](#). Other considerations for deploying projects and applications to a multi-user TBL Enterprise Server installation include setting security and authentication such as that provided by LDAP. This information is covered by the TBL Server Enterprise Installation Guide.

Chapter

3

TopBraid Ensemble User Interface Components

Topics:

- [Event Button](#)
- [Info Box](#)
- [SPARQL Editor](#)
- [Basket](#)
- [Form](#)
- [History Form](#)
- [Graph Editor and Query](#)
- [Results Grid](#)
- [Tree](#)
- [Search Form](#)
- [White Space](#)
- [Yahoo Map](#)
- [Search Field](#)
- [TopBraid Ensemble User Interface Components](#)

This section describes the components that you can use on your TBE forms, pages, and popup windows. See the Component Configuration Fields section below for descriptions of the individual fields used to configure the components.

Examples assume that one of the default applications is loaded with the kennedys.owl data.

3.1 Event Button

An event button gives your application's user a simple way to trigger events that can take place in your application, such as the opening or closing of a popup window or the loading of data into another component. Specify the event for it to trigger as a Click Event under Events/Post.

Buttons can be renamed and can listen for events that disable and enable them, if you want a button grayed out under certain conditions. See the Creating Popups section for an example of adding a Button Component to an application.

3.1.1 Attributes and Events

Attributes

Name	Description	Default Value
Button Label	The label to be used on the actual button.	Button
Icon	The amount in pixels that this button can resize to.	
Icon property	The amount in pixels that this button can be resized to when the label changes.	
Tooltip	The amount in pixels that this button can resize to.	
Tooltip property	The amount in pixels that this button can resize to.	
URL	The url or website that will be launched in a new tab on click.	
URL property	The property to use to match against the selected resource to find the 'URL' value.	
Button's Maximum Width	If not using Static width, setting a Max Width on a button's which label changes is useful. The amount in pixels that this button can resize to.	200
Button's Width	The width of the button in pixels.	
Icon Width	Set this to the desired width of the icon. please note that not setting this value when using an icon can result in an icon that is too large for the buttons max width.	
Icon Height	Set this to the desired height of the icon.	

Name	Description	Default Value
Button's Label Placement	The location of the label on the button. This value is only relevant when an icon is specified. The valid options are 'left', 'right', 'top' and 'bottom'.	right
Update Button Label Automatically	Whether or not to update the button's label automatically with the 'Current Resource'.	false
Enabled State	When set to true, the button will be clickable. When set to false, the button will appear grayed out and unclickable.	true
Current Resource	This is the resource that will be passed in 'On Click' post event, and set by the 'Update Current Resource' listen event.	

Post events

Name	Description
Click Event	This event is dispatched when the user single clicks on a button which 'Enabled State' is true. If there is a 'Current Resource', then it will be posted with the event.

Listen events

Name	Description
Update Current Resource	When posted to, 'Current Resource' will be set to that of the resource in the event.
update Tooltip Property	When this listen event is posted to, this button's 'tooltip property' (see icon property) will be updated. Useful for dynamically changing the icon. (For many applications, using this effectively may require the use of a sparql relay to decide on the property).
update Icon Property	When this listen event is posted to, this button's 'icon property' (see icon property) will be updated. Useful for dynamically changing the icon. (For many applications, using this effectively may require the use of a sparql relay to decide on the property).
Enable Event	When this listen event is posted to, this button's 'Enabled State' will be set to true.

Name	Description
Disable Event	When this listen event is posted to, this button's 'Enabled State' will be set to false.

3.2 Info Box

The Info Box component lets you add text to a form, making it ideal for instructions about how end users should use the rest of the form. To configure this component with hardcoded text, enter a value in its Static Text attribute. For dynamic text, set its Property attribute to the name of a property that holds the text you want displayed. The listen events Update Property and Update Current Resource give you control over when a dynamic value is updated.

To format the text, inline HTML elements such as **b** for bold and *i* for italics are supported, and additional configuration attributes let you control the font size and the spacing around the displayed text.

3.2.1 Attributes and Events

Attributes

Name	Description	Default Value
Padding Left	The padding on the left-hand side of the component.	5
Padding Right	The padding on the right-hand side of the component.	5
Padding Top	The padding on the top side of the component.	5
Padding Bottom	The padding on the bottom side of the component.	5
Static Text	Alternatively from displaying the text of a property of an instance or class(current resource), the infobox can display any text specified here. Please note that the infobox displays html text.	
Property	The property which will be used to get the displayed text from the current resource. i.e. if the property is <code>rdfs:comment</code> , the infobox will display the <code>rdfs:comment</code> for the current resource. note: the text found will override the 'static text'.	http://www.w3.org/2000/01/rdf-schema#comment
Font Size	A number representing the size of the font to use.	13
Current Resource	The resource the infobox currently is holding. The resource is used via the 'Property' field to display found text. Use the 'Update Current Resource' listen event or set this value manually.	

Listen events

Name	Description
Update Current Resource	When posted to, this event will update the "Current Resource" value to the passed resource.
Update Property	When posted to, this event will update the "property" value to that of the passed resource.

3.3 SPARQL Editor

This component lets your application's users enter SPARQL queries. In the default applications, when the user clicks the "Submit Query" button, the Results Grid component is configured to listen for the query results and display them with a column for each bound variable, as shown here:

The screenshot displays two components. The top component is the 'Results Grid', which shows a table of 15 results with columns 'first', 'last', and 'born'. The results are: Patricia Kennedy (1924), Alfred Tucker (1967), Linda Potter (1956), Robin Lawford (1961), Jeannie Ripp (1965), Virginia Bennett (1936), Molly Stark (1968), and Jeffrey Ruhe (1952). The bottom component is the 'SPARQL Editor', which has tabs for 'Form', 'Graph Editor and Qu...', and 'Yahoo Map'. The 'Form' tab is active, showing a SPARQL query: `SELECT ?first ?last ?born WHERE { ?s :firstName ?first; :lastName ?last; :birthYear ?born }`. A 'Submit Query' button is located at the bottom of the editor.

3.3.1 Attributes and Events

Post events

Name	Description
Query Results Requested	This event will be posted when the user clicks the 'Submit Query' button and will post the available text as a SPARQL query. (Grid can listen to this)

Listen events

Name	Description
Edit Query	When posted to, this listen event will populate the component with the posted SPARQL in text form.

3.4 Basket

The Basket is a holder for resources. It can listen to an event and then add the resource associated with the event to the basket (for example, upon clicking a resource in the grid). You can drag and drop resource into the basket, and you can drag a copy of the resource from the basket to another component.

This component includes a button to remove the selected/highlighted resources from the basket.

3.4.1 Attributes and Events

Attributes

Name	Description	Default Value
Selected resource	This value is updated by selecting a resource in the basket by clicking on it. Setting the value of this attribute manually will highlight the resource (if the resource is present in the basket).	

Post events

Name	Description
Single-click Selection	This event is dispatched when the user single clicks a resource in the basket.
Double-click Selection	This event is dispatched when the user double clicks a resource in the basket.
Selection Change	This event is dispatched when the selection in the basket has changed. The resource which is now selected will be the resource which is posted.

Listen events

Name	Description
Add a Resource	This listen event will add the passed resource to the basket.

Buttons

Name	Description
Delete Button	Delete the resources selected in the basket.

Drag events

Name	Description
Dragging of a basket item	This event occurs when user releases a dragged node from the basket onto a valid drop target.(i.e. tree, visualization) Adding an event here will enable dragging.

Drop events

Name	Description
Add a resource via drag/drop	This event occurs when user releases a dragged node onto the basket. Adding an event here will enable this as a drop target for the specified event.

3.5 Form

A form component displays the data associated with a resource, giving you the TBE equivalent of the main form at the center of TopBraid Composer development. This lets you display information about a particular class, instance, or property, depending on the setting of the form's Item Displayed attribute. This also resets the name of the form displayed on its tab, as shown in the following, where the form has been set to point to the data for Jacqueline Bouvier:

The screenshot shows a web-based form for 'Jacqueline Bouvier'. The form has a blue header bar with the name 'Jacqueline Bouvier' and navigation buttons. Below the header, there are two tabs: 'Person' and 'Family'. The 'Person' tab is selected and displays a photo of Jacqueline Bouvier, along with fields for first name (Jacqueline), MI (L), last name (Bouvier), suffix, gender (female), year of birth (1929), year of death (1994), and alma mater (George Washington University). The 'Family' tab is also visible, showing fields for spouse (Aristotle Onassis), child (Caroline Kennedy, John Kennedy Jr, Patrick Kennedy), parent, comment, label, middle name (Lee), and name (Jacqueline Bouvier).

See the [Configuring Custom Forms and Grids](#) section for information on customizing the appearance of forms.

The label displayed in the tab for a form component is the label of the currently selected resource, where the label is defined by the TBS dictionary (see). To define a static form tab label, configure the form's "Update label Automatically" attribute to "true" and define the form's label to be the desired tab label.

While the editing of most text fields in the TBE user interface requires you to click on the "edit" button to the left of the field first, you have a greater control over this in the Form Editing Mode attribute of the Form component gives you more control over this. A value of "Review" puts the form in the default behavior, a value of "View" sets the form to a read-only mode for just displaying data, and a value of "Edit" opens up all the fields at once for editing.

3.5.1 Attributes and Events

Attributes

Name	Description	Default Value
Form Definition Class	If not specified, the form will use Form definition provided by TBC to the class which the instance belongs to. Apart from being able to specify search form, or edit form layouts, this property(when specified) allows a configurer to specify a different class's form definition (also defined by TBC) to use for this form.This is especially useful if a resource belongs to more than one class and you want to make sure that it's displayed in TBE with the form designed for one particular class.	
Form Editing Mode	The editing mode of this form. Form modes: View, Edit and Review. View mode has no editing available to the user. Review Mode allows editing, while Edit mode has every widget in 'edit mode' to allow quick editing of all values.	http://server.topbraidlive.org/2008/ensemble#ReviewMode
Displayed Resource	The resource for which information is displayed in the form.	
Include constraint violations	If set to true, a warning will be shown for all values which violate constraints. Please refer to documentation for more about constraint violation.The Getting Started with SPIN tutorial describes how to define and test SPIN constraints.	false
Maximum numbers of values for a widget	The maximum values to display before showing a 'show all' link which will provide a triple match to a grid. See 'Show all subject post'	10
Update Label Automatically	Indicates whether to use the label of the currently displayed resource as the label on the component.	false

Post events

Name	Description
Began editing values	
Changing values complete	
Click Resource Link	Click Resource Link: this event is dispatched when a user clicks a hyperlink in the form. Currently, the

Name	Description
	components which best can accept a resource is the form or search form component.
Constraints are violated	The form is considered valid until a constraint is violated, at which point an event will be dispatched indicating so. This will only be relevant when 'Include constraint violations' is true.
Constraints are satisfied	The form is considered valid until a constraint is violated. Thus, after all violations have been removed, an event will be dispatched indicating that the form is valid. This will only be relevant when 'Include constraint violations' is true.
Show all property post	The event corresponding to the matching Predicate of a Triple Match(Subject, Predicate, Object) which the form uses for the "show all" feature when a widget has more values than the maximum value. A default event has been provided, but the configurer can specify any grid/popup/page to listen to this event.
Show all object post	The event corresponding to the matching Object of a Triple Match(Subject, Predicate, Object) which the form uses for the "show all" feature when a widget has more values than the maximum value. A default event has been provided, but the configurer can specify any grid/popup/page to listen to this event.
Show all subject post	The event corresponding to the matching Subject of a Triple Match(Subject, Predicate, Object) which the form uses for the "show all" feature when a widget has more values than the maximum value. A default event has been provided, but the configurer can specify any grid/popup/page to listen to this event.

Listen events

Name	Description
Update Displayed Resource	This event when posted to will update the 'displayed resource' property and consequently, the view of the form to the resource that was posted. Any event which posts a resource can be used to post to this listen event.
Clear Form	Clear the form of anything its displaying which will remove the 'displayed resource' value.

Name	Description
Update the Form Definition Class	An Event which will update the 'Form Definition Class' to the posted resource (should be a class). This functionality, to automatically update the properties which a form displays for a given instance, is in beta.

Buttons

Name	Description
Backward Button	Navigate backward to the previously displayed item
Forward Button	Navigate forward to the most recent item

Drop events

Name	Description
Drop a in as input to add it as a value.	This event occurs when user releases a dragged node into an input widget on the form.

3.6 History Form

A history form lets you see the history of edits made to resources in a model if a Teamwork Repository has been set up for that model. To set one up, you must log in and set up a repository within the model you're going to use.

To log in when using the Personal Edition of TopBraid Live included with TopBraid Composer Maestro Edition, send your browser to the TopBraid Suite Console, which will check the users.n3 file stored in the server.topbraidlive.org \dynamic directory of your TopBraid Composer workspace for valid names. When using EVN with the Enterprise TBL Server on a shared system, the contents of the users.n3 file are updated automatically as each new user on the system logs in to EVN, based on what TBL knows from the TomCat server about authenticated users on that system. (TomCat may rely on LDAP for authentication, or it can be configured to provide basic authentication by itself.)




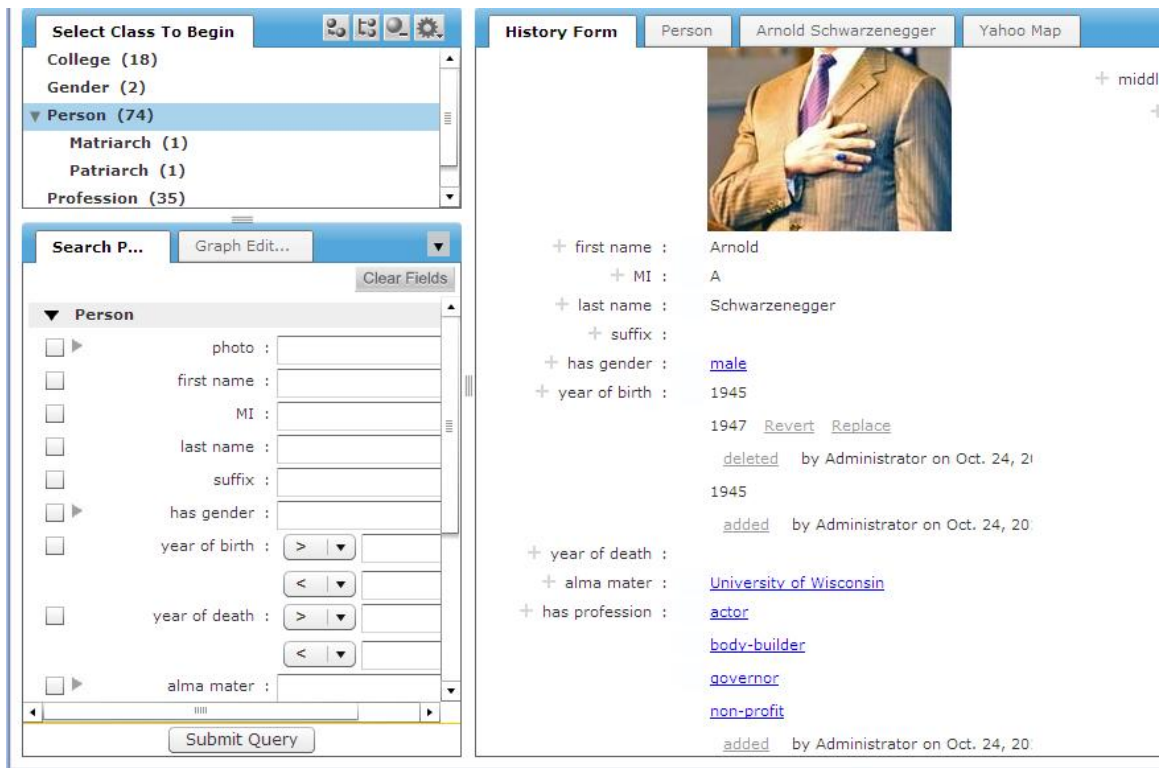
Teamwork support is only supported for models stored using formats that support persistent blank node identifiers, including Jena SDB, TDB, and Oracle RDF projects. It will not work with a disk text file such as RDF/XML, N3, or Turtle.

To set up a repository for a model, open the model in TopBraid Composer and select **Set up teamwork changes repository** from the **Model** menu. Set the user name to a name in the users.n3 file described above. The best format would be the same one as the model itself is stored in. The display label and short description are used by applications such as EVN when listing models to work with. Fill these out and click **next**, which leads to a dialog box with configuration parameters that depend on the format you've chosen to store the data. After filling this out, click **Finish**.

Once you've finished setting up the teamwork repository, select **Close All** from the TopBraid Composer **File** menu, and then restart TBC-ME.

Adding a History Form to a TBE application that uses this data model is very similar to adding a regular **Form** on page 45. After editing one of the resources and clicking the **Save Data** button, though, clicking the History Form's

history button  toggles a display of changes made to that resource. For example, in the following, it shows that "non-profit" has been added as a "has profession" value, and that the "year of birth" value has been changed from 1947 to 1945:



The screenshot displays the TopBraid Ensemble interface. On the left, the 'Select Class To Begin' pane shows a list of classes: College (18), Gender (2), Person (74), Matriarch (1), Patriarch (1), and Profession (35). The 'Person' class is selected. Below this is a 'Search P...' pane with a 'Graph Edit...' button and a 'Clear Fields' button. The search results for 'Person' are shown, including fields like photo, first name, MI, last name, suffix, has gender, year of birth, year of death, and alma mater. The right pane shows the 'History Form' for the 'Person' class, displaying a photo of Arnold Schwarzenegger and a list of changes. The changes include: first name: Arnold, MI: A, last name: Schwarzenegger, suffix: , has gender: male, year of birth: 1945, year of death: , alma mater: University of Wisconsin, and has profession: actor, body-builder, governor, non-profit. The history also shows that the year of birth was changed from 1947 to 1945, and the has profession was changed from actor to non-profit.

The **Revert** link next to a removed value reverses the deletion, restoring the original value. The **Replace** link replaces the existing value with the one next to this link. (In the example above, both of these links would have the same effect.)

Along with the same configuration options that a regular Form has, a History Form lets you configure Hide History and Show History listen events so that, in addition to the history button, you can have TBE application logic trigger the displaying and hiding of a resource's edit history.

3.6.1 Attributes and Events

Post events

Name	Description
Change Clicked	Triggered when the history view's "added" or "deleted" links are clicked.

Listen events

Name	Description
Show History	Display the version of the form showing the edit history.

Name	Description
Hide History	Hide the edit history, showing the regular version of the form.

Buttons

Name	Description
Toggle History Button	Show or hide history on the form.

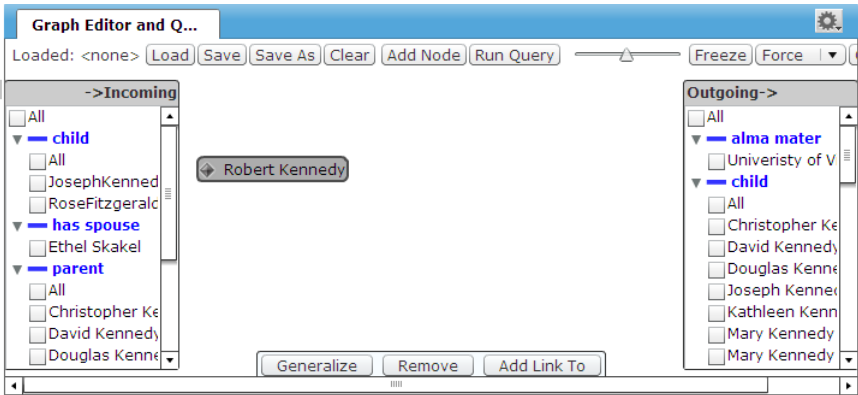
3.7 Graph Editor and Query

The graph editor lets your end user graphically generate and run SPARQL queries much like you can on the Graph view of TopBraid Composer. For example:

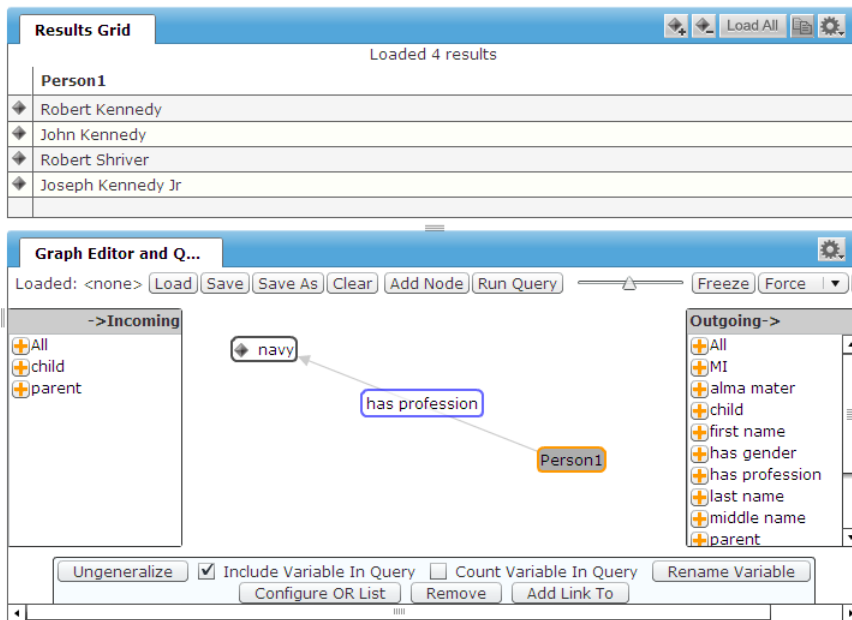
1. Click the Add Node Button, move the cursor to the place in the window where you want the new node, and click there.
2. In the Select Instance message box, enter Robert Kennedy and click OK. You'll see the new node appear.
3. Click the new node, and buttons and list boxes appear so that you can add more to your graph:



You can also drag and drop resources from other components into the Graph Editor and Query canvas provided that the Drag and Drop events have been configured.



4. The Incoming list box lets you build patterns where Robert Kennedy is the object of a triple, and the Outgoing list box lets you build patterns where Robert Kennedy is the subject of a triple. Scroll down to "profession" in the Outgoing list, click "navy", and a new node appears titled "navy" with a labeled line titled "profession" linking the two nodes.
5. Click on the Robert Kennedy button and then the Generalize button underneath that to turn this node into a variable. Now the query represented by this graph asks for everyone in the database who ever had a profession of "navy". In a SPARQL triple pattern form, the query reads "?Person1 :hasProfession :navy".
6. Click the Run Query button, and the results of your query appear in the Results Grid and the query itself appears in the SPARQL Editor component.



As you saw in the Graph Editor after you clicked Generalize, many other buttons appeared to let you graphically modify your query. These let you create even more complex queries without entering any actual SPARQL code.

Graphs (including graphs representing queries) can be saved for future use by clicking on the "Save" button. Saved graphs are displayed by clicking on the "Load" button and selecting the previously saved graph. Saved graphs are saved in the current session. To save a graph so others can choose it from the "Load" button the application must be saved. This will save the graph as part of the application definition that can then be deployed to a TBL Enterprise Server for other users.

3.7.1 Attributes and Events

Attributes

Name	Description	Default Value
Settings	This attribute is for internal use only is under development.	
Image Property	This attribute accepts <code>rdfs:Resource</code> as a property to be used for any given nodes thumbnail image.	

Post events

Name	Description
Node Selected	This event is dispatched when the user selects a node in the graph viz. The resource of the node will be passed. (form, button, relay can listen to this event)
Query Results Requested	This event is dispatched when the user clicks the 'run query' button. Currently, the chart and grid can listen to this event.

Drop events

Name	Description
Drop resource into visualization	This is a listen event which will add the dragged item to the graph visualization as a new node. Currently, the grid and tree can post these drop events.

3.8 Results Grid

The Results Grid displays a set of results. These can be the results of a query or they can be all the resources that match the triple with the predicate specified in the grid's Grid Property attribute and the subject or object specified in the grid's Grid Resource attribute, depending on the setting of the 'Use resource as object of triple match?' attribute.

Queries can be provided to the Results Grid in the following way:

- A query can be entered directly in the SPARQL Query field. When a Results Grid component is first displayed it will run the query. A re-run of the query can be triggered by Refresh search results event.
- A query template can be entered in the Query Template field. A query template is a SPARQL query that includes ?this variable. Bind and Execute Query Template event, replaces ?this with the resource that is passed by the posting event and runs the query.
- A query can be passed to the Results Grid by the SPARQL Editor component using Display SPARQL Query event.
- Q query can be generated and passed to the Results Grid by the Search Form and Graph and Query Editor components using Execute Query event..

The results grid can also post data to other components when rows on the grid are selected. See the descriptions of the Tree, Graph Editor and Query, and Yahoo Map components for examples of the Results Grid in use.

See the [Configuring Custom Forms and Grids](#) section for information on customizing the appearance of a results grid.

3.8.1 Attributes and Events

Attributes

Name	Description	Default Value	Dependency
New resource filter	When adding a resource in the grid using the 'Add' icon, the options to add will be filtered by this value.		
Show number of results loaded	To show or not show the "loaded:x" status box.Setting this to true will display the number of results loaded so far; setting it to false will not show that number.	true	
Grid Object	The 'Grid Object' will be used as the object of a triple match in which the 'Property' is the predicate. The grid will then be populated with the subjects of the matching triples. Setting this will nullify the 'Grid Subject'. Note: posting to this event is equivalent to deprecated listen event 'update		'Grid Property' must have some value. To dynamically reset 'Grid Object' and refresh results,

Name	Description	Default Value	Dependency
	resource' when the deprecated value 'resource as object of triple match' was true.		use 'Refresh Grid Object' event.
Grid Subject	The 'Grid Subject' will be used as the subject of a triple match in which the 'Property' is the predicate. The grid will then be populated with the objects of the matching triples. Setting this will nullify the 'Grid Object'.Note: posting to this event is equivalent to deprecated listen event 'update resource' when the deprecated value 'resource as object of triple match' was false.		'Grid Property' must have some value. To dynamically reset 'Grid Object' and refresh results, use 'Refresh Grid Object' event.
Column resource	<p>This attribute allows you to override the default column layout of a grid populated using either 'Grid Object' and 'Property' attributes (and corresponding events) or 'Grid Subject' and 'Property' attributes (and corresponding events).</p> <p>When a grid's 'Property' attribute = "rdf:type" and the grid is populated using a 'Refresh Grid Object' event, by default the columns of a grid are determined by the column layout for the 'Grid Object' resource as defined in the TBC column layout, which is stored in the .tbc metadata file for a given data file. When the property is different from "rdf:type" and a grid is populated using any of the non-query events ('Refresh Grid Object', 'Refresh Grid Subject', 'Refresh property'), by default the grid will present a single column.</p> <p>The 'Column Resource' feature allows you to override the default column layout with the (specified in corresponding .tbc) column layout of ANY class. Simply set this value to a resource which has the desired column layout.</p>		'Grid Property' and either 'Grid Object' or 'Grid Subject' attributes must have some value set either directly or through the corresponding events.
Property	The grid will be populated with all resources that are either the subject or object of a triple match where 'Property' is the predicate in the triple match.		Either 'Grid Object' or 'Grid Subject' attribute (but not both) must have some value. To dynamically reset Property and refresh results, use the 'Refresh property' event.

Name	Description	Default Value	Dependency
Update Label Automatically	If set to true, the tab's label will update to the label of the displayed resource when the displayed resource changes. Otherwise you can define the component's label manually, and set this property to false.	false	Should only be used when Property = "rdf:type" and the grid values are populated based on the 'Grid Object'.
Selected resource	This value is updated by selecting a resource in the grid by clicking on it. Setting the value of this attribute manually will highlight the resource (if the resource is present in the grid).		
Allow to rearrange columns	Allow the user to drag the columns in order to rearrange them. This is in beta because column layout for the is in development. (rearranged columns do not persist)	false	
SPARQL Query	This value represents the current SPARQL query used to populate the grid. You can enter any SPARQL query in this field. This attribute can also be set via the 'Display SPARQL Query' event.		The 'Display SPARQL Query' event will update this attribute and trigger a rerun of the query. To rerun the query when the attribute is populated using a different approach or when the data has changed while the query did not change, use the 'Refresh search results' event.
Search Criteria	This value represents the SPIN query used to populate the grid. This attribute is set via the 'Execute Query' event. The difference between a SPIN and a SPARQL query is that a SPARQL query is a text string while a SPIN query uses an RDF serialization of SPARQL. It is represented as a node of rdf:type sp:Select, or a SPIN node.		The 'Execute Query' event will populate the attribute and run the query. To rerun the query when the attribute is populated using a different approach or when the data has changed while the query did not

Name	Description	Default Value	Dependency
			change, use the 'Refresh search results' event.
Query Template Variable	This value represents the node that is to be bound to the ?this variable when the 'Query Template' functionality is executed. This variable is set to the passed resource via the 'Bind and Execute Query Template' event.		Requires some value in the 'Query Template' attribute.
Query Template	The SPARQL query to be used for the Query Template Functionality. The query's WHERE clause must contain a variable ?this which will be bound to the node specified in 'Query Template Variable Binding' attribute.		Requires some value in the 'Query Template Variable Binding' attribute. Works with the 'Bind and Execute Query Template' event.

Post events

Name	Description
Single-click Selection	This event is dispatched when the user single clicks on the row in the grid. The resource which is clicked will be the object which is posted.
Double-click Selection	This event is dispatched when the user double clicks on the row in the grid. The resource which is clicked will be the object which is posted.
Selection Change	This event is dispatched when the selection in the grid has changed. The resource which is now selected will be the object which is posted.

Listen events

Name	Description
Refresh Grid Object	This listen event will update the 'Grid Object' value which is used to populate the grid with a triple match.
Refresh Grid Subject	This listen event will update the 'Grid Subject' value which is used to populate the grid with a triple match. Note: used to be 'update resource'.

Name	Description
Refresh property	When posted to, this listen event will update the 'Property' attribute and repopulate the grid with triples corresponding to the newly passed property.
Refresh search results	This listen event will refresh the grid if it has been populated via a SPARQL/SPIN query. If the grid is populated with the 'Grid subject' or 'Grid object' attributes, triggering this event will have no effect.
Execute Query	When posted to, this listen event will populate the grid with the results of a SPIN query. (Search Form and Graph Visualization post these queries).
Display SPARQL Query	When posted to, this listen event will populate the grid with the results of a SPARQL query. (SPARQL editor post these queries).
Bind and Execute Query Template	This listen event, when posted to will trigger the query template functionality by setting the 'query template' variable to the passed resource, and executing the query specified in the 'Query Template'.
Update the Column Resource	Change the column resource to that given in the event. Note: when specified, the column resource will always be used to determine which columns show in the grid.
Clear selection(s)	Deselect any selected items in the grid.

Buttons

Name	Description
Create New Button	Add new resource.
Delete Button	Remove the resources selected in the grid.
Load All Button	Load all matches into the grid.
Copy Button	Copy the selected entries to the clipboard for pasting into another application (e.g. to a spreadsheet).

Drag events

Name	Description
Dragging of a grid entry	This event occurs when user releases a dragged node from the grid onto a valid drop target.(i.e. tree, visualization) Adding an event here will enable dragging.

Drop events

Name	Description
Refresh display via drag/drop	This event occurs when user releases a dragged node onto the grid. The grid will shows subjects related to the dragged resource. Adding an event here will enable this as a drop target for the specified event.

3.9 Tree

The Tree component displays data in a tree. The end user can expand and collapse branches of the tree for viewing and can add and delete new nodes if your configuration of the tree allows them to.

In the default applications, the clicking of a tree node posts the event ClassOfInterest. The Search Form and Results Grid components listen for this information and populate their display with the relevant data. For example, with the kennedys data, if you click the Person class on the Tree component, you'll see the instances of that class on the Results Grid, and the Search Form will become a Search Form for Person data, with all the relevant properties:

Default Application Save App Save Data ⚙️ + ⚙️ 🔍

Tree

- College (18)
- Gender (2)
- Person (74)**
 - Matriarch (1)
 - Patriarch (1)

Search Person Clear Fields

Other Properties

MI :

age : >

<

▶ alma mater :

▶ child :

first name :

gender :

last name :

Submit Query

Results Grid Load All ⚙️

Loaded 26 results of 74

Name
Robert Kennedy
Mary Kennedy
Jean Olsson
Christopher Kennedy
Robin Lawford
Victoria Gifford
John Kennedy
Robert Kennedy Jr
Kathleen Kennedy
Rose Kennedy
Mark Shriver
Ethel Skakel
Jean Kennedy
Sydney Lawford
Molly Stark
Jeannie Ripp
Joseph Kennedy Jr
John Kennedy Jr

3.9.1 Attributes and Events

Attributes

Name	Description	Default Value
Treat property objects as parents?	Setting this attribute to true will display a tree in which each parent-child relationship represents a triple in which the parent is the object and the child is the subject; setting this attribute to false displays a tree in which each parent-child relationship represents a triple in which the parent is the subject and the child is the object.	true
Treat nodes as object of count statements?	Count queries (which are just triple matches) use the counts property and the node as the subject of the triple match. When using 'show tree in reverse', you may need to toggle this value to use the node as the object instead.	true
Default Expanded	Change this attribute so that each time the tree is populated, all nodes will be expanded(to a depth level of 4).	false
Root Node	The root node which the tree is currently displaying. All nodes in the tree are populated by recursively executing a triple match over found nodes beginning with the root node.	http://www.w3.org/2002/07/owl#Thing
Transitive Property	an rdfs Property (e.g. rdfs:subClassOf) by which to use for the triple match that relates child-parent nodes in the tree in the tree.	http://www.w3.org/2000/01/rdf-schema#subClassOf
Counts Property	an rdfs Property (e.g. rdfs:type) by which to find instances of and use to provide count information(e.g. Thing(45)).	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
Update label Automatically with root	Setting this to true will automatically include the 'root node' in the label. Not mutually exclusive to 'Update label Automatically with property'	false
Update label Automatically with transitive property	Setting this to true will automatically include the 'transitive property' in the label. Not mutually exclusive to 'Update label Automatically with root'	false
Display the search field	Include a convenience search field to quickly find and highlight items in the tree (helpful for large class hierarchies).	false
Show the root of the tree in the display	Toggle this feature to true in order to make the root node visible/hidden in the tree.	false

Name	Description	Default Value
Selected resource	The currently selected item in the tree.	
Sorting property for nodes in the tree	A rdfs Property (i.e. rdfs:Label) for the tree to use when sorting items. If no property is specified, then the displayed label will be used to sort the nodes.	

Post events

Name	Description
Single-click Selection	This event is dispatched when the user single clicks on an item in the tree. The resource which is clicked will be the object which is posted.
Double-click Selection	This event is dispatched when the user single clicks on the row in the grid. The resource which is clicked will be the object which is posted.
Selection change	This event is dispatched when the 'Selected Item' in the tree is changed. The resource which is clicked will be the object which is posted.

Listen events

Name	Description
Switch Root	Post to this listen event to set the 'Root Node' in the tree to the passed resource.
Highlight a node	Post to this listen event to set the 'Selected Item' in the tree to the passed resource (if that resource exists in the tree).
Change the transitive property	Post to this listen event to dynamically change the 'Transitive Property' to the passed resource.

Buttons

Name	Description
Add Child Button	Add a child node to the selected node.
Add Sibling Button	Add a sibling node to the selected node.

Name	Description
Delete Button	Delete the selected node.

Drag events

Name	Description
Drag a tree resource	If you use this event, it will enable dragging of a tree resource. The event will be dispatched when the user begins dragging the dragged node.(the tree, grid, or visualization, if it has a drop event configured to listen to the drag event). The resource posted will be the dragged item.

Drop events

Name	Description
Drop resource into tree	This event is fired when a dragged node is dropped onto a valid drop target. To make a drop target a valid drop target for a given drag, simply configure the 'drop event' to be the same event as the 'drag event'. Currently, only the tree and grid can post to this event.

3.10 Search Form

The Search Form displays a form for a particular class and, after the end user clicks the Submit Query button, posts the result. In the default applications, the Results Grid listens for and displays the result of the search, as shown here:

Search form fields that represent relationships between resources (object properties) can be expanded to reveal nested form. This enables a user to formulate richer queries by setting search criteria not only for the direct properties of the selected resource type, but also for the properties of related resources. For example, one could create a search that finds all Roberts (direct property of resources of type Person) that went to an Ivy League school (property of related resources of type College).

Another useful feature of the Search Form is ability to identify what columns should be shown in the result. This is done by checking a checkbox to the left of each field. Search form must be configured to show the checkboxes by setting "Include Checkboxes" attribute to true.

Listen events

Name	Description
Refresh Display	Updates display to show a search form which corresponds to the posted resource. Currently, the tree, grid, and clicking a resource in a form are best supported for posting to this event.

Buttons

Name	Description
Clear Fields Button	Clear all fields in the search form.

Drop events

Name	Description
Drop a in as input to add it as a value.	This event occurs when user releases a dragged node into an input widget on the form.

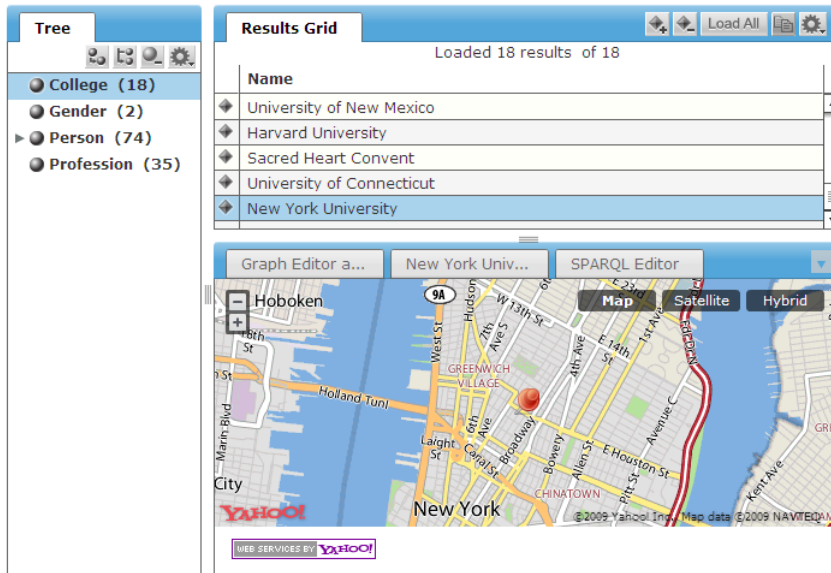
3.11 White Space

White space components give you greater freedom in how you arrange your application's components. Because you can hide the display of any component's tab by clicking next to the tab, white space components can be arbitrarily-size blank rectangles that you use to space out other components. For example, the following has a white space component on the left, one on the right, and one between the Search and Search Results components shown:

3.11.1 Attributes and Events

3.12 Yahoo Map

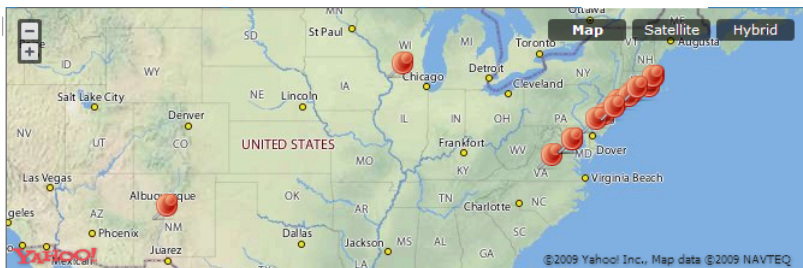
This zoomable map uses Yahoo's API to display one or more points on a map when given the latitude and longitude of those points. For example, when using the default applications with the kennedys data, click the College node on the Tree to list the data graph's colleges on the Results Grid Component. Click one of the college names there and a map will appear in the Yahoo map component with a red pushpin showing the location of the college.



This happens because in the default applications the Results Grid posts the event `ItemOfInterest` when a row of the grid gets single-clicked, and the Yahoo Map component listens for `ItemOfInterest` and refreshes the display upon receiving this event if the data accompanying the event has longitude and latitude information.

The map can also display the members of a class. In the Configuration Console for the default applications, you can see that the Yahoo Map has two options for events to listen for: "Refresh Display (class members)" and "Refresh Display (individual)." The latter is set to both `ClassOfInterest` and `ItemOfInterest`, which is how it displayed the map for the individual college that you selected above.

Because a single click to a node of a Tree Component posts the `ClassOfInterest` event, if you add `ClassOfInterest` as a "Refresh Display (class members)" event for the Yahoo Map to listen for, clicking class names in the Tree Component that have any latitude and longitude information associated with their members will display a map with push pins at those locations. The Person class only has one member with this information, California governor Arnold Schwarzenegger, so a pin is displayed on Sacramento. The College class has many more members with this data, so the map displays a larger portion of the United States with many more pins displayed.





For classes with a large number of members that have latitude and longitude data, configuring the Yahoo Map to display all of their locations can lead to performance issues, so use it with care.

3.12.1 Attributes and Events

Attributes

Name	Description	Default Value
Yahoo Application ID	A developer yahoo map api key. Please get one at http://developer.yahoo.com/maps/simple/	Xg_nVozV34FE7rSC4h7v4.QKEpnCe7FA8
Mapped Class	Resources of this type are marked on the map.	
Mapped Resource	This resources is marked on the map.	
Display Info for Selected Marker	When a marker is clicked, display it's information on the map.	true

Post events

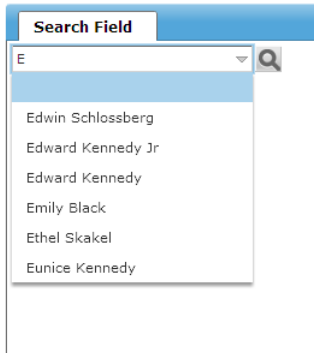
Name	Description
Marker Click	This event is dispatched when the user clicks a marker on the map. The resource corresponding to the clicked marker will be the object which is posted.
Marker Double Click	This event is dispatched when the user double-clicks a marker on the map. The resource corresponding to the double-clicked marker will be the object which is posted.

Listen events

Name	Description
Refresh Display (individual)	Display markers on the map for a given resource if it has geographical information. Currently, the tree, the grid and resources in the form can post this event.
Refresh Display (class members)	Display markers on the map for all instances of a given class with geographical information. Currently, it is best to use the tree to post to this event since it reliably displays classes.

3.13 Search Field

The Search Field component has a single search field. When you set its Lookup Filter attribute to the name of a particular class, it gives your end user lookahead when they type into the field. For example, if you configure its LookupFilter to the class Person with the kennedys data, when your end user types the letter "e" into the search field, all the names beginning with that letter will display in the search field's drop-down list.



Clicking the field's magnifying class icon triggers the Submit Search Post event. For example, let's say you set LookupFilter to the class Person when using the kennedys data with the default applications (note that no Search Field Component is included as part of the default applications, so it must be added from the + menu), and you set the Submit Search event to the ItemOfInterest event. The default applications' Form component has its Refresh Display listen field set to ItemOfInterest, so if your user selects the name Eunice Kennedy in the Search Field and clicks the magnifying glass, the form will display all the information it has on her, including her picture.

3.13.1 Attributes and Events

Attributes

Name	Description	Default Value
Lookup Filter	The lookup will use this attribute (should be a specified rdfs:Class) to determine which values can be selected in the lookup. The default value is rdfs:Resource.	http://www.w3.org/2000/01/rdf-schema#Resource

Post events

Name	Description
Submit Search	Post a resource via clicking on the search button.
Select from Drop-down	Post a resource via selecting it in the dropdown.

Listen events

Name	Description
Change the filter	When posted to, this listen event will update the 'Lookup Filter'. The default value is rdfs:Resource.

3.14 TopBraid Ensemble User Interface Components

This section describes the components that you can use on your TBE forms, pages, and popup windows. See the Component Configuration Fields section below for descriptions of the individual fields used to configure the components.

Examples assume that one of the default applications is loaded with the kennedys.owl data.

3.14.1 Event Button

An event button gives your application's user a simple way to trigger events that can take place in your application, such as the opening or closing of a popup window or the loading of data into another component. Specify the event for it to trigger as a Click Event under Events/Post.

Buttons can be renamed and can listen for events that disable and enable them, if you want a button grayed out under certain conditions. See the Creating Popups section for an example of adding a Button Component to an application.

Attributes and Events

Attributes

Name	Description	Default Value
Button Label	The label to be used on the actual button.	Button
Icon	The amount in pixels that this button can resize to.	
Icon property	The amount in pixels that this button can be resized to when the label changes.	
Tooltip	The amount in pixels that this button can resize to.	
Tooltip property	The amount in pixels that this button can resize to.	
URL	The url or website that will be launched in a new tab on click.	
URL property	The property to use to match against the selected resource to find the 'URL' value.	
Button's Maximum Width	If not using Static width, setting a Max Width on a button's which label changes is useful. The amount in pixels that this button can resize to.	200
Button's Width	The width of the button in pixels.	

Name	Description	Default Value
Icon Width	Set this to the desired width of the icon. please note that not setting this value when using an icon can result in an icon that is too large for the buttons max width.	
Icon Height	Set this to the desired height of the icon.	
Button's Label Placement	The location of the label on the button. This value is only relevant when an icon is specified. The valid options are 'left', 'right', 'top' and 'bottom'.	right
Update Button Label Automatically	Whether or not to update the button's label automatically with the 'Current Resource'.	false
Enabled State	When set to true, the button will be clickable. When set to false, the button will appear grayed out and unclickable.	true
Current Resource	This is the resource that will be passed in 'On Click' post event, and set by the 'Update Current Resource' listen event.	

Post events

Name	Description
Click Event	This event is dispatched when the user single clicks on a button which 'Enabled State' is true. If there is a 'Current Resource', then it will be posted with the event.

Listen events

Name	Description
Update Current Resource	When posted to, 'Current Resource' will be set to that of the resource in the event.
update Tooltip Property	When this listen event is posted to, this button's 'tooltip property' (see icon property) will be updated. Useful for dynamically changing the icon. (For many applications, using this effectively may require the use of a sparql relay to decide on the property).
update Icon Property	When this listen event is posted to, this button's 'icon property' (see icon property) will be updated. Useful for

Name	Description
	dynamically changing the icon. (For many applications, using this effectively may require the use of a sparql relay to decide on the property).
Enable Event	When this listen event is posted to, this button's 'Enabled State' will be set to true.
Disable Event	When this listen event is posted to, this button's 'Enabled State' will be set to false.

3.14.2 Info Box

The Info Box component lets you add text to a form, making it ideal for instructions about how end users should use the rest of the form. To configure this component with hardcoded text, enter a value in its Static Text attribute. For dynamic text, set its Property attribute to the name of a property that holds the text you want displayed. The listen events Update Property and Update Current Resource give you control over when a dynamic value is updated.

To format the text, inline HTML elements such as **b** for bold and *i* for italics are supported, and additional configuration attributes let you control the font size and the spacing around the displayed text.

Attributes and Events

Attributes

Name	Description	Default Value
Padding Left	The padding on the left-hand side of the component.	5
Padding Right	The padding on the right-hand side of the component.	5
Padding Top	The padding on the top side of the component.	5
Padding Bottom	The padding on the bottom side of the component.	5
Static Text	Alternatively from displaying the text of a property of an instance or class(current resource), the infobox can display any text specified here. Please note that the infobox displays html text.	
Property	The property which will be used to get the displayed text from the current resource. i.e. if the property is <code>rdfs:comment</code> , the infobox will display the <code>rdfs:comment</code> for the current resource. note: the text found will override the 'static text'.	http://www.w3.org/2000/01/rdf-schema#comment
Font Size	A number representing the size of the font to use.	13

Name	Description	Default Value
Current Resource	The resource the infobox currently is holding. The resource is used via the 'Property' field to display found text. Use the 'Update Current Resource' listen event or set this value manually.	

Listen events

Name	Description
Update Current Resource	When posted to, this event will update the "Current Resource" value to the passed resource.
Update Property	When posted to, this event will update the "property" value to that of the passed resource.

3.14.3 SPARQL Editor

This component lets your application's users enter SPARQL queries. In the default applications, when the user clicks the "Submit Query" button, the Results Grid component is configured to listen for the query results and display them with a column for each bound variable, as shown here:

The screenshot displays two components from the TopBraid Ensemble interface. The top component is the 'Results Grid', which shows a table of 15 results. The table has three columns: 'first', 'last', and 'born'. The results are as follows:

first	last	born
Patricia	Kennedy	1924
Alfred	Tucker	1967
Linda	Potter	1956
Robin	Lawford	1961
Jeannie	Ripp	1965
Virginia	Bennett	1936
Molly	Stark	1968
Jeffrey	Ruhe	1952

The bottom component is the 'SPARQL Editor'. It has tabs for 'Form', 'Graph Editor and Qu...', and 'Yahoo Map'. The 'Form' tab is active, showing a text area with the following SPARQL query:

```
SELECT ?first ?last ?born
WHERE {
  ?s :firstName ?first;
     :lastName ?last;
     :birthYear ?born
}
```

Below the text area is a 'Submit Query' button.

Attributes and Events**Post events**

Name	Description
Query Results Requested	This event will be posted when the user clicks the 'Submit Query' button and will post the available text as a SPARQL query. (Grid can listen to this)

Listen events

Name	Description
Edit Query	When posted to, this listen event will populate the component with the posted SPARQL in text form.

3.14.4 Basket

The Basket is a holder for resources. It can listen to an event and then add the resource associated with the event to the basket (for example, upon clicking a resource in the grid). You can drag and drop resource into the basket, and you can drag a copy of the resource from the basket to another component.

This component includes a button to remove the selected/highlighted resources from the basket.

Attributes and Events**Attributes**

Name	Description	Default Value
Selected resource	This value is updated by selecting a resource in the basket by clicking on it. Setting the value of this attribute manually will highlight the resource (if the resource is present in the basket).	

Post events

Name	Description
Single-click Selection	This event is dispatched when the user single clicks a resource in the basket.
Double-click Selection	This event is dispatched when the user double clicks a resource in the basket.
Selection Change	This event is dispatched when the selection in the basket has changed. The resource which is now selected will be the resource which is posted.

Listen events

Name	Description
Add a Resource	This listen event will add the passed resource to the basket.

Buttons

Name	Description
Delete Button	Delete the resources selected in the basket.

Drag events

Name	Description
Dragging of a basket item	This event occurs when user releases a dragged node from the basket onto a valid drop target.(i.e. tree, visualization) Adding an event here will enable dragging.

Drop events

Name	Description
Add a resource via drag/drop	This event occurs when user releases a dragged node onto the basket. Adding an event here will enable this as a drop target for the specified event.

3.14.5 Form

A form component displays the data associated with a resource, giving you the TBE equivalent of the main form at the center of TopBraid Composer development. This lets you display information about a particular class, instance, or property, depending on the setting of the form's Item Displayed attribute. This also resets the name of the form displayed on its tab, as shown in the following, where the form has been set to point to the data for Jacqueline Bouvier:

The screenshot shows a web-based form for 'Jacqueline Bouvier'. The form has a blue header bar with the name 'Jacqueline Bouvier' and navigation buttons. Below the header, there are two tabs: 'Person' and 'Family'. The 'Person' tab is selected and displays a photo of Jacqueline Bouvier, along with fields for first name (Jacqueline), MI (L), last name (Bouvier), suffix, gender (female), year of birth (1929), year of death (1994), and alma mater (George Washington University). The 'Family' tab is also visible, showing fields for has spouse (Aristotle Onassis), child (Caroline Kennedy, John Kennedy Jr, Patrick Kennedy), parent, comment, label, middle name (Lee), and name (Jacqueline Bouvier).

See the [Configuring Custom Forms and Grids](#) section for information on customizing the appearance of forms.

The label displayed in the tab for a form component is the label of the currently selected resource, where the label is defined by the TBS dictionary (see). To define a static form tab label, configure the form's "Update label Automatically" attribute to "true" and define the form's label to be the desired tab label.

While the editing of most text fields in the TBE user interface requires you to click on the "edit" button to the left of the field first, you have a greater control over this in the Form Editing Mode attribute of the Form component gives you more control over this. A value of "Review" puts the form in the default behavior, a value of "View" sets the form to a read-only mode for just displaying data, and a value of "Edit" opens up all the fields at once for editing.

Attributes and Events

Attributes

Name	Description	Default Value
Form Definition Class	If not specified, the form will use Form definition provided by TBC to the class which the instance belongs to. Apart from being able to specify search form, or edit form layouts, this property(when specified) allows a configurer to specify a different class's form definition (also defined by TBC) to use for this form.This is especially useful if a resource belongs to more than one class and you want to make sure that it's displayed in TBE with the form designed for one particular class.	
Form Editing Mode	The editing mode of this form. Form modes: View, Edit and Review. View mode has no editing available to the user. Review Mode allows editing, while Edit mode has every widget in 'edit mode' to allow quick editing of all values.	http://server.topbraidlive.org/2008/ensemble#ReviewMode
Displayed Resource	The resource for which information is displayed in the form.	
Include constraint violations	If set to true, a warning will be shown for all values which violate constraints. Please refer to documentation for more about constraint violation.The Getting Started with SPIN tutorial describes how to define and test SPIN constraints.	false
Maximum numbers of values for a widget	The maximum values to display before showing a 'show all' link which will provide a triple match to a grid. See 'Show all subject post'	10
Update Label Automatically	Indicates whether to use the label of the currently displayed resource as the label on the component.	false

Post events

Name	Description
Began editing values	
Changing values complete	
Click Resource Link	Click Resource Link: this event is dispatched when a user clicks a hyperlink in the form. Currently, the

Name	Description
	components which best can accept a resource is the form or search form component.
Constraints are violated	The form is considered valid until a constraint is violated, at which point an event will be dispatched indicating so. This will only be relevant when 'Include constraint violations' is true.
Constraints are satisfied	The form is considered valid until a constraint is violated. Thus, after all violations have been removed, an event will be dispatched indicating that the form is valid. This will only be relevant when 'Include constraint violations' is true.
Show all property post	The event corresponding to the matching Predicate of a Triple Match(Subject, Predicate, Object) which the form uses for the "show all" feature when a widget has more values than the maximum value. A default event has been provided, but the configurer can specify any grid/popup/page to listen to this event.
Show all object post	The event corresponding to the matching Object of a Triple Match(Subject, Predicate, Object) which the form uses for the "show all" feature when a widget has more values than the maximum value. A default event has been provided, but the configurer can specify any grid/popup/page to listen to this event.
Show all subject post	The event corresponding to the matching Subject of a Triple Match(Subject, Predicate, Object) which the form uses for the "show all" feature when a widget has more values than the maximum value. A default event has been provided, but the configurer can specify any grid/popup/page to listen to this event.

Listen events

Name	Description
Update Displayed Resource	This event when posted to will update the 'displayed resource' property and consequently, the view of the form to the resource that was posted. Any event which posts a resource can be used to post to this listen event.
Clear Form	Clear the form of anything its displaying which will remove the 'displayed resource' value.

Name	Description
Update the Form Definition Class	An Event which will update the 'Form Definition Class' to the posted resource (should be a class). This functionality, to automatically update the properties which a form displays for a given instance, is in beta.

Buttons

Name	Description
Backward Button	Navigate backward to the previously displayed item
Forward Button	Navigate forward to the most recent item

Drop events

Name	Description
Drop a in as input to add it as a value.	This event occurs when user releases a dragged node into an input widget on the form.

3.14.6 History Form

A history form lets you see the history of edits made to resources in a model if a Teamwork Repository has been set up for that model. To set one up, you must log in and set up a repository within the model you're going to use.


To log in when using the Personal Edition of TopBraid Live included with TopBraid Composer Maestro Edition, send your browser to the TopBraid Suite Console, which will check the users.n3 file stored in the server.topbraidlive.org \dynamic directory of your TopBraid Composer workspace for valid names. When using EVN with the Enterprise TBL Server on a shared system, the contents of the users.n3 file are updated automatically as each new user on the system logs in to EVN, based on what TBL knows from the TomCat server about authenticated users on that system. (TomCat may rely on LDAP for authentication, or it can be configured to provide basic authentication by itself.)



Teamwork support is only supported for models stored using formats that support persistent blank node identifiers, including Jena SDB, TDB, and Oracle RDF projects. It will not work with a disk text file such as RDF/XML, N3, or Turtle.

To set up a repository for a model, open the model in TopBraid Composer and select **Set up teamwork changes repository** from the **Model** menu. Set the user name to a name in the users.n3 file described above. The best format would be the same one as the model itself is stored in. The display label and short description are used by applications such as EVN when listing models to work with. Fill these out and click **next**, which leads to a dialog box with configuration parameters that depend on the format you've chosen to store the data. After filling this out, click **Finish**.

Once you've finished setting up the teamwork repository, select **Close All** from the TopBraid Composer **File** menu, and then restart TBC-ME.

Adding a History Form to a TBE application that uses this data model is very similar to adding a regular **Form** on page 45. After editing one of the resources and clicking the **Save Data** button, though, clicking the History Form's history button  toggles a display of changes made to that resource. For example, in the following, it shows that

"non-profit" has been added as a "has profession" value, and that the "year of birth" value has been changed from 1947 to 1945:

The **Revert** link next to a removed value reverses the deletion, restoring the original value. The **Replace** link replaces the existing value with the one next to this link. (In the example above, both of these links would have the same effect.)

Along with the same configuration options that a regular Form has, a History Form lets you configure Hide History and Show History listen events so that, in addition to the history button, you can have TBE application logic trigger the displaying and hiding of a resource's edit history.

Attributes and Events

Post events

Name	Description
Change Clicked	Triggered when the history view's "added" or "deleted" links are clicked.

Listen events

Name	Description
Show History	Display the version of the form showing the edit history.

Name	Description
Hide History	Hide the edit history, showing the regular version of the form.

Buttons

Name	Description
Toggle History Button	Show or hide history on the form.

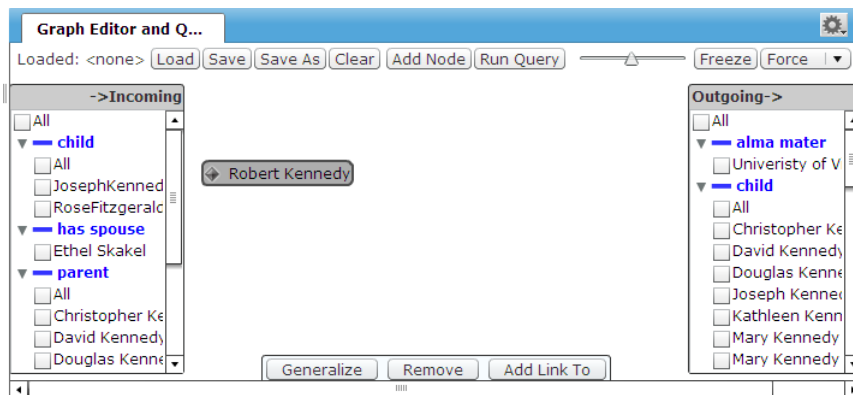
3.14.7 Graph Editor and Query

The graph editor lets your end user graphically generate and run SPARQL queries much like you can on the Graph view of TopBraid Composer. For example:

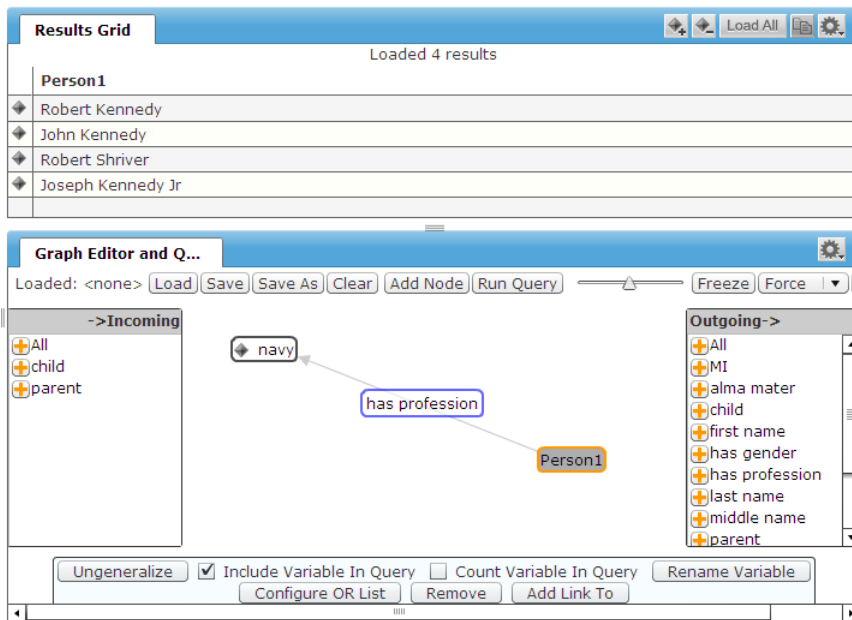
1. Click the Add Node Button, move the cursor to the place in the window where you want the new node, and click there.
2. In the Select Instance message box, enter Robert Kennedy and click OK. You'll see the new node appear.
3. Click the new node, and buttons and list boxes appear so that you can add more to your graph:



You can also drag and drop resources from other components into the Graph Editor and Query canvas provided that the Drag and Drop events have been configured.



4. The Incoming list box lets you build patterns where Robert Kennedy is the object of a triple, and the Outgoing list box lets you build patterns where Robert Kennedy is the subject of a triple. Scroll down to "profession" in the Outgoing list, click "navy", and a new node appears titled "navy" with a labeled line titled "profession" linking the two nodes.
5. Click on the Robert Kennedy button and then the Generalize button underneath that to turn this node into a variable. Now the query represented by this graph asks for everyone in the database who ever had a profession of "navy". In a SPARQL triple pattern form, the query reads "?Person1 :hasProfession :navy".
6. Click the Run Query button, and the results of your query appear in the Results Grid and the query itself appears in the SPARQL Editor component.



As you saw in the Graph Editor after you clicked Generalize, many other buttons appeared to let you graphically modify your query. These let you create even more complex queries without entering any actual SPARQL code.

Graphs (including graphs representing queries) can be saved for future use by clicking on the "Save" button. Saved graphs are displayed by clicking on the "Load" button and selecting the previously saved graph. Saved graphs are saved in the current session. To save a graph so others can choose it from the "Load" button the application must be saved. This will save the graph as part of the application definition that can then be deployed to a TBL Enterprise Server for other users.

Attributes and Events

Attributes

Name	Description	Default Value
Settings	This attribute is for internal use only is under development.	
Image Property	This attribute accepts <code>rdfs:Resource</code> as a property to be used for any given nodes thumbnail image.	

Post events

Name	Description
Node Selected	This event is dispatched when the user selects a node in the graph viz. The resource of the node will be passed. (form, button, relay can listen to this event)
Query Results Requested	This event is dispatched when the user clicks the 'run query' button. Currently, the chart and grid can listen to this event.

Drop events

Name	Description
Drop resource into visualization	This is a listen event which will add the dragged item to the graph visualization as a new node. Currently, the grid and tree can post these drop events.

3.14.8 Results Grid

The Results Grid displays a set of results. These can be the results of a query or they can be all the resources that match the triple with the predicate specified in the grid's Grid Property attribute and the subject or object specified in the grid's Grid Resource attribute, depending on the setting of the 'Use resource as object of triple match?' attribute.

Queries can be provided to the Results Grid in the following way:

- A query can be entered directly in the SPARQL Query field. When a Results Grid component is first displayed it will run the query. A re-run of the query can be triggered by Refresh search results event.
- A query template can be entered in the Query Template field. A query template is a SPARQL query that includes ?this variable. Bind and Execute Query Template event, replaces ?this with the resource that is passed by the posting event and runs the query.
- A query can be passed to the Results Grid by the SPARQL Editor component using Display SPARQL Query event.
- Q query can be generated and passed to the Results Grid by the Search Form and Graph and Query Editor components using Execute Query event..

The results grid can also post data to other components when rows on the grid are selected. See the descriptions of the Tree, Graph Editor and Query, and Yahoo Map components for examples of the Results Grid in use.

See the [Configuring Custom Forms and Grids](#) section for information on customizing the appearance of a results grid.

Attributes and Events**Attributes**

Name	Description	Default Value	Dependency
New resource filter	When adding a resource in the grid using the 'Add' icon, the options to add will be filtered by this value.		
Show number of results loaded	To show or not show the "loaded:x" status box. Setting this to true will display the number of results loaded so far; setting it to false will not show that number.	true	
Grid Object	The 'Grid Object' will be used as the object of a triple match in which the 'Property' is the predicate. The grid will then be populated with the subjects of the matching triples. Setting this will nullify the 'Grid Subject'. Note: posting to this event is equivalent to deprecated listen event 'update resource' when the deprecated value 'resource as object of triple match' was true.		'Grid Property' must have some value. To dynamically reset 'Grid Object' and refresh results, use 'Refresh Grid Object' event.

Name	Description	Default Value	Dependency
Grid Subject	The 'Grid Subject' will be used as the subject of a triple match in which the 'Property' is the predicate. The grid will then be populated with the objects of the matching triples. Setting this will nullify the 'Grid Object'. Note: posting to this event is equivalent to deprecated listen event 'update resource' when the deprecated value 'resource as object of triple match' was false.		'Grid Property' must have some value. To dynamically reset 'Grid Object' and refresh results, use 'Refresh Grid Object' event.
Column resource	<p>This attribute allows you to override the default column layout of a grid populated using either 'Grid Object' and 'Property' attributes (and corresponding events) or 'Grid Subject' and 'Property' attributes (and corresponding events).</p> <p>When a grid's 'Property' attribute = "rdf:type" and the grid is populated using a 'Refresh Grid Object' event, by default the columns of a grid are determined by the column layout for the 'Grid Object' resource as defined in the TBC column layout, which is stored in the .tbc metadata file for a given data file. When the property is different from "rdf:type" and a grid is populated using any of the non-query events ('Refresh Grid Object', 'Refresh Grid Subject', 'Refresh property'), by default the grid will present a single column.</p> <p>The 'Column Resource' feature allows you to override the default column layout with the (specified in corresponding .tbc) column layout of ANY class. Simply set this value to a resource which has the desired column layout.</p>		'Grid Property' and either 'Grid Object' or 'Grid Subject' attributes must have some value set either directly or through the corresponding events.
Property	The grid will be populated with all resources that are either the subject or object of a triple match where 'Property' is the predicate in the triple match.		Either 'Grid Object' or 'Grid Subject' attribute (but not both) must have some value. To dynamically reset Property and refresh results, use the 'Refresh property' event.
Update Label Automatically	If set to true, the tab's label will update to the label of the displayed resource when the displayed resource changes. Otherwise you can define the	false	Should only be used when Property = "rdf:type" and the grid values are

Name	Description	Default Value	Dependency
	component's label manually, and set this property to false.		populated based on the 'Grid Object'.
Selected resource	This value is updated by selecting a resource in the grid by clicking on it. Setting the value of this attribute manually will highlight the resource (if the resource is present in the grid).		
Allow to rearrange columns	Allow the user to drag the columns in order to rearrange them. This is in beta because column layout for the is in development. (rearranged columns do not persist)	false	
SPARQL Query	This value represents the current SPARQL query used to populate the grid. You can enter any SPARQL query in this field. This attribute can also be set via the 'Display SPARQL Query' event.		The 'Display SPARQL Query' event will update this attribute and trigger a rerun of the query. To rerun the query when the attribute is populated using a different approach or when the data has changed while the query did not change, use the 'Refresh search results' event.
Search Criteria	This value represents the SPIN query used to populate the grid. This attribute is set via the 'Execute Query' event. The difference between a SPIN and a SPARQL query is that a SPARQL query is a text string while a SPIN query uses an RDF serialization of SPARQL. It is represented as a node of <code>rdf:type sp:Select</code> , or a SPIN node.		The 'Execute Query' event will populate the attribute and run the query. To rerun the query when the attribute is populated using a different approach or when the data has changed while the query did not change, use the 'Refresh search results' event.

Name	Description	Default Value	Dependency
Query Template Variable	This value represents the node that is to be bound to the ?this variable when the 'Query Template' functionality is executed. This variable is set to the passed resource via the 'Bind and Execute Query Template' event.		Requires some value in the 'Query Template' attribute.
Query Template	The SPARQL query to be used for the Query Template Functionality. The query's WHERE clause must contain a variable ?this which will be bound to the node specified in 'Query Template Variable Binding' attribute.		Requires some value in the 'Query Template Variable Binding' attribute. Works with the 'Bind and Execute Query Template' event.

Post events

Name	Description
Single-click Selection	This event is dispatched when the user single clicks on the row in the grid. The resource which is clicked will be the object which is posted.
Double-click Selection	This event is dispatched when the user double clicks on the row in the grid. The resource which is clicked will be the object which is posted.
Selection Change	This event is dispatched when the selection in the grid has changed. The resource which is now selected will be the object which is posted.

Listen events

Name	Description
Refresh Grid Object	This listen event will update the 'Grid Object' value which is used to populate the grid with a triple match.
Refresh Grid Subject	This listen event will update the 'Grid Subject' value which is used to populate the grid with a triple match. Note: used to be 'update resource'.
Refresh property	When posted to, this listen event will update the 'Property' attribute and repopulate the grid with triples corresponding to the newly passed property.

Name	Description
Refresh search results	This listen event will refresh the grid if it has been populated via a SPARQL/SPIN query. If the grid is populated with the 'Grid subject' or 'Grid object' attributes, triggering this event will have no effect.
Execute Query	When posted to, this listen event will populate the grid with the results of a SPIN query. (Search Form and Graph Visualization post these queries).
Display SPARQL Query	When posted to, this listen event will populate the grid with the results of a SPARQL query. (SPARQL editor post these queries).
Bind and Execute Query Template	This listen event, when posted to will trigger the query template functionality by setting the 'query template' variable to the passed resource, and executing the query specified in the 'Query Template'.
Update the Column Resource	Change the column resource to that given in the event. Note: when specified, the column resource will always be used to determine which columns show in the grid.
Clear selection(s)	Deselect any selected items in the grid.

Buttons

Name	Description
Create New Button	Add new resource.
Delete Button	Remove the resources selected in the grid.
Load All Button	Load all matches into the grid.
Copy Button	Copy the selected entries to the clipboard for pasting into another application (e.g. to a spreadsheet).

Drag events

Name	Description
Dragging of a grid entry	This event occurs when user releases a dragged node from the grid onto a valid drop target.(i.e. tree, visualization) Adding an event here will enable dragging.

Drop events

Name	Description
Refresh display via drag/drop	This event occurs when user releases a dragged node onto the grid. The grid will show subjects related to the dragged resource. Adding an event here will enable this as a drop target for the specified event.

3.14.9 Tree

The Tree component displays data in a tree. The end user can expand and collapse branches of the tree for viewing and can add and delete new nodes if your configuration of the tree allows them to.

In the default applications, the clicking of a tree node posts the event `ClassOfInterest`. The Search Form and Results Grid components listen for this information and populate their display with the relevant data. For example, with the kennedys data, if you click the Person class on the Tree component, you'll see the instances of that class on the Results Grid, and the Search Form will become a Search Form for Person data, with all the relevant properties:

The screenshot shows the 'Default Application' interface. At the top, there are buttons for 'Save App', 'Save Data', and a search bar. Below this, the interface is divided into three main sections:

- Tree:** A hierarchical view of data. The 'Person' class is selected, showing 74 instances. Below it, 'Matriarch' (1) and 'Patriarch' (1) are listed.
- Search Person:** A form for searching Person data. It includes fields for 'MI', 'age', 'alma mater', 'child', 'first name', 'gender', and 'last name'. A 'Submit Query' button is at the bottom.
- Results Grid:** A table displaying 26 results of 74 total. The results are listed in a table with a 'Name' column and a diamond icon next to each entry.

Attributes and Events**Attributes**

Name	Description	Default Value
Treat property objects as parents?	Setting this attribute to true will display a tree in which each parent-child relationship represents a triple in which the parent is the object and the child is the subject; setting this attribute to false displays a tree in which each parent-child relationship represents a triple in which the parent is the subject and the child is the object.	true
Treat nodes as object of count statements?	Count queries (which are just triple matches) use the counts property and the node as the subject of the triple match. When using 'show tree in reverse', you may need to toggle this value to use the node as the object instead.	true

Name	Description	Default Value
Default Expanded	Change this attribute so that each time the tree is populated, all nodes will be expanded(to a depth level of 4).	false
Root Node	The root node which the tree is currently displaying. All nodes in the tree are populated by recursively executing a triple match over found nodes beginning with the root node.	http://www.w3.org/2002/07/owl#Thing
Transitive Property	an rdfs Property (e.g. rdfs:subClassOf) by which to use for the triple match that relates child-parent nodes in the tree in the tree.	http://www.w3.org/2000/01/rdf-schema#subClassOf
Counts Property	an rdfs Property (e.g. rdfs:type) by which to find instances of and use to provide count information(e.g. Thing(45)).	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
Update label Automatically with root	Setting this to true will automatically include the 'root node' in the label. Not mutually exclusive to 'Update label Automatically with property'	false
Update label Automatically with transitive property	Setting this to true will automatically include the 'transitive property' in the label. Not mutually exclusive to 'Update label Automatically with root'	false
Display the search field	Include a convenience search field to quickly find and highlight items in the tree (helpful for large class hierarchies).	false
Show the root of the tree in the display	Toggle this feature to true in order to make the root node visible/hidden in the tree.	false
Selected resource	The currently selected item in the tree.	
Sorting property for nodes in the tree	A rdfs Property (i.e. rdfs:Label) for the tree to use when sorting items. If no property is specified, then the displayed label will be used to sort the nodes.	

Post events

Name	Description
Single-click Selection	This event is dispatched when the user single clicks on an item in the tree. The resource which is clicked will be the object which is posted.
Double-click Selection	This event is dispatched when the user single clicks on the row in the grid. The resource which is clicked will be the object which is posted.
Selection change	This event is dispatched when the 'Selected Item' in the tree is changed. The resource which is clicked will be the object which is posted.

Listen events

Name	Description
Switch Root	Post to this listen event to set the 'Root Node' in the tree to the passed resource.
Highlight a node	Post to this listen event to set the 'Selected Item' in the tree to the passed resource (if that resource exists in the tree).
Change the transitive property	Post to this listen event to dynamically change the 'Transitive Property' to the passed resource.

Buttons

Name	Description
Add Child Button	Add a child node to the selected node.
Add Sibling Button	Add a sibling node to the selected node.
Delete Button	Delete the selected node.

Drag events

Name	Description
Drag a tree resource	If you use this event, it will enable dragging of a tree resource. The event will be dispatched when the user begins dragging the dragged node.(the tree, grid, or visualization, if it has a drop event configured to listen to the drag event). The resource posted will be the dragged item.

Drop events

Name	Description
Drop resource into tree	This event is fired when a dragged node is dropped onto a valid drop target. To make a drop target a valid drop target for a given drag, simply configure the 'drop event' to be the same event as the 'drag event'. Currently, only the tree and grid can post to this event.

3.14.10 Search Form

The Search Form displays a form for a particular class and, after the end user clicks the Submit Query button, posts the result. In the default applications, the Results Grid listens for and displays the result of the search, as shown here:

Default Application

Save App Save Data

Search Person

Clear Fields

Other Properties

MI :

age :

>

<

▶ alma mater :

▶ child :

first name :

gender :

last name :

Submit Query

Results Grid

Load All

Loaded 5 results of 5

Person

Robert Kennedy Jr

Robert Kennedy

Robert Shriver III

Robert Pender Jr

Robert Shriver

Search form fields that represent relationships between resources (object properties) can be expanded to reveal nested form. This enables a user to formulate richer queries by setting search criteria not only for the direct properties of the selected resource type, but also for the properties of related resources. For example, one could create a search that finds all Roberts (direct property of resources of type Person) that went to an Ivy League school (property of related resources of type College).

Another useful feature of the Search Form is ability to identify what columns should be shown in the result. This is done by checking a checkbox to the left of each field. Search form must be configured to show the checkboxes by setting "Include Checkboxes" attribute to true.

Default Application

Save App Save Data

Search Person

Clear Fields

Person

photo :

first name :

MI :

last name :

suffix :

has gender :

☒ year of birth :

>

<

☐ year of death :

>

<

☒ alma mater :

is college ivy league? :

latitude :

longitude :

label :

has profession :

Family

has spouse :

Submit Query

Results Grid

Load All

Loaded 3 results of 3

Person

Person year of birth

Person alma mater

Robert Kennedy Jr

1954

Harvard University

Robert Shriver

1915

Yale

Robert Shriver III

1954

Yale

See the [Configuring Custom Forms and Grids](#) section for information on customizing the appearance of search forms.

Attributes and Events

Attributes

Name	Description	Default Value
Displayed Resource	The resource for which to display the input form for.	
Include Resource ID in SPARQL	Include the found resources (of type Search Type) as a variable in the generated results.	true
Update Label Automatically	If set to true, the tab's label will update to the displayed resource when the displayed resource changes. Otherwise you can define the component's label manually, and set this property to false.	false
Include Nested Forms	If set to true, the search form will show expandable and collapsible nested forms for more complex searches.	true
Include Checkboxes	If set to true, the search form will show checkboxes which allow the user to exclude or include search items.	false

Post events

Name	Description
Submit Search	This event will post a query event when a user clicks on the "Submit Search" button. Currently, the grid is best supported to listen to this event.

Listen events

Name	Description
Refresh Display	Updates display to show a search form which corresponds to the posted resource. Currently, the tree, grid, and clicking a resource in a form are best supported for posting to this event.

Buttons

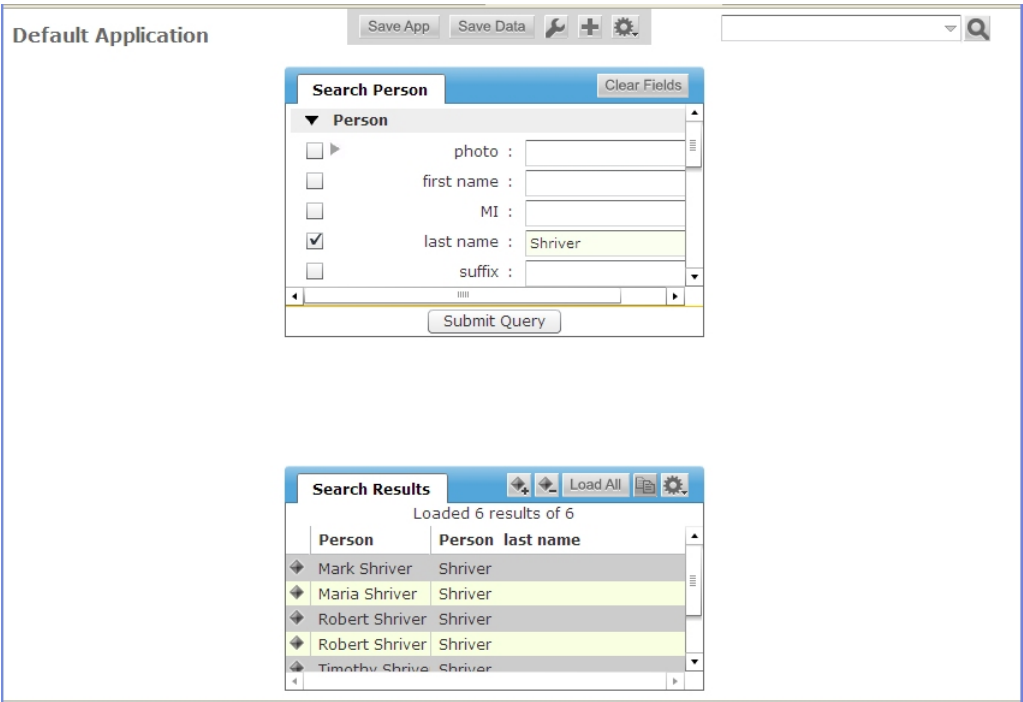
Name	Description
Clear Fields Button	Clear all fields in the search form.

Drop events

Name	Description
Drop a in as input to add it as a value.	This event occurs when user releases a dragged node into an input widget on the form.

3.14.11 White Space

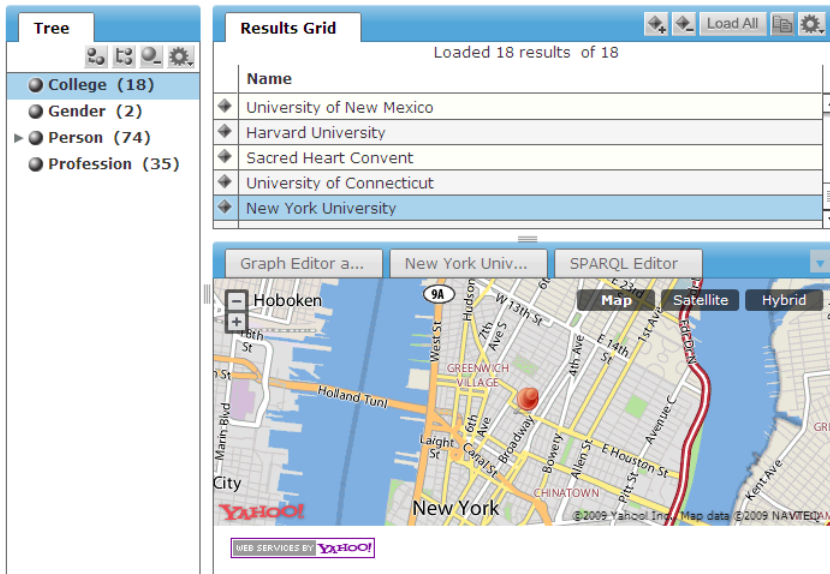
White space components give you greater freedom in how you arrange your application's components. Because you can hide the display of any component's tab by clicking next to the tab, white space components can be arbitrarily-size blank rectangles that you use to space out other components. For example, the following has a white space component on the left, one on the right, and one between the Search and Search Results components shown:



Attributes and Events

3.14.12 Yahoo Map

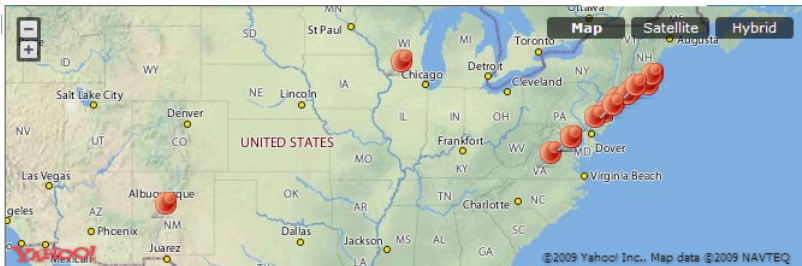
This zoomable map uses Yahoo's API to display one or more points on a map when given the latitude and longitude of those points. For example, when using the default applications with the kennedys data, click the College node on the Tree to list the data graph's colleges on the Results Grid Component. Click one of the college names there and a map will appear in the Yahoo map component with a red pushpin showing the location of the college.



This happens because in the default applications the Results Grid posts the event `ItemOfInterest` when a row of the grid gets single-clicked, and the Yahoo Map component listens for `ItemOfInterest` and refreshes the display upon receiving this event if the data accompanying the event has longitude and latitude information.

The map can also display the members of a class. In the Configuration Console for the default applications, you can see that the Yahoo Map has two options for events to listen for: "Refresh Display (class members)" and "Refresh Display (individual)." The latter is set to both `ClassOfInterest` and `ItemOfInterest`, which is how it displayed the map for the individual college that you selected above.

Because a single click to a node of a Tree Component posts the `ClassOfInterest` event, if you add `ClassOfInterest` as a "Refresh Display (class members)" event for the Yahoo Map to listen for, clicking class names in the Tree Component that have any latitude and longitude information associated with their members will display a map with push pins at those locations. The Person class only has one member with this information, California governor Arnold Schwarzenegger, so a pin is displayed on Sacramento. The College class has many more members with this data, so the map displays a larger portion of the United States with many more pins displayed.



For classes with a large number of members that have latitude and longitude data, configuring the Yahoo Map to display all of their locations can lead to performance issues, so use it with care.

Attributes and Events

Attributes

Name	Description	Default Value
Yahoo Application ID	A developer yahoo map api key. Please get one at http://developer.yahoo.com/maps/simple/	Xg_nVozV34FE7rSC4h7v4.QKEpnCe7FA8
Mapped Class	Resources of this type are marked on the map.	
Mapped Resource	This resources is marked on the map.	
Display Info for Selected Marker	When a marker is clicked, display it's information on the map.	true

Post events

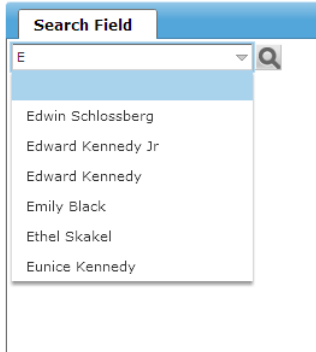
Name	Description
Marker Click	This event is dispatched when the user clicks a marker on the map. The resource corresponding to the clicked marker will be the object which is posted.
Marker Double Click	This event is dispatched when the user double-clicks a marker on the map. The resource corresponding to the double-clicked marker will be the object which is posted.

Listen events

Name	Description
Refresh Display (individual)	Display markers on the map for a given resource if it has geographical information. Currently, the tree, the grid and resources in the form can post this event.
Refresh Display (class members)	Display markers on the map for all instances of a given class with geographical information. Currently, it is best to use the tree to post to this event since it reliably displays classes.

3.14.13 Search Field

The Search Field component has a single search field. When you set its Lookup Filter attribute to the name of a particular class, it gives your end user lookahead when they type into the field. For example, if you configure its LookupFilter to the class Person with the kennedys data, when your end user types the letter "e" into the search field, all the names beginning with that letter will display in the search field's drop-down list.



Clicking the field's magnifying glass icon triggers the Submit Search Post event. For example, let's say you set LookupFilter to the class Person when using the kennedys data with the default applications (note that no Search Field Component is included as part of the default applications, so it must be added from the + menu), and you set the Submit Search event to the ItemOfInterest event. The default applications' Form component has its Refresh Display listen field set to ItemOfInterest, so if your user selects the name Eunice Kennedy in the Search Field and clicks the magnifying glass, the form will display all the information it has on her, including her picture.

Attributes and Events

Attributes

Name	Description	Default Value
Lookup Filter	The lookup will use this attribute (should be a specified rdfs:Class) to determine which values can be selected in the lookup. The default value is rdfs:Resource.	http://www.w3.org/2000/01/rdf-schema#Resource

Post events

Name	Description
Submit Search	Post a resource via clicking on the search button.
Select from Drop-down	Post a resource via selecting it in the dropdown.

Listen events

Name	Description
Change the filter	When posted to, this listen event will update the 'Lookup Filter'. The default value is rdfs:Resource.

Chapter

4

TopBraid Ensemble Event Rules

Topics:

- [Deep Linking Component](#)
- [SPARQL Rule](#)
- [SPARQLMotion Relay](#)

This section describes the event rules that you can add to your applications to automate the execution of various kinds of logic.

4.1 Deep Linking Component

A Deep Linking rule lets you start up a TBE application with a URL that brings it up in a particular state. The rule modifies the URL in your browser's navigation toolbar so you can copy it and use the URL to invoke the application in a state specified by the rule. Add a Deep Linking Rule to your TBE application by selecting **Add Event Rules** from the + menu and then **Deep Linking Component** from the cascade menu that appears.

To configure the Deep Linking Component, set its Relevant Event attribute to a particular event. When this event is posted, the URL in the browser navigation toolbar will change to reflect the event. When the URL is copied from the toolbar and used in a different browser session, the Ensemble application will open the specified data graph and react as if that event had just been posted with the named resource, and all the components listening for that event will immediately take the appropriate action.

For example, let's say you created a TBE application and saved it with the name `DeepLinkingTest`. This application has a Deep Linking rule whose Relevant Event is set to `ViewKennedy`, and it also has a form component whose Update Displayed Resource listen event is also set to `ViewKennedy`. If you had a results grid with a post event of `ViewKennedy`, then when someone using the application clicked on Aristotle Onassis's name on the grid, the form would display information about Aristotle Onassis and update the URL in your browser's navigation toolbar. This URL can be used to start up this application with the `kennedys.rdf` data and the Aristotle Onassis data already displayed. Using TBL Personal server, the URL will look something like this (split here to fit on the page):

```
http://localhost:8083/tbl/app/server.topbraidlive.org/
user-applications/DeepLinkingTest.n3/-/TopBraid/Examples/
kennedys.rdf#TBE:viewKennedy=http://topbraid.org/examples/kennedys#AristotleOnassis
```

The URL specifies the following:

- The URL for the application, "`http://localhost:8083/tbl/app/server.topbraidlive.org/user-applications/DeepLinkingTest.n3/`".
- The path of the data graph within the Live server's workspace, "`/TopBraid/Examples/kennedys.rdf`".
- The name of the event, "`TBE:viewKennedy`".
- The resource passed to the event, "`http://topbraid.org/examples/kennedys#AristotleOnassis`".

The result of pointing a browser to this URL (via a hyperlink, for example) is that the application will be opened to the specified data graph and the event will be posted. In this example, application would be opened with Aristotle Onassis's information displayed in the form.

Note that the application can be opened without any data graph chosen by using the URL for the application. Selecting the application to show the `kennedys.rdf` data graph is accomplished by copying the application URL and the data graph path.

4.2 SPARQL Rule

A SPARQL rule listens for an event, performs a SPARQL query specified as part of the rule's configuration with the resource URL passed along with the event, and then posts another event with the result of the query. Add one to your TBE application by selecting **Add Event Rules** from the + menu and then **SPARQL Rule** from the cascade menu that appears.

As an example, picture a SPARQL Rule whose variable attribute is set to the value `in` and whose SPARQL attribute has the following query as its value:

```
PREFIX simple:<http://topbraid.org/examples/kennedyfamily#>
SELECT ?out
WHERE {
  ?in simple:almaMater ?out
}
```

If this rule has its Input listen event set to `ItemOfInterest` and its Relayed Event post event set to `CollegeOfInterest`, then if another component posts an `ItemOfInterest` event with a resource URI of `http://topbraid.org/examples/kennedys#ArnoldSchwarzenegger`, the rule would "hear" this `ItemOfInterest` event and use this URI for the `?in` value. Executing the query with the kennedys data included with TopBraid would set `?out` to `http://topbraid.org/examples/kennedys#UniversityOfWisconsin`, which would be passed along with the `CollegeOfInterest` event that this rule was configured to relay.

If this TBE application had a form that, like this SPARQL rule, was listening for the `ItemOfInterest` event, and it had another form that was listening for the `CollegeOfInterest` event, then when the application's user did something to trigger the `ItemOfInterest` event with the `http://topbraid.org/examples/kennedys#ArnoldSchwarzenegger` URI (for example, clicking Schwarzenegger's name on a results grid that was configured to post that event), the first form would display all the graph's information about Schwarzenegger and the second form would display all the graph's information about the University of Wisconsin, Schwarzenegger's alma mater.



When a posted event triggers this rule, The Relay Input attribute (under Relay/Attributes on the configuration window, not under SPARQLRule/Attributes) gets set to the URL passed with the event. If you then save the application, then the next time you open it, this attribute will still be set to this value, and the query will be automatically triggered with the variable specified in Variable name set to this value, and the result will be posted in the event specified as the Relayed event. This is useful for executing a SPARQL query that executes setup logic for your application.

4.3 SPARQLMotion Relay

A SPARQLMotion relay executes a SPARQLMotion script when a particular event is posted. Add one to your TBE application by selecting **Add Event Rules** from the + menu and then **SPARQLMotion Relay** from the cascade menu that appears.

This rule has two key pieces of information to configure:

- Set its listen event to the event that should trigger the execution of the script.
- Set the relay's SM Script ID attribute to the URL of the SPARQLMotion function that has the SPARQLMotion script as its `sm:returnModule` property.

You can also configure the relay to post another event when it is triggered.

For example, if the function with the URI `http://server.topbraidlive.org/dynamic/user-sessions/funcTest#func1` points to a SPARQLMotion script that you want executed whenever your application's user double-clicks a node on a tree component, you could add an event with a name like `TreeDoubleClick` as the tree component's Double-Click Selection post event. If you configured a SPARQLMotion relay with `TreeDoubleClick` as its Input listen event and `http://server.topbraidlive.org/dynamic/user-sessions/funcTest#func1` as the value of its SM Script ID attribute, then whenever a node of that tree was double-clicked, the function would be called, and it would execute the SPARQLMotion script that it pointed to.



When you set the Relay Input attribute (under Relay/Attributes on the configuration window, not under SPARQLRule/Attributes) to a resource—even `rdfs:Resource`, which will show up as **Resource** when you click the + sign and select **Add Existing Entry**—the script named in the Script ID value will be executed automatically when this TBE application is started up. This is useful for executing a SPARQL query that executes setup logic for your application.

Chapter

5

TBE Architecture

Topics:

- **Customizing Component Interactions: the TBE Event Bus (DAG)**

The available customization options for TBE applications are defined in a model represented as a TBE configuration ontology. This configuration ontology is a combination (merge) of the four primary models identified below (and provided with TopBraid Suite), along with the RDF files for each component used by the application. Application developers may want to inspect these models to see how customizations to TBE applications are represented and stored. For reference, the four primary models are:

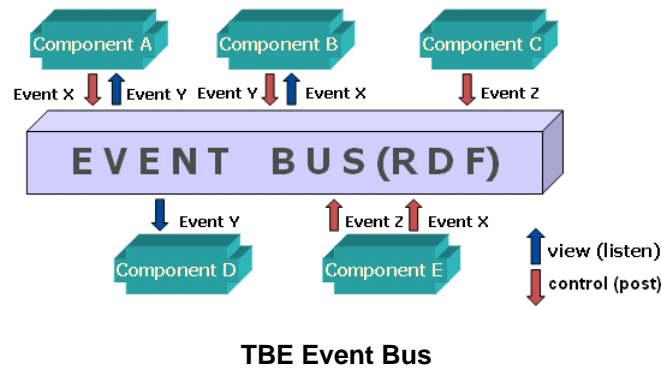
- base URI: <http://server.topbraidlive.org/2008/ensemble>; stored in the file: `ensemble.rdf`; in the folder: `server.topbraidlive.org\web\2008`.
- base URI: <http://server.topbraidlive.org/2008/live>; stored in the file: `live.rdf` in the folder: `server.topbraidlive.org\web\2008`
- base URI: <http://server.topbraidlive.org/2008/11/tbe-ex> ; stored in the file: `org.topbraidlive.tbe.rdf`; in the folder: `server.topbraidlive.org\dynamic\components`
- base URI: <http://server.topbraidlive.org/2008/11/tbe> ; stored in the file: `org.topbraidlive.component.rdf`; in the folder: `server.topbraidlive.org\dynamic\components`

5.1 Customizing Component Interactions: the TBE Event Bus (DAG)

TBE application customizations are stored as triples forming a Dynamic Application Graph (DAG). The application graph is dynamic in two ways:

- At customization time, an application graph is created and changed by the application designers as they modify the configuration options
- At run time, as users interact with the application, the application graph gets updated with events:
 - Each component can either post an event or listen to an event
 - Events are coordinated by Event Types

While a TBE application runs, its DAG is saved in a TBLxxxxx.n3 file. At application run time, the application graph serves as an Event Bus. Each component can post one or more events to the bus and/or listen to one or more events, as shown in the following diagram:



When an application user takes an action (for example, clicking, double-clicking, or pressing Enter), a component in which this action takes place determines which event types are triggered by the action. Correlation between event types and actions are stored in the manifest file for each Component Type.

An event is posted to the Dynamic Application Graph (Event Bus) as two triples (represented here in n3 notation; "a" is a short hand for `rdf:type`):

```
_:e a ?EventType.
_:e rdf:value ?resource .
```

The first triple identifies the type of the event being posted and the second specifies information being passed along with the event for the listening component to act on. For example, when a node in a tree is clicked, the value associated with that particular node (for example, the URI associated with the class `College`) is passed along as the value. If another component is listening for that particular event, it can use that value to display related information (for example, a list of members of the `College` class) as part of that component's layout.

A component that is listening for events checks for the following triple pattern in the DAG:

```
_:e a ?WatchedType .
```

Once a component detects that an event it is listening for has occurred, it performs an operation. For example, it loads and displays some information related to the value passed in the triple with the `rdf:value` predicate above.

In addition to the components that have a visual representation, TBE also provides an "invisible" component called a relay. Relays are not displayed on the form, but can be used to listen for events and perform some transformations and computations in order to generate and post an event.

Chapter

6

Integrating SPARQLMotion Scripts with TBE Applications

Topics:

- [Defining SPARQLMotion Scripts](#)
- [Using Ensemble Events to Invoke SPARQLMotion Scripts](#)
- [Invoking SPARQLMotion Scripts on Ensemble Components](#)
- [Using the SPARQLMotion Debugger in TBL Personal Server](#)

TopBraid Ensemble events can invoke server-side processing that allow application developers to harness the power of SPARQLMotion in their TBE applications. Executing a SPARQLMotion script on a TopBraid Live server can be performed by any Ensemble event through SPARQLMotion Relays. SPARQLMotion scripts can also be associated with some TBE components, such as Trees, Grids and Forms. Virtually any operation that can be specified in SPARQLMotion is available; typical examples include resetting values of properties such as labels or level numbers on selected tree nodes, importing and exporting data, defining drag-and-drop semantics, deleting creating new resources where deletion and creation require execution of special business logic, and many other circumstances where specialized processing is needed.

Scripts invoked by Ensemble are executed on a TopBraid Live server. If a script requires extra information, a dialog box will prompt for it at run time. Any changes to the data model as a result of running the script will be reflected in the display as soon as the script completes.

6.1 Defining SPARQLMotion Scripts

This section describes how to create a SPARQLMotion script that can be invoked by Ensemble. TopBraid Composer is used to define the scripts that will be executed on the server in response to Ensemble requests. All scripts and interaction between Ensemble and server-side scripts are designed and tested on TBC-ME, which includes the TBL Personal Server, and can be deployed to a TBL Enterprise server through Export... > Deploy in TBC-ME.

To learn more about the full range of capabilities of SPARQLMotion scripting, including tutorials and examples, please see the [SPARQLMotion Home Page](#).

When designing a script for use in TBE, there are five main parts to consider:

- *The name by which the script will be known in TBE.* TopBraid Live uses the SPARQL Inference Notation (SPIN) to publish custom functions outside the server as web services or as functions available for TBE applications. SPIN functions can either define SPARQL queries or link to a SPARQLMotion script to perform the service.
- *A specification of what information is required by the script.* Parameters can be passed as arguments to SPIN functions. Additional information may be collected using the SPARQLMotion modules that accept user input. This approach is particularly useful when the type of the input needed must be determined dynamically as part of script execution.
- *A specification of the script itself.* More information on creating SPARQLMotion scripts can be found in the [SPARQLMotion home page](#) and the SPARQLMotion Tutorial.
- *Pass data back to the Ensemble client data graph.* To assert new information in the Ensemble application, the script will need to execute a PerformUpdate operation to insert or delete triples to the session graph used by Ensemble, <http://tb-session>.
- *The script must be registered with the TopBraid Live server.* This is accomplished by placing the SPIN function and SPARQLMotion script in a file with a .sms extension.

Creating a SPARQLMotion script for use in Ensemble

In the following we step the reader through the creation of an example SPARQLMotion script that can be invoked from Ensemble. This is a limited example of SPARQLMotion to help the reader understand how server-side scripts can be applied in Ensemble. Please see the [SPARQLMotion home page](#) for examples and tutorials.

The example will create a simple script that uses regular expression patterns to transform a string from one format to another. An example with family names will be used. Subsequent sections will show how this script is applied in Ensemble through SPARQLMotion Relays and Ensemble component scripts.

1. *The script must be defined in a file with a .sms extension to be registered with TopBraid Live.* To create a file to store the SPIN function for the action, use the "SPARQLMotion File" option from the New> menu in TopBraid Composer's Navigator pane; this will import all the appropriate system files you will need.

Make sure to check the "Script will declare (Web) Services or Functions (.sms extension)" checkbox on the Create SPARQLMotion File dialog box. If this was not done, the file can be changed to register script by changing the file name to include ".sms" and refreshing SPARQLMotion functions. Give the file any name you want and save in any of the available file serializations (.rdf, .owl, .ttl, etc.).

2. *Define a SPIN function.* Create a subclass of spin:Functions (a subclass of spin:Modules) and name it "CreateFormattedLabel". The URI of this class definition will be the URI of the SPIN function.

Configure the appropriate parameters for your function. Three parameters are of particular importance when integrating SPARQLMotion scripts into a TBE application:

- *rdfs:label* will appear as the display name for the function when invoked from Ensemble.

- *spin:constraint* specifies an argument to pass to the function. You can define as many of these as you like.
 - *sm:returnModule* links the SPIN function to a SPARQLMotion script. Specifically, the returnModule property specifies the terminal SPARQLMotion module in the script that will implement the function.
3. *Create two properties to be used as arguments to the function.* These will define the source and target properties for the transformation. Click on the property *sp:arg* and create subproperties named "*sp:sourceProp*", "*sp:targetProp*". Any property can be used as an argument, but it is often convenient to define these as subproperties of *sp:arg* to keep track of your arguments.
 4. *Add arguments to the SPIN function.* Go back to the form view for the class you created, "CreateFormattedLabel". Add "*sp:sourceProp*" as a *spin:constraint* for the class. This can be done either by a) dragging *sp:sourceProp* from the property view on top of *spin:constraint* or b) choosing the context menu (small triangle) next to *spin:constraint* and entering "*sp:sourceProp*" in the predicate field of the template.

Fill the *valueType* field with "*rdf:Property*" as the argument will be a property. For this example, leave the default value blank. Filling in the comment field will display a label when the arguments are displayed to the user. The template should look like the following screen image.

Add "*sp:targetProp*" as a *spin:constraint* in the same way. For the default value, use "*rdfs:label*".

Create from SPIN template...

Available Ask/Construct Templates:

- sp:Argument
- sp:Attribute
- sp:InferDefaultValue
- sml:SelectedResourceArgument

Arguments:

sp: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 defaultValue. (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)

SPARQL Expression:

```

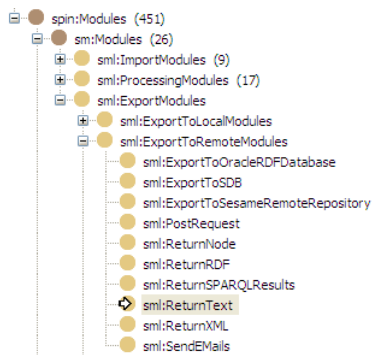
CONSTRUCT {
  ...b0 a spin:ConstraintViolation .
  ...b0 spin:violationRoot ?this .
  ...b0 spin:violationPath ?predicate .
}
WHERE {
  FILTER (isIRI(?this))
}

```

OK Cancel

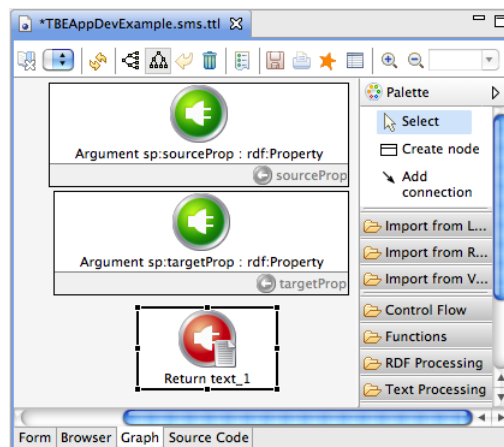
Argument template for *sp:sourceProp*

5. *Define a SPARQLMotion script and link it to the SPIN function.* The body of the SPIN function can be defined by a SPARQL query, using *spin:body* (a SPARQL query), or a SPARQLMotion script, using *sm:returnModule* (alternatively a JavaScript function if SPINx is imported into the model). To create and attach a new SPARQLMotion script, choose the context menu next to *sm:returnModule* and choose "Create and add..." The screen shown below will appear after opening the tree view. In this example we may want to display a pop-up window that alerts the user when the script is completed. Therefore, choose *sml:Return text* to create an instance of that module. This will create a module (i.e. a SPARQLMotion script with one module) and populate that module's URI in the *sm:returnModule*.



If you already have a SPARQLMotion script, choose "Add Existing" and choose the terminal module of the script.

To edit the script that corresponds to the SPIN function just created, select Scripts > Edit SPARQLMotion Script and select the return module you created for this purpose. The corresponding arguments will appear in the SPARQLMotion editor along with the return module. The following shows a return module and the two argument modules created for this example.



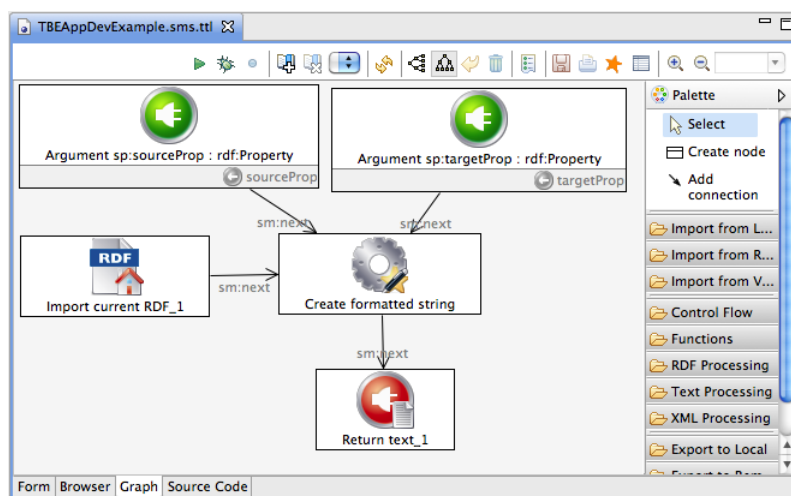
SPARQLMotion modules resulting from the SPIN function definition

Edit the script to perform the actions you want to take place by adding additional modules to the script. A few hints:

- Use the module type ImportCurrentRDF to refer to whatever model is loaded at the time the script is run. Most scripts running from Ensemble will include this module.
- If your SPARQLMotion script uses an Apply Construct module, it creates new triples that will only be accessible for other modules in the script while it runs. To create new triples that will be accessible in your TBE application—for example, to be loaded into a Tree or Results Grid component—use an INSERT statement in a Perform Update module, and insert the triples into the <http://tb-session> graph that stores the current session.

- If your script ends with a Return Text module, setting its `sml:mimeType` property to `text/html` means that if you set the `sml:text` property to the contents an HTML document, TBE will display the HTML to the TBE application's user in a pop-up browser window, giving you great flexibility in how your application presents information.

In our running example, use the palette to create an instance of PerformUpdate. You can find in the RDF Processing category. Click on PerformUpdate and click in the workspace. You will be asked for a name for the module instance and it will be created. Then use Add Connection to link each of the arguments to the PerformUpdate module and the PerformUpdate module to the ReturnText module. The script pipeline should look like the figure below. The script will bind values entered into the arguments that will be available to the PerformUpdate module (named "Create new label" in the figure).



Completed script

The last module in the script is an instance of the ImportCurrentRDF module. This is found in the palette under the Import from Various category. This will import the data in the current Ensemble session, the graph `<http://tb-session>` into the script.

To complete the script, enter the following query by double-clicking on the PerformUpdate module, selecting Add empty row for the property `sml:updateQuery`, and entering the query below. This example query that transforms strings using standard XML regular expressions. For example, the string "Rose Fitzgerald" can be turned into "Fitzgerald, Rose". There's no need to become an expert in regular expressions. This is just one of many examples of how to transform data using TBS and SPARQL queries.

```
# Detects three patterns to transform. Examples in order of evaluation:
# "Edward Kennedy Jr" --> "Kennedy, Edward Jr"
# "Kathleen H. Kennedy" --> "Kennedy, Kathleen H."
# "John Kennedy" --> "Kennedy, John"
INSERT INTO <http://tb-session> {
  ?rsc ?targetProp ?newname .
}
WHERE {
```

```

?rsc ?sourceProp ?name .
LET (?newname := smf:firstBound(smf:regex(?name,
                                "([A-z]+) ([A-z]+) ([A-z]+)", "$2, $1 $3", ""),
                                smf:regex(?name, "([A-z]+) ([A-z]+\.\.) ([A-z]+)", "$3, $1 $2", ""),
                                smf:regex(?name, "([A-z]+) ([A-z]+)", "$2, $1", ""))) .
}

```

This query will find all instances of the source property passed in as an argument and apply a regular expression function to arguments passed into it. The string created by the regular expressions is used to build a triple with the new name attached to the target property for each resources matched in the WHERE clause. The triples generated are inserted into the <http://tb-session> graph and the data will be displayed immediately to the Ensemble user. This will be shown in the next section.

Although the scope of this manual is not to describe SPARQL queries or regular expression in any detail, the query does bear some explanation. The `smf:firstBound()` function will evaluate each of the parameters in order. When a binding is found, it is returned and bound to `?newname` in the `LET` clause. In this case `smf:firstBound()` is used as a kind if-then-else construct. The `smf:regex()` function specifies a string, a match pattern, a pattern for string replace, and a replace string if no match occurs. Since the last parameter (replace string if no match occurs) is specified as an empty string, no binding is passed. I.e. if the first `smf:regex()` parameter to `smf:firstBound()` fails to match the pattern `"([A-z]+) ([A-z]+) ([A-z]+)"`, no binding is passed and `smf:firstBound()` will try the next match. Regular expressions are a topic beyond the scope of this manual (see [F. Regular Expressions, XML Schema Part 2: Datatypes Second Edition](#)), but a general explanation of the example will help the reader understand how this works. The pattern `"([A-z]+) ([A-z]+) ([A-z]+)"` has three subexpressions, denoted by the parenthesis. When these match, they are bound to variables named \$1, \$2, \$3, etc. in order of appearance of the subexpression. The string replace pattern `"$2, $1 $3"` will place the match from the second subexpression first followed by a comma, a space, and the first match. Then the third match is placed in the string followed by a space and the third match. So a string such as "Joseph Kennedy III" is replaced by "Kennedy, Joseph III".

6. When running TBL Personal server, choose Scripts > Refresh/Display SPARQLMotion functions... to register the script. Scripts are also registered when opening a project or deploying a project to TBL Enterprise Server.

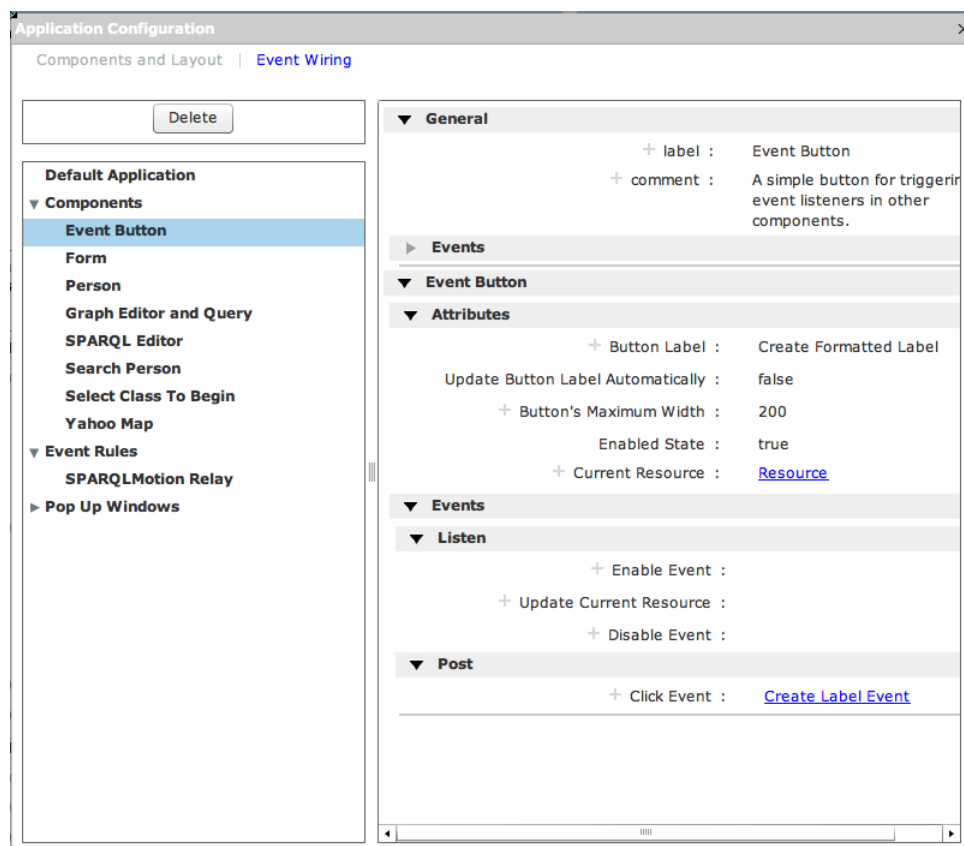
Given that these steps have been taken, a script can be invoked from TopBraid Ensemble by either a SPARQLMotion Relay attached to any event, or a component script invoked by selecting the component name in the cog icon in a component.

6.2 Using Ensemble Events to Invoke SPARQLMotion Scripts

A script can be associated with any Ensemble event by associating the script with a SPARQLMotion Relay. The relay will listen for an event, submit the SPARQLMotion script call to the server, and relay the event to another event. As an example, the following steps show how to modify the Default Application to invoke the script designed in the previous section when clicking on a button component.

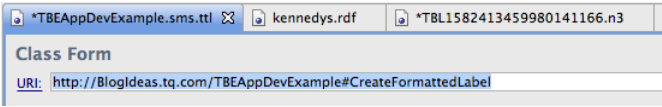
1. Open the TBL Personal Server console by starting TBC-ME and opening a browser with the URL `http://localhost:8083/tbl`
2. Choose a configurable Ensemble application, such as the Default Application, and open the kennedys data graph in TopBraid/kennedys (any model can be used, but using this readily available model will help facilitate the example).
3. Using the Add Components button at the top of the page (see the "+" icon), add the following components:
 - To create a button, choose Add Components > Event Button.
 - To create the relay that will call the script, choose Add Event Rules > SPARQLMotion Relay.
4. Edit the configuration (Show/Hide configuration or F2) and choose the newly created button in Components > Event Button. You can change the name of the button through the Button Label attribute. Open the Events and create a new event for the Post > Click Event. Use "Add new row" ('+' to the left of Click Event) and choose Create New... Type in any name any name you want. This will define a named event that other components can

listen to. In the example shown below, we have named the event "Create Label Event". Also set the Current Resource to a value, such as `rdfs:Resource`.



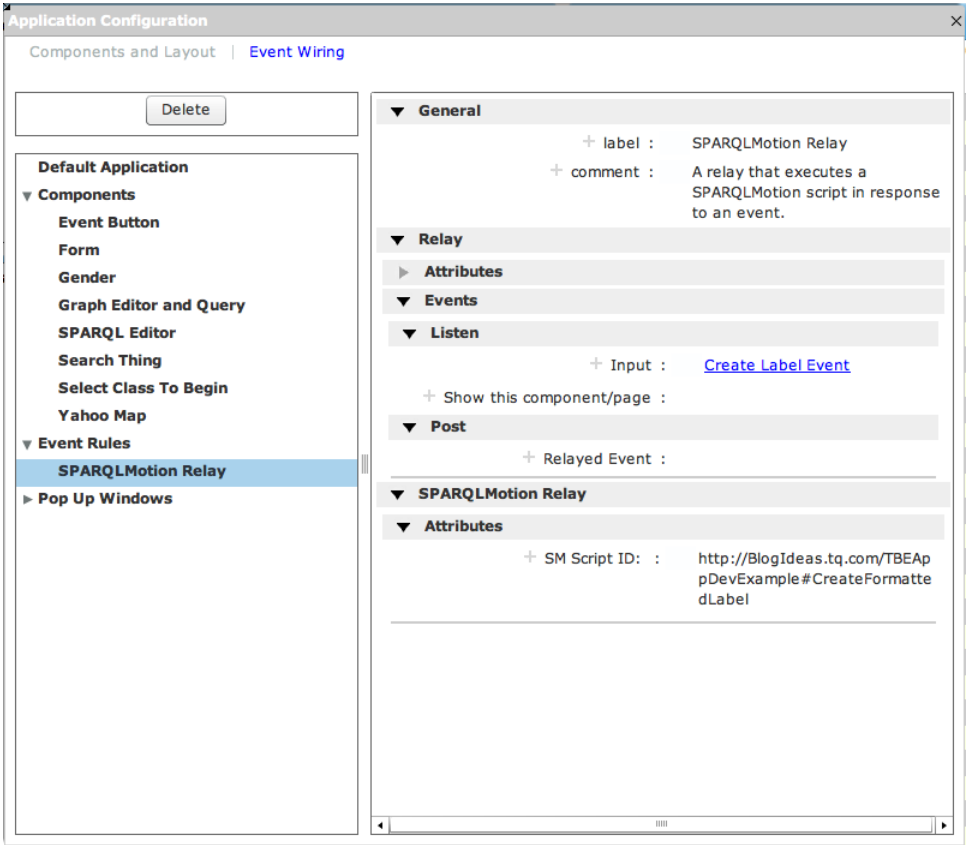
Configuring the Button component

5. Next, configure the SPARQLMotion Relay component. An event rule will listen for an event (Listen > Input), invoke a SPARQLMotion script on the server, and relay a different event (Post > Relayed Event). Do the following:
 - The relay component will appear under Event Rules in the configuration window. Use the same event name created in the previous step by choosing "Add existing entry" for the Listen > Input event, and type in the first few characters of your event name. Autocomplete will find your named event and you can save that. In this case, you don't need to relay the event, but the relay could post an event when finished.
 - Find the URI of the script created in the previous section by opening the script file in Composer and selecting the SPIN function class you created. You can copy either the Name or the URI of the function from the top text box of the Class Form in Composer. In this case, we copied the URI (see figure), and placed it in the SM Script ID attribute for the relay.



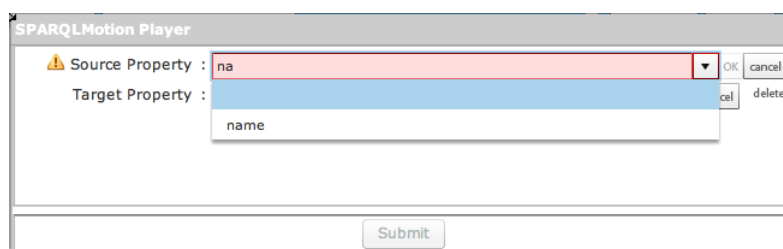
The SPIN function name

The SPARQLMotion relay will be configured to listen for the named event and execute the script registered in TopBraid Live. The following figure shows an example configured relay.



Configuring the SPARQLMotion Relay component

- 6. The button is now configured to call the SPARQLMotion script on the server. Following this example, click on the Person class of the classes view (since we are using the kennedys model) and note that the label column in the grid display is empty. Click on your button. A window will appear asking for input corresponding to the argument you created for the SPIN function and used in the SPARQLMotion query. Add the property name as shown in the figure (The property chosen by autocomplete is the label for kennedys:name, which hold strings of the format "firstname lastname".)




Running the example script

After clicking the Submit button, the SPIN function (which calls the SPARQLMotion script) will be invoked and the data applied to Ensemble session data. To save your data on the server, click the Save Data button in the top icon row of Ensemble. Of course, more sophisticated processing scripts can be created. As stated before, any SPARQLMotion script can be called in this manner for server-side processing.

All information displayed by Ensemble must be included in the `http://tb-session` graph. It is typical for SPARQLMotion scripts to include a `PerformUpdate` module with the query `"INSERT INTO <http://tb-session> { ?s ?p ?o } WHERE { ?s ?p ?o }"` to place triples processed by the script into Ensemble's session graph.

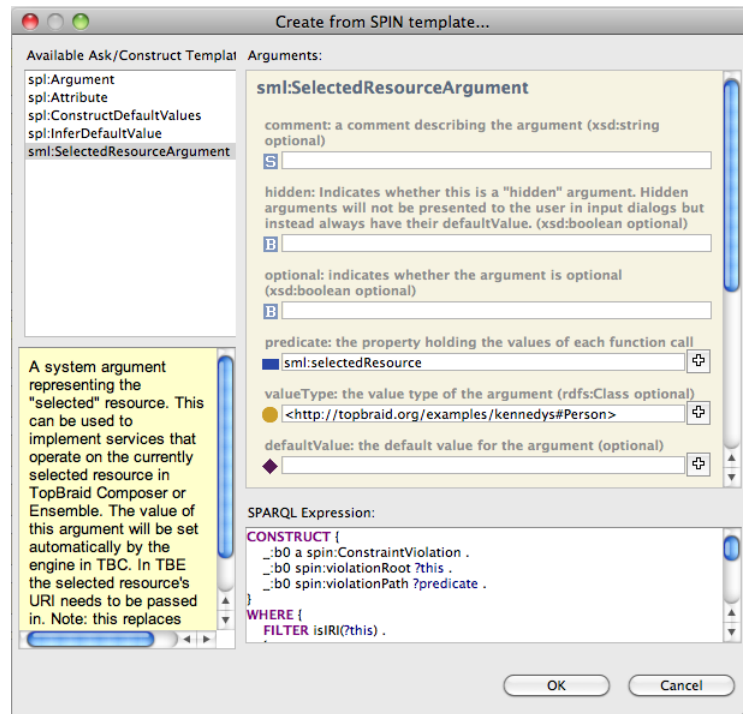
6.3 Invoking SPARQLMotion Scripts on Ensemble Components

SPARQLMotion scripts can also be associated with Ensemble components. All scripts registered with a type selected in a component are listed when a user clicks on the cog icon  in the top-left of the component. Scripts can also be defined for applicability to all applications and are selected from the cog icon in the top icon row of Ensemble. The list of applicable scripts is computed dynamically when the user selects the cog icon, making it possible to dynamically turn scripts on and off during application sessions.

Because component scripts are invoked on selected items in a component, a `SelectedResourceArgument` is required as an argument for the SPIN function that accesses the script. The `SelectedResourceArgument` passes the URI of the resource to the server for access by SPARQL queries. The `SelectedResourceArgument` can also be used in Ensemble events, per the previous section, to pass the selected resource as an argument to the script.

Using the example from the previous sections, we will modify the script to generate a formatted string for only the selected resource. The first steps are to modify the previous script in Composer.

1. Open the definition for the "CreateFormattedLabel" function from the previous section. Add a new argument by choosing "Create from SPIN template..." and choose the `sml:SelectedResourceArgument` template.
2. In this example, we want to restrict the script to run only when an instance of the Person class in the Kennedys model is chosen. The `valueType` property is used for this. Since the script model does not have the "kennedys" prefix defined, enter the full URI for Person in the kennedys model: `"<http://topbraid.org/examples/kennedys#Person>"`. The template should look like the following screen image.



Argument template for SelectedResourceArgument

3. This will add the argument to the script. Edit the script and connect the SelectedResource module to the script so that the variable ?selectedResource can be used.
4. Since we want to change only the selected resource, not all resources with the source property, we need to modify the PerformUpdate query to use ?selectedResource:

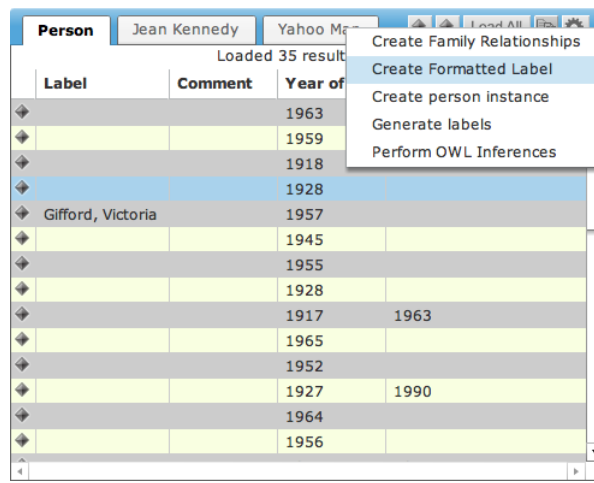
```
INSERT INTO <http://tb-session> {
    ?selectedResource ?targetProp ?newname .
}
WHERE {
    ?selectedResource ?sourceProp ?name .
    LET (?newname := smf:firstBound(smf:regex(?name,
        "([A-z]+) ([A-z]+) ([A-z]+)", "$2, $1 $3", ""),
        smf:regex(?name, "([A-z]+) ([A-z]+\.\.) ([A-z]+)", "$3, $1 $2", ""),
        smf:regex(?name, "([A-z]+) ([A-z]+)", "$2, $1", ""))) .
}
```

5. When running TBL Personal server, choose Scripts > Refresh/Display SPARQLMotion functions... to register changes to the script.

Open an Ensemble application with the kennedys model. You can use the application created in the previous section, but make sure you first clear sessions and close the kennedys.rdf file without saving (if using the TBL Personal Server) so we you again see the effects of creating a new label.

Choose a member of the "Person" class. In the Default Application, choose "Person" in the Tree component and instances will appear in the Grid component. If the Default Application has not been modified, you will see a list of instances of "Person" with blank labels. Choose the SPARQLMotion cog icon in the Grid component. Ensemble will communicate with the TBL server to get a list of all scripts applicable to members of kennedys:Person. The list in

the image below shows your "Create Formatted Label" (so named because this is the `rdfs:label` entry for the SPIN function) in the pull-down menu. The example shows that a few other scripts whose valueType is `kennedys:Person` have been defined for this TBL workspace..

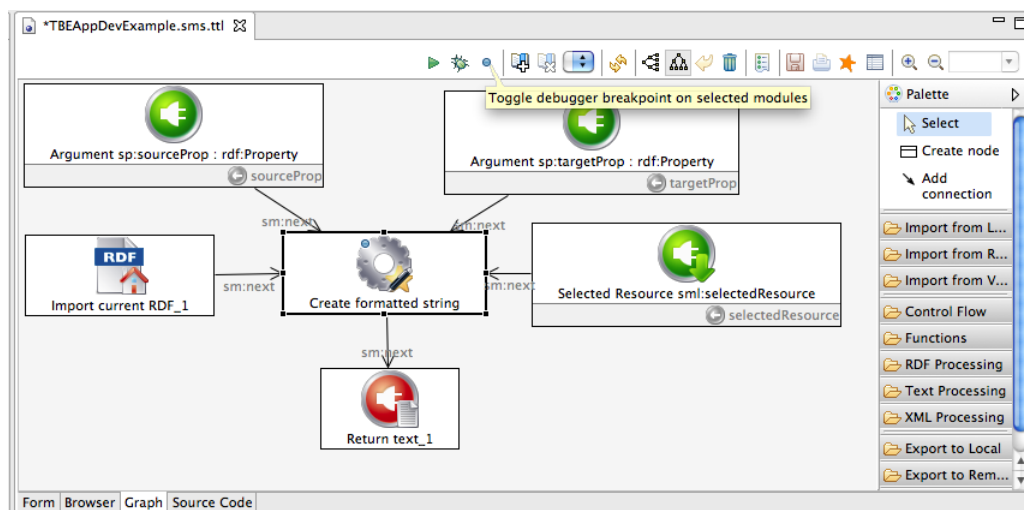


Displaying and Choosing scripts from an Ensemble component cog

6.4 Using the SPARQLMotion Debugger in TBL Personal Server

If you are using the Personal Server edition to TBL to debug your scripts, the SPARQLMotion debugger is a valuable tool. In the context of debugging scripts executed from Ensemble, breakpoints can be set in a script. When the breakpoint is encountered during server processing, the debugger will appear in TBC-ME.

Using the script from the previous section, the following image shows a breakpoint set at the "Create formatted string" module. This is accomplished by clicking on the module in the script then clicking on "Toggle debugger breakpoint..." (blue dot in the icon bar). This marks the module with a blue dot indicating that a breakpoint is set. To unset, select the module again and click the toggle icon.



Setting a script breakpoint

After setting the breakpoint, run the script in Ensemble. The script will stop at the breakpoint and a debug window will appear in Composer. As shown in the figure below the debug window will display information about arguments, variable bindings, allow single step through the script, etc. Queries can also be applied to the the data graph (i.e. <http://tb-session>) as it exists at the breakpoint. For more information, please see the SPARQLMotion tutorial.

SPARQLMotion debugger invoked from Ensemble script

Chapter 7

Building Custom Flex Components

While all TopBraid Live server side operations use the Java based Jena library, the Flex based TopBraid Ensemble clients employ a Flex library for basic graph operations (node management, triple matching) on an equivalent representation of the server stored graph. TopBraid Ensemble clients then defer to the server for more intensive graph operations such as SPARQL queries. Developing custom components for TBE does not require knowledge of Java or Jena APIs.

TopQuadrant offers a Software Development Kit (TBE SDK) for developing custom TBE components. When creating a custom component, TBE APIs must be used to enable component to operate within the TBE framework – to listen to and post events, to be customizable and to participate with the prepackaged components in the application assembly.

To create custom TBE components, go to the TBS Administration Console and select "Download SDK". You can then use the SDK within any Flex development environment such as Adobe® Flex® Builder.

When your new component is ready for use, go to the TBS Administration Console and select "Component Upload". When developing a component in ActionScript™ its manifest file is initially created in XML. When the component is to be used, this information is transformed into RDF using a transformer provided with TBE. This transformation is invoked by uploading a new or changed component using "Component Upload" function.

Manifest files for all built-in components are provided already transformed into RDF. They can be found in the following folder: `server.topbraidlive.org\dynamic\components`.

Chapter

8

Component Configuration Fields

Topics:

- [Common Events](#)
- [User Interface](#)
- [Pages and Popups](#)
- [Event Rules](#)
- [Listen Events Summary](#)
- [Post Events Summary](#)
- [Drag Events Summary](#)
- [Drop Events Summary](#)

8.1 Common Events

8.1.1 All Components

Listen events

Name	Description
Show this component/page	When this event is posted to, this component/page will become the currently viewed page. If the page or component is already displayed, it will become the active one; if it is hidden, it will be displayed first.

8.2 User Interface

8.2.1 Basket

Attributes

Name	Description	Default Value
Selected resource	This value is updated by selecting a resource in the basket by clicking on it. Setting the value of this attribute manually will highlight the resource (if the resource is present in the basket).	

Post events

Name	Description
Single-click Selection	This event is dispatched when the user single clicks a resource in the basket.
Double-click Selection	This event is dispatched when the user double clicks a resource in the basket.
Selection Change	This event is dispatched when the selection in the basket has changed. The resource which is now selected will be the resource which is posted.

Listen events

Name	Description
Add a Resource	This listen event will add the passed resource to the basket.

Buttons

Name	Description
Delete Button	Delete the resources selected in the basket.

Drag events

Name	Description
Dragging of a basket item	This event occurs when user releases a dragged node from the basket onto a valid drop target.(i.e. tree, visualization) Adding an event here will enable dragging.

Drop events

Name	Description
Add a resource via drag/drop	This event occurs when user releases a dragged node onto the basket. Adding an event here will enable this as a drop target for the specified event.

8.2.2 Event Button

Attributes

Name	Description	Default Value
Button Label	The label to be used on the actual button.	Button
Icon	The amount in pixels that this button can resize to.	
Icon property	The amount in pixels that this button can be resized to when the label changes.	
Tooltip	The amount in pixels that this button can resize to.	
Tooltip property	The amount in pixels that this button can resize to.	
URL	The url or website that will be launched in a new tab on click.	
URL property	The property to use to match against the selected resource to find the 'URL' value.	
Button's Maximum Width	If not using Static width, setting a Max Width on a button's which label changes is useful. The amount in pixels that this button can resize to.	200

Name	Description	Default Value
Button's Width	The width of the button in pixels.	
Icon Width	Set this to the desired width of the icon. please note that not setting this value when using an icon can result in an icon that is too large for the buttons max width.	
Icon Height	Set this to the desired height of the icon.	
Button's Label Placement	The location of the label on the button. This value is only relevant when an icon is specified. The valid options are 'left', 'right', 'top' and 'bottom'.	right
Update Button Label Automatically	Whether or not to update the button's label automatically with the 'Current Resource'.	false
Enabled State	When set to true, the button will be clickable. When set to false, the button will appear grayed out and unclickable.	true
Current Resource	This is the resource that will be passed in 'On Click' post event, and set by the 'Update Current Resource' listen event.	

Post events

Name	Description
Click Event	This event is dispatched when the user single clicks on a button which 'Enabled State' is true. If there is a 'Current Resource', then it will be posted with the event.

Listen events

Name	Description
Update Current Resource	When posted to, 'Current Resource' will be set to that of the resource in the event.
update Tooltip Property	When this listen event is posted to, this button's 'tooltip property' (see icon property) will be updated. Useful for dynamically changing the icon. (For many applications, using this effectively may require the use of a sparql relay to decide on the property).
update Icon Property	When this listen event is posted to, this button's 'icon property' (see icon property) will be updated. Useful for

Name	Description
	dynamically changing the icon. (For many applications, using this effectively may require the use of a sparql relay to decide on the property).
Enable Event	When this listen event is posted to, this button's 'Enabled State' will be set to true.
Disable Event	When this listen event is posted to, this button's 'Enabled State' will be set to false.

8.2.3 Search Form

Attributes

Name	Description	Default Value
Displayed Resource	The resource for which to display the input form for.	
Include Resource ID in SPARQL	Include the found resources (of type Search Type) as a variable in the generated results.	true
Update Label Automatically	If set to true, the tab's label will update to the displayed resource when the displayed resource changes. Otherwise you can define the component's label manually, and set this property to false.	false
Include Nested Forms	If set to true, the search form will show expandable and collapsible nested forms for more complex searches.	true
Include Checkboxes	If set to true, the search form will show checkboxes which allow the user to exclude or include search items.	false

Post events

Name	Description
Submit Search	This event will post a query event when a user clicks on the "Submit Search" button. Currently, the grid is best supported to listen to this event.

Listen events

Name	Description
Refresh Display	Updates display to show a search form which corresponds to the posted resource. Currently, the tree,

Name	Description
	grid, and clicking a resource in a form are best supported for posting to this event.

Buttons

Name	Description
Clear Fields Button	Clear all fields in the search form.

Drop events

Name	Description
Drop a in as input to add it as a value.	This event occurs when user releases a dragged node into an input widget on the form.

8.2.4 History Form**Post events**

Name	Description
Change Clicked	Triggered when the history view's "added" or "deleted" links are clicked.

Listen events

Name	Description
Show History	Display the version of the form showing the edit history.
Hide History	Hide the edit history, showing the regular version of the form.

Buttons

Name	Description
Toggle History Button	Show or hide history on the form.

8.2.5 Form**Attributes**

Name	Description	Default Value
Form Definition Class	If not specified, the form will use Form definition provided by TBC to the class which the instance belongs to. Apart from being able to specify search form, or edit form layouts, this property(when specified) allows a configurer to specify a different class's form definition (also defined by TBC) to use for this form.This is especially useful if a resource belongs to more than one class and you want to make sure that it's displayed in TBE with the form designed for one particular class.	

Name	Description	Default Value
Form Editing Mode	The editing mode of this form. Form modes: View, Edit and Review. View mode has no editing available to the user. Review Mode allows editing, while Edit mode has every widget in 'edit mode' to allow quick editing of all values.	http://server.topbraidlive.org/2008/ensemble#ReviewMode
Displayed Resource	The resource for which information is displayed in the form.	
Include constraint violations	If set to true, a warning will be shown for all values which violate constraints. Please refer to documentation for more about constraint violation. The Getting Started with SPIN tutorial describes how to define and test SPIN constraints.	false
Maximum numbers of values for a widget	The maximum values to display before showing a 'show all' link which will provide a triple match to a grid. See 'Show all subject post'	10
Update Label Automatically	Indicates whether to use the label of the currently displayed resource as the label on the component.	false

Post events

Name	Description
Began editing values	
Changing values complete	
Click Resource Link	Click Resource Link: this event is dispatched when a user clicks a hyperlink in the form. Currently, the components which best can accept a resource is the form or search form component.
Constraints are violated	The form is considered valid until a constraint is violated, at which point an event will be dispatched indicating so. This will only be relevant when 'Include constraint violations' is true.
Constraints are satisfied	The form is considered valid until a constraint is violated. Thus, after all violations have been removed, an event will be dispatched indicating that the form is valid. This will only be relevant when 'Include constraint violations' is true.

Name	Description
Show all property post	The event corresponding to the matching Predicate of a Triple Match(Subject, Predicate, Object) which the form uses for the "show all" feature when a widget has more values than the maximum value. A default event has been provided, but the configurer can specify any grid/popup/page to listen to this event.
Show all object post	The event corresponding to the matching Object of a Triple Match(Subject, Predicate, Object) which the form uses for the "show all" feature when a widget has more values than the maximum value. A default event has been provided, but the configurer can specify any grid/popup/page to listen to this event.
Show all subject post	The event corresponding to the matching Subject of a Triple Match(Subject, Predicate, Object) which the form uses for the "show all" feature when a widget has more values than the maximum value. A default event has been provided, but the configurer can specify any grid/popup/page to listen to this event.

Listen events

Name	Description
Update Displayed Resource	This event when posted to will update the 'displayed resource' property and consequently, the view of the form to the resource that was posted. Any event which posts a resource can be used to post to this listen event.
Clear Form	Clear the form of anything its displaying which will remove the 'displayed resource' value.
Update the Form Definition Class	An Event which will update the 'Form Definition Class' to the posted resource (should be a class). This functionality, to automatically update the properties which a form displays for a given instance, is in beta.

Buttons

Name	Description
Backward Button	Navigate backward to the previously displayed item
Forward Button	Navigate forward to the most recent item

Drop events

Name	Description
Drop a in as input to add it as a value.	This event occurs when user releases a dragged node into an input widget on the form.

8.2.6 Yahoo Map**Attributes**

Name	Description	Default Value
Yahoo Application ID	A developer yahoo map api key. Please get one at http://developer.yahoo.com/maps/simple/	Xg_nVozV34FE7rSC4h7v4.QKEpnCe7FA8
Mapped Class	Resources of this type are marked on the map.	
Mapped Resource	This resources is marked on the map.	
Display Info for Selected Marker	When a marker is clicked, display it's information on the map.	true

Post events

Name	Description
Marker Click	This event is dispatched when the user clicks a marker on the map. The resource corresponding to the clicked marker will be the object which is posted.
Marker Double Click	This event is dispatched when the user double-clicks a marker on the map. The resource corresponding to the double-clicked marker will be the object which is posted.

Listen events

Name	Description
Refresh Display (individual)	Display markers on the map for a given resource if it has geographical information. Currently, the tree, the grid and resources in the form can post this event.
Refresh Display (class members)	Display markers on the map for all instances of a given class with geographical information. Currently, it is best to use the tree to post to this event since it reliably displays classes.

8.2.7 Results Grid

Attributes

Name	Description	Default Value	Dependency
New resource filter	When adding a resource in the grid using the 'Add' icon, the options to add will be filtered by this value.		
Show number of results loaded	To show or not show the "loaded:x" status box.Setting this to true will display the number of results loaded so far; setting it to false will not show that number.	true	
Grid Object	The 'Grid Object' will be used as the object of a triple match in which the 'Property' is the predicate. The grid will then be populated with the subjects of the matching triples. Setting this will nullify the 'Grid Subject'. Note: posting to this event is equivalent to deprecated listen event 'update resource' when the deprecated value 'resource as object of triple match' was true.		'Grid Property' must have some value. To dynamically reset 'Grid Object' and refresh results, use 'Refresh Grid Object' event.
Grid Subject	The 'Grid Subject' will be used as the subject of a triple match in which the 'Property' is the predicate. The grid will then be populated with the objects of the matching triples. Setting this will nullify the 'Grid Object'.Note: posting to this event is equivalent to deprecated listen event 'update resource' when the deprecated value 'resource as object of triple match' was false.		'Grid Property' must have some value. To dynamically reset 'Grid Object' and refresh results, use 'Refresh Grid Object' event.
Column resource	<p>This attribute allows you to override the default column layout of a grid populated using either 'Grid Object' and 'Property' attributes (and corresponding events) or 'Grid Subject' and 'Property' attributes (and corresponding events).</p> <p>When a grid's 'Property' attribute = "rdf:type" and the grid is populated using a 'Refresh Grid Object' event, by default the columns of a grid are determined by the column layout for the 'Grid Object' resource as defined in the TBC column layout, which is stored in the .tbc metadata file for a given data file. When the property is different from "rdf:type" and a grid is populated using any of the non-query events ('Refresh Grid Object', 'Refresh Grid Subject', 'Refresh property'), by default the grid will present a single column.</p>		'Grid Property' and either 'Grid Object' or 'Grid Subject' attributes must have some value set either directly or through the corresponding events.

Name	Description	Default Value	Dependency
	The 'Column Resource' feature allows you to override the default column layout with the (specified in corresponding .tbc) column layout of ANY class. Simply set this value to a resource which has the desired column layout.		
Property	The grid will be populated with all resources that are either the subject or object of a triple match where 'Property' is the predicate in the triple match.		Either 'Grid Object' or 'Grid Subject' attribute (but not both) must have some value. To dynamically reset Property and refresh results, use the 'Refresh property' event.
Update Label Automatically	If set to true, the tab's label will update to the label of the displayed resource when the displayed resource changes. Otherwise you can define the component's label manually, and set this property to false.	false	Should only be used when Property = "rdf:type" and the grid values are populated based on the 'Grid Object'.
Selected resource	This value is updated by selecting a resource in the grid by clicking on it. Setting the value of this attribute manually will highlight the resource (if the resource is present in the grid).		
Allow to rearrange columns	Allow the user to drag the columns in order to rearrange them. This is in beta because column layout for tbc is in development. (rearranged columns do not persist)	false	
SPARQL Query	This value represents the current SPARQL query used to populate the grid. You can enter any SPARQL query in this field. This attribute can also be set via the 'Display SPARQL Query' event.		The 'Display SPARQL Query' event will update this attribute and trigger a rerun of the query. To rerun the query when the attribute is populated using a different approach or when the data has changed while

Name	Description	Default Value	Dependency
			the query did not change, use the 'Refresh search results' event.
Search Criteria	This value represents the SPIN query used to populate the grid. This attribute is set via the 'Execute Query' event. The difference between a SPIN and a SPARQL query is that a SPARQL query is a text string while a SPIN query uses an RDF serialization of SPARQL. It is represented as a node of rdf:type sp:Select, or a SPIN node.		The 'Execute Query' event will populate the attribute and run the query. To rerun the query when the attribute is populated using a different approach or when the data has changed while the query did not change, use the 'Refresh search results' event.
Query Template Variable	This value represents the node that is to be bound to the ?this variable when the 'Query Template' functionality is executed. This variable is set to the passed resource via the 'Bind and Execute Query Template' event.		Requires some value in the 'Query Template' attribute.
Query Template	The SPARQL query to be used for the Query Template Functionality. The query's WHERE clause must contain a variable ?this which will be bound to the node specified in 'Query Template Variable Binding' attribute.		Requires some value in the 'Query Template Variable Binding' attribute. Works with the 'Bind and Execute Query Template' event.

Post events

Name	Description
Single-click Selection	This event is dispatched when the user single clicks on the row in the grid. The resource which is clicked will be the object which is posted.
Double-click Selection	This event is dispatched when the user double clicks on the row in the grid. The resource which is clicked will be the object which is posted.

Name	Description
Selection Change	This event is dispatched when the selection in the grid has changed. The resource which is now selected will be the object which is posted.

Listen events

Name	Description
Refresh Grid Object	This listen event will update the 'Grid Object' value which is used to populate the grid with a triple match.
Refresh Grid Subject	This listen event will update the 'Grid Subject' value which is used to populate the grid with a triple match. Note: used to be 'update resource'.
Refresh property	When posted to, this listen event will update the 'Property' attribute and repopulate the grid with triples corresponding to the newly passed property.
Refresh search results	This listen event will refresh the grid if it has been populated via a SPARQL/SPIN query. If the grid is populated with the 'Grid subject' or 'Grid object' attributes, triggering this event will have no effect.
Execute Query	When posted to, this listen event will populate the grid with the results of a SPIN query. (Search Form and Graph Visualization post these queries).
Display SPARQL Query	When posted to, this listen event will populate the grid with the results of a SPARQL query. (SPARQL editor post these queries).
Bind and Execute Query Template	This listen event, when posted to will trigger the query template functionality by setting the 'query template' variable to the passed resource, and executing the query specified in the 'Query Template'.
Update the Column Resource	Change the column resource to that given in the event. Note: when specified, the column resource will always be used to determine which columns show in the grid.
Clear selection(s)	Deselect any selected items in the grid.

Buttons

Name	Description
Create New Button	Add new resource.
Delete Button	Remove the resources selected in the grid.
Load All Button	Load all matches into the grid.
Copy Button	Copy the selected entries to the clipboard for pasting into another application (e.g. to a spreadsheet).

Drag events

Name	Description
Dragging of a grid entry	This event occurs when user releases a dragged node from the grid onto a valid drop target.(i.e. tree, visualization) Adding an event here will enable dragging.

Drop events

Name	Description
Refresh display via drag/drop	This event occurs when user releases a dragged node onto the grid. The grid will show subjects related to the dragged resource. Adding an event here will enable this as a drop target for the specified event.

8.2.8 Info Box**Attributes**

Name	Description	Default Value
Padding Left	The padding on the left-hand side of the component.	5
Padding Right	The padding on the right-hand side of the component.	5
Padding Top	The padding on the top side of the component.	5
Padding Bottom	The padding on the bottom side of the component.	5
Static Text	Alternatively from displaying the text of a property of an instance or class(current resource), the infobox can display any text specified here. Please note that the infobox displays html text.	

Name	Description	Default Value
Property	The property which will be used to get the displayed text from the current resource. i.e. if the property is <code>rdfs:comment</code> , the infobox will display the <code>rdfs:comment</code> for the current resource. note: the text found will override the 'static text'.	http://www.w3.org/2000/01/rdf-schema#comment
Font Size	A number representing the size of the font to use.	13
Current Resource	The resource the infobox currently is holding. The resource is used via the 'Property' field to display found text. Use the 'Update Current Resource' listen event or set this value manually.	

Listen events

Name	Description
Update Current Resource	When posted to, this event will update the "Current Resource" value to the passed resource.
Update Property	When posted to, this event will update the "property" value to that of the passed resource.

8.2.9 Search Field**Attributes**

Name	Description	Default Value
Lookup Filter	The lookup will use this attribute (should be a specified <code>rdfs:Class</code>) to determine which values can be selected in the lookup. The default value is <code>rdfs:Resource</code> .	http://www.w3.org/2000/01/rdf-schema#Resource

Post events

Name	Description
Submit Search	Post a resource via clicking on the search button.
Select from Drop-down	Post a resource via selecting it in the dropdown.

Listen events

Name	Description
Change the filter	When posted to, this listen event will update the 'Lookup Filter'. The default value is <code>rdfs:Resource</code> .

8.2.10 SPARQL Editor

Post events

Name	Description
Query Results Requested	This event will be posted when the user clicks the 'Submit Query' button and will post the available text as a SPARQL query. (Grid can listen to this)

Listen events

Name	Description
Edit Query	When posted to, this listen event will populate the component with the posted SPARQL in text form.

8.2.11 TopBraid Ensemble

Attributes

Name	Description	Default Value
Multi-paged Application	Whether or not this application uses paging infrastructure. Please save your work by 'saving as' before changing this attribute because switching between Multi-Page and non-Multi-Page will not restore your original app design.	false
Lookup Filter	The lookup will use this attribute (should be a specified rdfs:Class) to determine which values can be selected in the lookup. The default value is rdfs:Resource.	http://www.w3.org/2000/01/rdf-schema#Resource
Display the Top Search Field		true
Show TopQuadrant Link	Turn on Top Quadrant Branding	false
Use Embedded Font	Turn off embedded fonts to use device fonts for better unicode support. *requires browser refresh to take effect	false
Draggable-Tabs	Whether tabs can be rearranged on the screen. This is a feature that would be common to turn false when finished configuring an application.	true
Component Background Color	Color choice for the background of all components.	0xFFFFFFFF

Name	Description	Default Value
Font Color - Primary	Color choice for all text in the application unless otherwise specified.	0x333333
Minimum Font Size	Font size to be used throughout the application.	11
Form Section Header Color	Color of the section header which appears in the form and search form.	0xeeeeee
Font Color - Literals	Font color used for literals in the form.	0x333333
Font Color - Resources	Font color used for resources (appears as a clickable hyperlink) in the form.	0x0000ff
Tab Roundness	Corner Radius of Tabs and Component where zero would be no roundness.	3
Selected Tab Font Color	Color used for all selected tab's label.	0x000000
Unselected Tab Font Color	Color used for all unselected tab's label.	0x666666
Selected Tab Color	Background color used for all selected tabs. It is recommended to make this color the same as the component background color.	0xffffffff
Unselected Tab Color	Background color used for all unselected tabs.	0xffffffff
Header Background Color	Background color used for the header of the application.	0xffffffff
Component Border Color	Border color used for all components.	0x666666
Font Color - Title	Color to be used for the title of the application.	0x666666

Name	Description	Default Value
Font Color - Popup Title	Color to be used for the title of every popup.	0xFFFFFFFF
Component Header Color	Color to be used for the Header of every component in the application.	0x53A6DA
Popup Header Color	Color to be used for the Header of every Popup in the application.	0xcccccc
Background Color	Color to be used for the background of the entire application.	0xFFFFFFFF
Focus Color	Color to be used as a highlight around an input such as a button, text field, or row in a grid to show that it has focus.	0x53A6DA
Header Strip Color	Color to be used for the strip separating the header from the rest of the application.	0xFFFFFFFF
Current Logo Selection	Http url of an image file to be used for a logo, i.e. http://www.topquadrant.com/logo.jpg . note: use 'empty' to use no logo	empty

Post events

Name	Description
Submit Search	Post a resource via using the search button.
Select from Drop-down	Post a resource via selecting it in the dropdown

Listen events

Name	Description
Change the Navigation Filter Type	Change the navigation filter type to the resource given by a posted event.

Buttons

Name	Description
Save App Button	Save application or Save this application as a new application on the server.
Save Data Button	Save current changes to data on the server

Name	Description
Launch Configuration Button	Show/Hide Configuration(F2)
Add Menu	Add components, pages, event rules and popups.

8.2.12 Tree

Attributes

Name	Description	Default Value
Treat property objects as parents?	Setting this attribute to true will display a tree in which each parent-child relationship represents a triple in which the parent is the object and the child is the subject; setting this attribute to false displays a tree in which each parent-child relationship represents a triple in which the parent is the subject and the child is the object.	true
Treat nodes as object of count statements?	Count queries (which are just triple matches) use the counts property and the node as the subject of the triple match. When using 'show tree in reverse', you may need to toggle this value to use the node as the object instead.	true
Default Expanded	Change this attribute so that each time the tree is populated, all nodes will be expanded(to a depth level of 4).	false
Root Node	The root node which the tree is currently displaying. All nodes in the tree are populated by recursively executing a triple match over found nodes beginning with the root node.	http://www.w3.org/2002/07/owl#Thing
Transitive Property	an rdfs Property (e.g. rdfs:subClassOf) by which to use for the triple match that relates child-parent nodes in the tree in the tree.	http://www.w3.org/2000/01/rdf-schema#subClassOf
Counts Property	an rdfs Property (e.g. rdfs:type) by which to find instances of and use to provide count information(e.g. Thing(45)).	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
Update label Automatically with root	Setting this to true will automatically include the 'root node' in the label. Not mutually exclusive to 'Update label Automatically with property'	false
Update label Automatically with transitive property	Setting this to true will automatically include the 'transitive property' in the label. Not mutually exclusive to 'Update label Automatically with root'	false

Name	Description	Default Value
Display the search field	Include a convenience search field to quickly find and highlight items in the tree (helpful for large class hierarchies).	false
Show the root of the tree in the display	Toggle this feature to true in order to make the root node visible/hidden in the tree.	false
Selected resource	The currently selected item in the tree.	
Sorting property for nodes in the tree	A rdfs Property (i.e. rdfs:Label) for the tree to use when sorting items. If no property is specified, then the displayed label will be used to sort the nodes.	

Post events

Name	Description
Single-click Selection	This event is dispatched when the user single clicks on an item in the tree. The resource which is clicked will be the object which is posted.
Double-click Selection	This event is dispatched when the user single clicks on the row in the grid. The resource which is clicked will be the object which is posted.
Selection change	This event is dispatched when the 'Selected Item' in the tree is changed. The resource which is clicked will be the object which is posted.

Listen events

Name	Description
Switch Root	Post to this listen event to set the 'Root Node' in the tree to the passed resource.
Highlight a node	Post to this listen event to set the 'Selected Item' in the tree to the passed resource (if that resource exists in the tree).
Change the transitive property	Post to this listen event to dynamically change the 'Transitive Property' to the passed resource.

Buttons

Name	Description
Add Child Button	Add a child node to the selected node.
Add Sibling Button	Add a sibling node to the selected node.
Delete Button	Delete the selected node.

Drag events

Name	Description
Drag a tree resource	If you use this event, it will enable dragging of a tree resource. The event will be dispatched when the user begins dragging the dragged node.(the tree, grid, or visualization, if it has a drop event configured to listen to the drag event). The resource posted will be the dragged item.

Drop events

Name	Description
Drop resource into tree	This event is fired when a dragged node is dropped onto a valid drop target. To make a drop target a valid drop target for a given drag, simply configure the 'drop event' to be the same event as the 'drag event'. Currently, only the tree and grid can post to this event.

8.2.13 Graph Editor and Query

Attributes

Name	Description	Default Value
Settings	This attribute is for internal use only is under development.	
Image Property	This attribute accepts rdfs:Resource as a property to be used for any given nodes thumbnail image.	

Post events

Name	Description
Node Selected	This event is dispatched when the user selects a node in the graph viz. The resource of the node will be passed. (form, button, relay can listen to this event)

Name	Description
Query Results Requested	This event is dispatched when the user clicks the 'run query' button. Currently, the chart and grid can listen to this event.

Drop events

Name	Description
Drop resource into visualization	This is a listen event which will add the dragged item to the graph visualization as a new node. Currently, the grid and tree can post these drop events.

8.2.14 White Space

8.3 Pages and Popups

8.3.1 TopBraid Page

Attributes

Name	Description	Default Value
Page Index	Set this value to reorder the buttons in the Navigation Bar. Do not delete, but set the index to a negative number to not have this page included in the Page Navigation Bar.	0
Page Height	Set this value to indicate the height of the page in pixels. If the value is ever below the height in pixels of the browser window, this value will have no effect.	0

Post events

Name	Description
Page is visible	This event will be dispatched when the page becomes visible.

8.3.2 Popup Window

Attributes

Name	Description	Default Value
Draggable-Tabs	Whether tabs can be rearranged on the screen. It is best to set this to true, if divider's dragging is also set to true.	true

Name	Description	Default Value
Can add Component/ Pages	Whether or not more components or pages can be added to the popup	true
Show Navigation Bar	Whether the navigator bar should be visible in the pop up.	false
Default Height	The current height of the popup window. Upon creating a popup, its height will be set to this property's default value (600 pixels). To determine what your user's will initially see as the popup's height, resize the popup to the desired value before saving the application. Because All popup's are resizable, this value updates in the user-session as they resize.	600
Popup Width	The current width of the popup window. Upon creating a popup, its width will be set to this property's default value (600 pixels). To determine what your user's will initially see as the popup's width, resize the popup to the desired value before saving the application. Because All popup's are resizable, this value updates in the user-session as they resize.	600

Post events

Name	Description
Pop up opened	This event will be dispatched when the popup is shown. The popup is shown via the 'Open the popup' listen event.

Listen events

Name	Description
Open the popup	When this listen event is posted to, this popup will show up on the screen. (note: please do not use the abstract event 'show/open' this component as it will have no effect on the popup.)
Reposition the popup	Reposition the popup to it's original location on the screen.
Close the popup	When this listen event is posted to, this popup will be hidden if it is currently being shown.

8.4 Event Rules

8.4.1 Relay

Attributes

Name	Description	Default Value
Relay Input	Setting this value will trigger the relay action using the given value. This is intended to be used to invoke the relay action at application start time in order to execute any initialization processes that may be required for an application.	

Post events

Name	Description
Relayed Event	This post event is posted when the 'Input' event is posted to and passes the 'Relay Input'.

Listen events

Name	Description
Input	When this event is posted to, it will trigger the relay action, which in turn will post the 'Relayed Event'.

8.4.2 Deep Linking Component

Attributes

Name	Description	Default Value
Relevant Event	Those events for which a corresponding key/value pair should be searched for in the URL fragment at startup.	

8.4.3 SPARQL Rule

Attributes

Name	Description	Default Value
Variable name	The variable name(a string) to be used in the SPARQL query specified by the 'Sparql:' string.	this
SPARQL:	The SPARQL to be executed on the passed resource.It's best to specify any prefixes that will be used with a PREFIX keyword at the beginning of the query.	SELECT ?out WHERE { ?this rdf:type ?out }

8.4.4 SPARQLMotion Relay

Attributes

Name	Description	Default Value
SM Script ID:	The ID of the SPIN function Web Service to be used in the relay. Learn more at http://www.topquadrant.com/products/SPARQLMotion.html	

8.5 Listen Events Summary

Component	Name	Description
Basket	Add a Resource	This listen event will add the passed resource to the basket.
All Components	Show this component/page	When this event is posted to, this component/page will become the currently viewed page.
Relay	Input	When this event is posted to, it will trigger the relay action, which in turn will post the 'Relayed Event'.
Config Event Delete Button	Update Current Resource	
Config Event Delete Button	Enable Event	When this listen event is posted to, this button's 'Enabled State' will be set to true.
Config Event Delete Button	Disable Event	When this listen event is posted to, this button's 'Enabled State' will be set to false.
Config Delete Button	Update Current Resource	
Event Button	Update Current Resource	When posted to, 'Current Resource' will be set to that of the resource in the event.
Event Button	update Tooltip Property	When this listen event is posted to, this button's 'tooltip property' (see icon property) will be updated. Useful for dynamically changing the icon. (For many applications, using this effectively may require the use of a sparql relay to decide on the property).
Event Button	update Icon Property	When this listen event is posted to, this button's 'icon property' (see icon property) will be updated. Useful for dynamically changing the

Component	Name	Description
		icon. (For many applications, using this effectively may require the use of a sparql relay to decide on the property).
Event Button	Enable Event	When this listen event is posted to, this button's 'Enabled State' will be set to true.
Event Button	Disable Event	When this listen event is posted to, this button's 'Enabled State' will be set to false.
Search Form	Refresh Display	Updates display to show a search form which corresponds to the posted resource. Currently, the tree, grid, and clicking a resource in a form are best supported for posting to this event.
History Form	Show History	Display the version of the form showing the edit history.
History Form	Hide History	Hide the edit history, showing the regular version of the form.
Form	Update Displayed Resource	This event when posted to will update the 'displayed resource' property and consequently, the view of the form to the resource that was posted. Any event which posts a resource can be used to post to this listen event.
Form	Clear Form	Clear the form of anything its displaying which will remove the 'displayed resource' value.
Form	Update the Form Definition Class	An Event which will update the 'Form Definition Class' to the posted resource (should be a class). This functionality, to automatically update the properties which a form displays for a given instance, is in beta.
Yahoo Map	Refresh Display (individual)	Display markers on the map for a given resource if it has geographical information. Currently, the tree, the grid and resources in the form can post this event.
Yahoo Map	Refresh Display (class members)	Display markers on the map for all instances of a given class with geographical information. Currently, it is best to use the tree to post to this event since it reliably displays classes.
Results Grid	Refresh Grid Object	This listen event will update the 'Grid Object' value which is used to populate the grid with a triple match.

Component	Name	Description
Results Grid	Refresh Grid Subject	This listen event will update the 'Grid Subject' value which is used to populate the grid with a triple match. Note: used to be 'update resource'.
Results Grid	Refresh property	When posted to, this listen event will update the 'Property' attribute and repopulate the grid with triples corresponding to the newly passed property.
Results Grid	Refresh search results	This listen event will refresh the grid if it has been populated via a SPARQL/SPIN query. If the grid is populated with the 'Grid subject' or 'Grid object' attributes, triggering this event will have no effect.
Results Grid	Execute Query	When posted to, this listen event will populate the grid with the results of a SPIN query. (Search Form and Graph Visualization post these queries).
Results Grid	Display SPARQL Query	When posted to, this listen event will populate the grid with the results of a SPARQL query. (SPARQL editor post these queries).
Results Grid	Bind and Execute Query Template	This listen event, when posted to will trigger the query template functionality by setting the 'query template' variable to the passed resource, and executing the query specified in the 'Query Template'.
Results Grid	Update the Column Resource	Change the column resource to that given in the event. Note: when specified, the column resource will always be used to determine which columns show in the grid.
Results Grid	Clear selection(s)	Deselect any selected items in the grid.
Info Box	Update Current Resource	When posted to, this event will update the "Current Resource" value to the passed resource.
Info Box	Update Property	When posted to, this event will update the "property" value to that of the passed resource.
Search Field	Change the filter	When posted to, this listen event will update the 'Lookup Filter'. The default value is <code>rdfs:Resource</code> .
Popup Window	Open the popup	When this listen event is posted to, this popup will show up on the screen. (note: please do not use the abstract event 'show/open' this component as it will have no effect on the popup.)

Component	Name	Description
Popup Window	Reposition the popup	Reposition the popup to it's original location on the screen.
Popup Window	Close the popup	When this listen event is posted to, this popup will be hidden if it is currently being shown.
SPARQL Editor	Edit Query	When posted to, this listen event will populate the component with the posted SPARQL in text form.
SPARQLMotion Popup Window	Handle Input	Execute the SM script again.
SPARQLMotion Popup Window	Constraints Satisfied	
SPARQLMotion Popup Window	Constraints Violated	
SPARQLMotion Popup Window	All Values Committed	
SPARQLMotion Popup Window	Values Being Edited	
TopBraid Ensemble	Change the Navigation Filter Type	Change the navigation filter type to the resource given by a posted event.
Tree	Switch Root	Post to this listen event to set the 'Root Node' in the tree to the passed resource.
Tree	Highlight a node	Post to this listen event to set the 'Selected Item' in the tree to the passed resource (if that resource exists in the tree).
Tree	Change the transitive property	Post to this listen event to dynamically change the 'Transitive Property' to the passed resource.

8.6 Post Events Summary

Component	Name	Description
Basket	Single-click Selection	This event is dispatched when the user single clicks a resource in the basket.
Basket	Double-click Selection	This event is dispatched when the user double clicks a resource in the basket.
Basket	Selection Change	This event is dispatched when the selection in the basket has changed. The resource which is now selected will be the resource which is posted.
Relay	Relayed Event	This post event is posted when the 'Input' event is posted to and passes the 'Relay Input'.
Config Event Delete Button	Delete Event	
Config Delete Button	Delete Component	
Event Button	Click Event	This event is dispatched when the user single clicks on a button which 'Enabled State' is true. If there is a 'Current Resource', then it will be posted with the event.
Search Form	Submit Search	This event will post a query event when a user clicks on the "Submit Search" button. Currently, the grid is best supported to listen to this event.
History Form	Change Clicked	Triggered when the history view's "added" or "deleted" links are clicked.
Form	Began editing values	
Form	Changing values complete	
Form	Click Resource Link	Click Resource Link: this event is dispatched when a user clicks a hyperlink in the form. Currently, the components which best can accept a resource is the form or search form component.
Form	Constraints are violated	The form is considered valid until a constraint is violated, at which point an event will be dispatched indicating so. This will only be relevant when 'Include constraint violations' is true.

Component	Name	Description
Form	Constraints are satisfied	The form is considered valid until a constraint is violated. Thus, after all violations have been removed, an event will be dispatched indicating that the form is valid. This will only be relevant when 'Include constraint violations' is true.
Form	Show all property post	The event corresponding to the matching Predicate of a Triple Match(Subject, Predicate, Object) which the form uses for the "show all" feature when a widget has more values than the maximum value. A default event has been provided, but the configurer can specify any grid/popup/page to listen to this event.
Form	Show all object post	The event corresponding to the matching Object of a Triple Match(Subject, Predicate, Object) which the form uses for the "show all" feature when a widget has more values than the maximum value. A default event has been provided, but the configurer can specify any grid/popup/page to listen to this event.
Form	Show all subject post	The event corresponding to the matching Subject of a Triple Match(Subject, Predicate, Object) which the form uses for the "show all" feature when a widget has more values than the maximum value. A default event has been provided, but the configurer can specify any grid/popup/page to listen to this event.
Yahoo Map	Marker Click	This event is dispatched when the user clicks a marker on the map. The resource corresponding to the clicked marker will be the object which is posted.
Yahoo Map	Marker Double Click	This event is dispatched when the user double-clicks a marker on the map. The resource corresponding to the double-clicked marker will be the object which is posted.
Results Grid	Single-click Selection	This event is dispatched when the user single clicks on the row in the grid. The resource which is clicked will be the object which is posted.
Results Grid	Double-click Selection	This event is dispatched when the user double clicks on the row in the grid. The resource which is clicked will be the object which is posted.
Results Grid	Selection Change	This event is dispatched when the selection in the grid has changed. The resource which is now selected will be the object which is posted.
Search Field	Submit Search	Post a resource via clicking on the search button.

Component	Name	Description
Search Field	Select from Drop-down	Post a resource via selecting it in the dropdown.
TopBraid Page	Page is visible	This event will be dispatched when the page becomes visible.
Popup Window	Pop up opened	This event will be dispatched when the popup is shown. The popup is shown via the 'Open the popup' listen event.
SPARQL Editor	Query Results Requested	This event will be posted when the user clicks the 'Submit Query' button and will post the available text as a SPARQL query. (Grid can listen to this)
SPARQLMotion Popup Window	Form Complete	
SPARQLMotion Popup Window	Form Incomplete	
TopBraid Ensemble	Submit Search	Post a resource via using the search button.
TopBraid Ensemble	Select from Drop-down	Post a resource via selecting it in the dropdown
Tree	Single-click Selection	This event is dispatched when the user single clicks on an item in the tree. The resource which is clicked will be the object which is posted.
Tree	Double-click Selection	This event is dispatched when the user single clicks on the row in the grid. The resource which is clicked will be the object which is posted.
Tree	Selection change	This event is dispatched when the 'Selected Item' in the tree is changed. The resource which is clicked will be the object which is posted.
Graph Editor and Query	Node Selected	This event is dispatched when the user selects a node in the graph viz. The resource of the node will be passed. (form, button, relay can listen to this event)
Graph Editor and Query	Query Results Requested	This event is dispatched when the user clicks the 'run query' button. Currently, the chart and grid can listen to this event.

8.7 Drag Events Summary

Component	Name	Description
Basket	Dragging of a basket item	This event occurs when user releases a dragged node from the basket onto a valid drop target.(i.e. tree, visualization) Adding an event here will enable dragging.
Results Grid	Dragging of a grid entry	This event occurs when user releases a dragged node from the grid onto a valid drop target.(i.e. tree, visualization) Adding an event here will enable dragging.
Tree	Drag a tree resource	If you use this event, it will enable dragging of a tree resource. The event will be dispatched when the user begins dragging the dragged node.(the tree, grid, or visualization, if it has a drop event configured to listen to the drag event). The resource posted will be the dragged item.

8.8 Drop Events Summary

Component	Name	Description
Basket	Add a resource via drag/drop	This event occurs when user releases a dragged node onto the basket. Adding an event here will enable this as a drop target for the specified event.
Search Form	Drop a in as input to add it as a value.	This event occurs when user releases a dragged node into an input widget on the form.
Form	Drop a in as input to add it as a value.	This event occurs when user releases a dragged node into an input widget on the form.
Results Grid	Refresh display via drag/drop	This event occurs when user releases a dragged node onto the grid. The grid will shows subjects related to the dragged resource. Adding an event here will enable this as a drop target for the specified event.
Tree	Drop resource into tree	This event is fired when a dragged node is dropped onto a valid drop target. To make a drop target a valid drop target for a given drag, simply configure the 'drop event' to be the same event as the 'drag event'. Currently, only the tree and grid can post to this event.
Graph Editor and Query	Drop resource into visualization	This is a listen event which will add the dragged item to the graph visualization as a new node. Currently, the grid and tree can post these drop events.