Red Hat JBoss BRMS 6.1

User Guide

The User Guide for Red Hat JBoss BRMS

Last Updated: 2017-11-13

Red Hat JBoss BRMS 6.1 User Guide

The User Guide for Red Hat JBoss BRMS

Doug Hoffman

Eva Kopalova

B Long
Red Hat Engineering Content Services
belong@redhat.com

Gemma Sheldon
Red Hat Engineering Content Services
gsheldon@redhat.com

Joshua Wulf
jwulf@redhat.com

Legal Notice

Abstract

A guide to defining and managing business processes withRed Hat JBoss BRMS.

Table of Contents

CHAPTER 1. INTRODUCTION

1.1. ABOUT RED HAT JBOSS BRMS

Red Hat JBoss BRMS is an open source decision management platform that combines Business Rules Management and Complex Event Processing. It automates business decisions and makes that logic available to the entire business.

Red Hat JBoss BRMS uses a centralized repository where all resources are stored. This ensures consistency, transparency, and the ability to audit across the business. Business users can modify business logic without requiring assistance from IT personnel.

Business Resource Planner is included with this release.

Red Hat JBoss BRMS is supported for use with Red Hat Enterprise Linux 7 (RHEL7).

1.2. USE CASE: BUSINESS DECISION MANAGEMENT IN THE INSURANCE INDUSTRY WITH RED HAT JBOSS BRMS

Red Hat JBoss BRMS comprises a high performance rule engine, a rule repository, easy to use rule authoring tools, and complex event processing rule engine extensions. The following use case describes how these features of JBoss BRMS are implemented in insurance industry.

The consumer insurance market is extremely competitive, and it is imperative that customers receive efficient, competitive, and comprehensive services when visiting an online insurance quotation solution. An insurance provider increased revenue from their online quotation solution by upselling relevant, additional products during the quotation process to the visitors of the solution.

The diagram below shows integration of JBoss BRMS with the insurance provider's infrastructure. This integration is fruitful in such a way that when a request for insurance is processed, JBoss BRMS is consulted and appropriate additional products are presented with the insurance quotation.

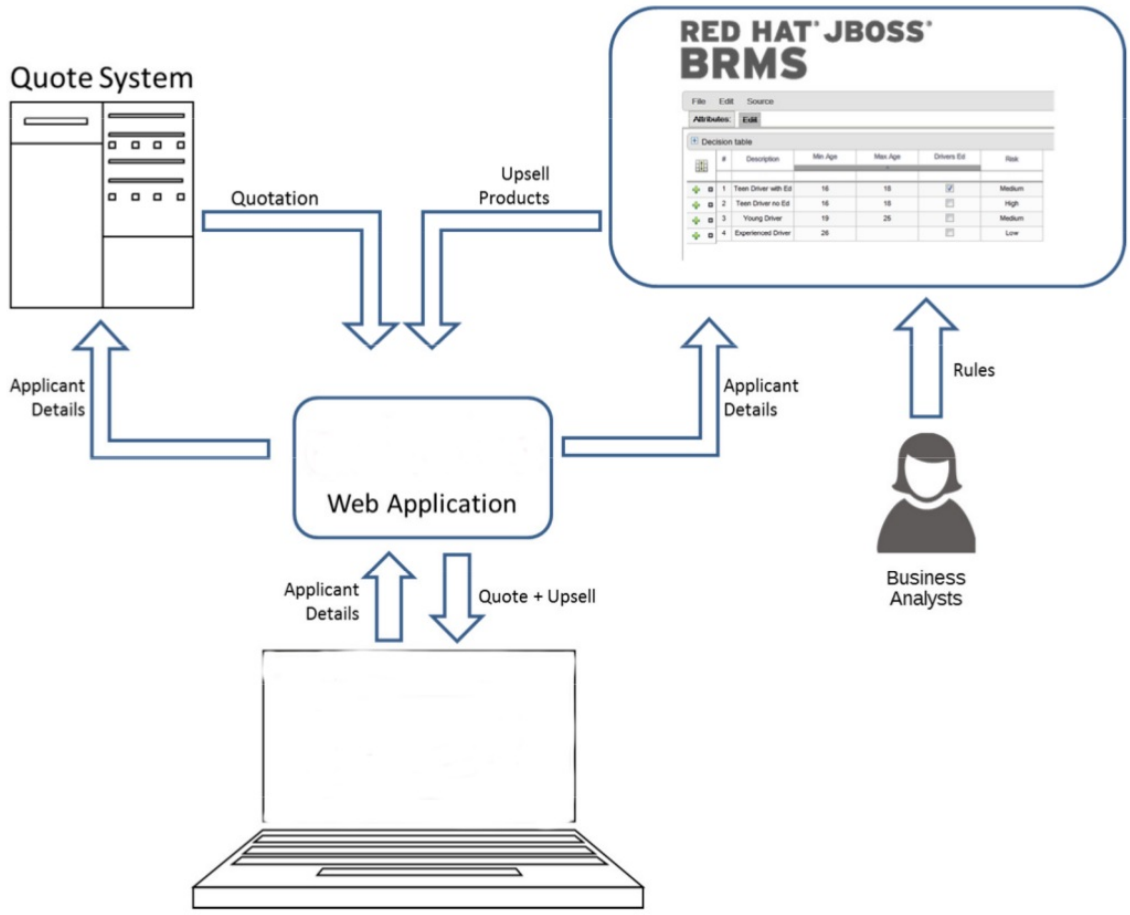


Figure 1.1. JBoss BRMS Use Case: Insurance Industry Decision Making

JBoss BRMS provides the decision management functionality, that automatically determines the products to present to the applicant based on the rules defined by the business analysts. The rules are implemented as decision tables, so they can be easily understood and modified without requiring additional support from IT.

1.3. ASSETS

Anything that can be stored as a version in the artifact repository is an asset. This includes rules, packages, business processes, decision tables, fact models, and DSLs.

Rules

Rules provide the logic for the rule engine to execute against. A rule includes a name, attributes, a 'when' statement on the left hand side of the rule, and a 'then' statement on the right hand side of the rule.

Business Rules

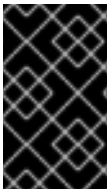
Business Rules define a particular aspect of a business that is intended to assert business structure or influence the behaviour of a business. Business Rules often focus on access control issues, pertain to business calculations and policies of an organization.

Business Processes

Business Processes are flow charts that describe the steps necessary to achieve business goals (see the *Red Hat JBoss BRMS Business Process Management Guide* for more details).

Projects

A project is a container for packages of assets (business processes, rules, work definitions, decision tables, fact models, data models, and DSLs) that lives in the Knowledge Repository. It is this container that defines the properties of the KIE Base and KIE Session that are applied to its content. In the GUI, you can edit these entities in the Project Editor.



IMPORTANT

If an asset, such as a Process or Rule definition, is not placed in a package with a Project, it cannot be deployed. Therefore, make sure to organize your assets in packages. Also note, that the name of the package must be identical with the KIE Session name.

As a project is a Maven project, it contains the Project Object Model file (**`pom.xml`**) with information on how to build the output artifact. It also contains the Module Descriptor file, **`kmodule.xml`**, that contains the KIE Base and KIE Session configuration for the assets in the project.

Packages

Packages are deployable collections of assets. Rules and other assets must be collected into a package before they can be deployed. When a package is built, the assets contained in the package are validated and compiled into a deployable package.

Domain Specific Languages

A domain specific languages, or DSL, is a rule language that is dedicated to the problem domain.

Decision Tables

Decision Tables are collections of rules stored in either a spreadsheet or in the JBoss BRMS user interface as guided decision tables.

Data Model

Data models are a collection of facts about the business domain. The rules interact with the data model in rules-based applications.

CHAPTER 2. BUSINESS CENTRAL

Business Central is the web based user interface used for both Red Hat JBoss BRMS 6 and Red Hat JBoss BPM Suite 6.

It is the user interface for the business rules manager and has been combined with the core drools engine and other tools. It allows a business user to manage rules in a multi user environment and implement changes in a controlled fashion.

The Business Central is used when:

- Users need to manage versions/deployment of rules.
- Multiple users of different skill levels need to access and edit rules.
- You need an infrastructure to manage rules.

Business Central is managed by the Business Analysts, Rule experts, Developers and Administrators (rule administrators).

The main features of the Business Central are:

- Multiple types of rule editors (GUI, text) including:-
  - Guided Rule Editor
  - Rule Templates
  - Decision Tables
- Store multiple rule "assets" together as a package
- Domain Specific Language support
- Complex Event Processing support
- Version control (historical assets)
- Testing of rules
- Validation and verification of rules
- Categorization
- Build and deploy including:-
  - Assembly of assets into a binary package for use with a ChangeSet or KnowledgeBuilder.
- REST API to manipulate assets.

2.1. PERSPECTIVES

Business Central provides the following groups of perspectives accessible from the main menu:

- **Authoring** group:
  - **`Project Authoring`** perspective contains the **`Project Explorer`** view (by default on

the left) with the overview of available repository structure, and information on available resources, such as, business process definitions, form definitions, etc.; the editor area on the right, where the respective editor appears when a resource is opened; and the **`Messages`** view with validation messages.

- **`Artifact Repository`** perspective contains a list of jars which can be added as dependencies. The available operations in this perspective are upload/download artifact and open (view) the **`pom.xml`** file.

- **`Administration`** perspective (available only for users with the **`ADMIN`** role) contains the **`File Explorer`** view (by default on the left) with available asset repositories; the editor area on the right, where the respective editor appears when a resource is opened. The perspective allows an administrator to connect Knowledge Store to a repository with assets and to create a new repository (refer to *Administration and Configuration Guide*).

- **Deploy** group:

  - **`Deployments`** perspective contains a list of the deployed resources and allows you to build and deploy an undeploy new units.

- **Process Management** group:

  - **`Process Definitions`** perspective contains a list of the deployed Process definitions. It allows you to instantiate and manage the deployed Processes.

  - **`Process Instances`** perspective contains a list of the instantiated Processes. It allows you to view their execution worklow and its history.

- **Tasks** group:

  - **`Task List`** perspective contains a list of Tasks produced by Human Task of the Process instances or produced manually. Only Tasks assigned to the logged-in user are visible. It allows you to claim Tasks assigned to a group you are a member of.

- **Dashboards** group (the BAM component):

  - **`Process & Task Dashboard`** perspective contains a prepared dashboard with statistics on runtime data of the Execution Server

  - **`Business Dashboards`** perspective contains the full BAM component, the Dashbuilder, including administration features available for users with the **`ADMIN`** role.

2.2. LOGGING ON TO BUSINESS CENTRAL

Log into Business Central after the server has successfully started.

1. Navigate to http://localhost:8080/business-central in a web browser. If the user interface has been configured to run from a domain name, substitute **`localhost`** for the domain name. For example http://www.example.com:8080/business-central.

2. Log in with the user credentials that were created during installation. For example: User = **`helloworlduser`** and password = **`Helloworld@123`**.

2.3. THE HOME SCREEN

The **Home** view or the "landing page" is the default view for the application. There are two menu items available in this view: **Authoring** and **Deployment**, besides the **Home** menu option.

The following screen shows what the Home view looks like:

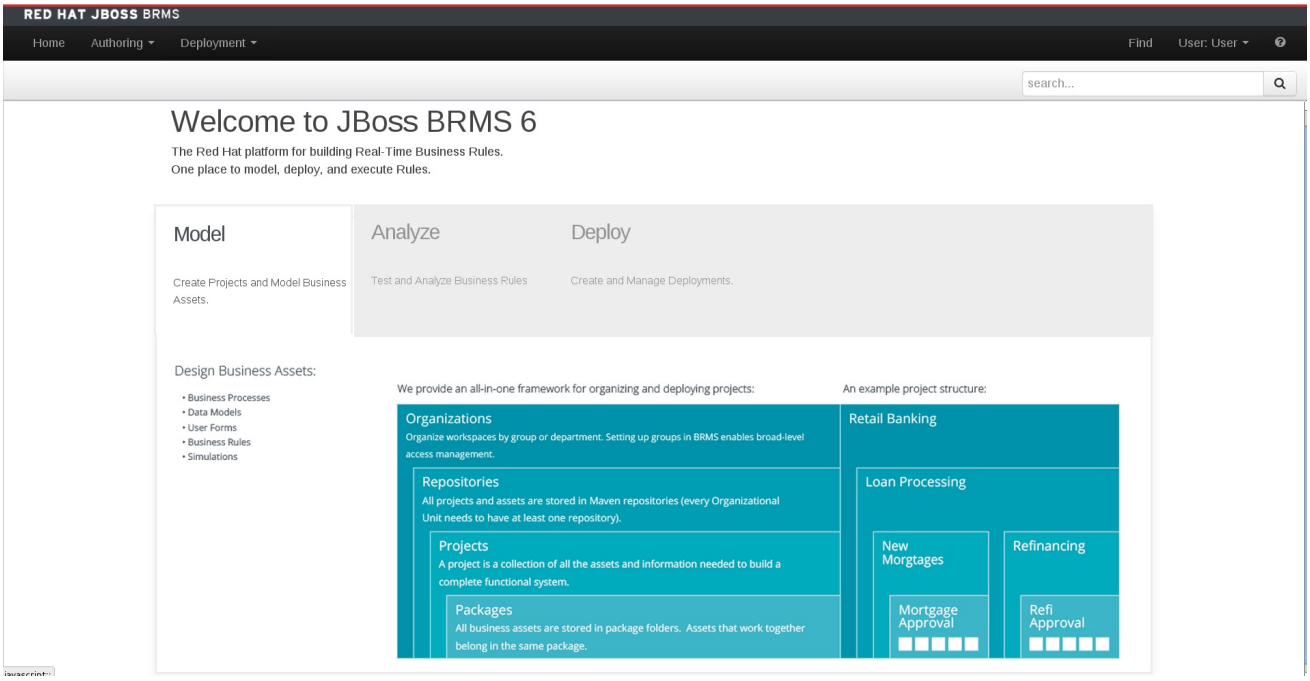


Figure 2.1. Business central home screen

Authoring menu

The Authoring menu with links to Project Authoring and Administration is used to create and maintain the Knowledge Projects (KProjects), rule assets and repositories.

Deployment menu

The Deployment menu allows a user to upload, download and manage the kjar files by using the artifact repository.

2.4. EMBEDDING BUSINESS CENTRAL

Business Central provides a set of editors to author assets in different formats. A specialized editor is used according to the asset format.

Business Central provides the ability to embed it in your own (Web) Applications using standalone mode. This allows you to edit rules, processes, decision tables, et cetera, in your own applications without switching to Business Central.

In order to embed Business Central in your application, you will need the Business Central application deployed and running in a web/application server and, from within your own web applications, an iframe with proper HTTP query parameters as described in the following table.

Table 2.1. HTTP Query Parameters for Standalone Mode

Parameter Name	Explanation	Allow Multiple Values	Example

Parameter Name	Explanation	Allow Multiple Values	Example
standalone	This parameter switches Business Central to standalone mode.	no	(none)
path	Path to the asset to be edited. Note that asset should already exists.	no	git://master@uf-playground/todo.md
perspective	Reference to an existing perspective name.	no	org.guvnor.m2repo.client.perspectives.GuvnorM2RepoPerspective
header	Defines the name of the header that should be displayed (useful for context menu headers).	yes	ComplementNavArea

The following example demonstrates how to set up an embedded Author Perspective for Business Central.

```
===test.html===
 <html>
  <head>
    <title>Test</title>
  </head>
  <body>
    <iframe id="ifrm" width="1920" height="1080"
src='http://localhost:8080/business-central?
standalone=&perspective=AuthoringPerspective&header=AppNavBar'></iframe>
  </body>
</html>
```

X-frame options can be set in **`web.xml`** of business-central. The default value for ***`x-frame-options`*** is as follows:

```
<param-name>x-frame-options</param-name>
  <param-value>SAMEORIGIN</param-value>
```

2.5. PROJECT AUTHORING

Projects and the associated assets can be authored from the Project Explorer. The Project Explorer can be accessed from the Home screen by clicking on **Authoring → Project Authoring**.

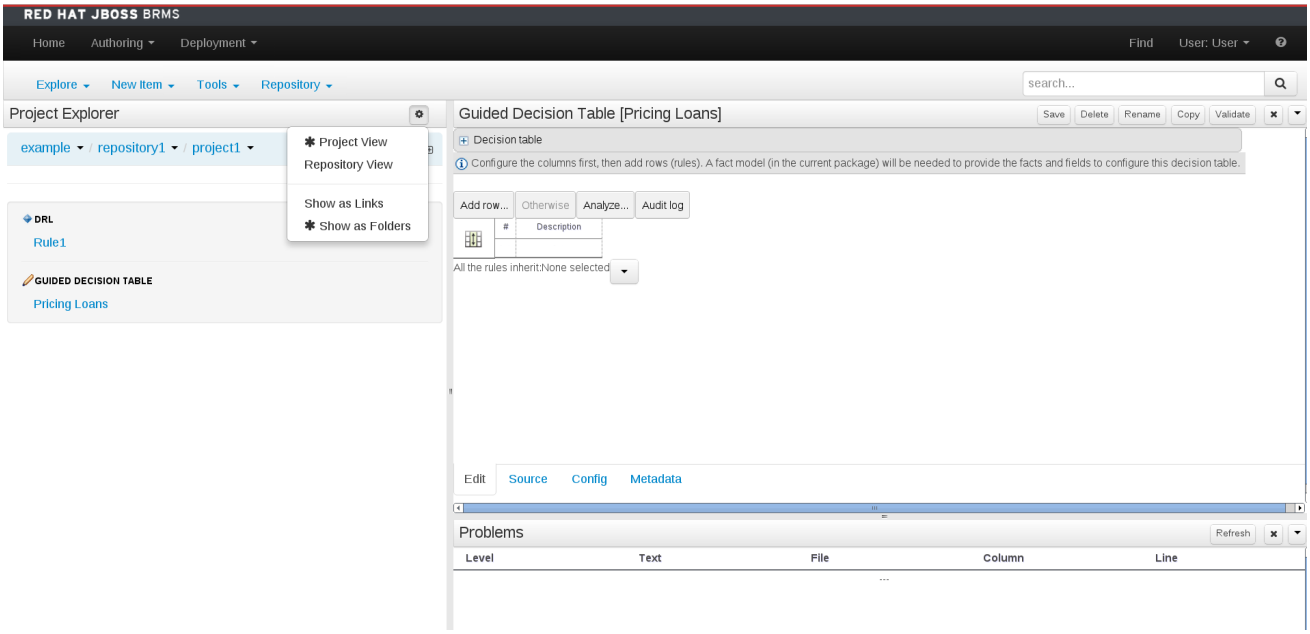


Figure 2.2. The Project Explorer screen

The project authoring screen is divided into 3 sections:

- **`Project Explorer`**: The left pane of the project authoring screen is the project explorer that allows you to navigate through projects and create the required packages and assets. Clicking on the () button allows you to set the view to Project view or Repository view. The contents of the project can be navigated in a tree view by clicking on the **`Show as Folders`** or in a single-line path by clicking on the **`Show as Links`**.

- **`Content area`**: The content area shows the assets which are opened for editing. It has a toolbar with buttons like **`Save`**, **`Delete`**, **`Rename`**, **`Copy`** and **`Validate`** that can be used to perform the required actions on the assets that are being worked upon.

- **`Problems`**: The problems area shows the validation errors of the project that occur while saving or validating a particular asset.

2.5.1. Changing the Layout

The layout of any panel can be changed by the user. Each panel can be resized and repositioned, except for the Project Explorer panel, which can only be resized and not repositioned.

Resizing the layout

The layout can be resized in the following ways:

1. To resize the width of the screen:
   a. Move the mouse pointer over the vertical panel splitter. The pointer changes to  .
   b. Adjust the width of the screen by dragging the splitter and setting it at the required position.
2. To resize the height of the screen:
   a. Hover the cursor over the horizontal panel splitter. The pointer changes to  .

b. Adjust the height of the screen by dragging the splitter and setting the required position.

Repositioning the layout

To reposition the layout, do the following:

1. Move the mouse pointer on the title of the panel. The pointer changes to  .
2. Press and hold the left click of the mouse and drag the screen to the required location. A  symbol indicating the target position is displayed to set the position of the screen.

2.5.2. Creating new assets

Assets can be created using the **`New Item`** perspective menu option.

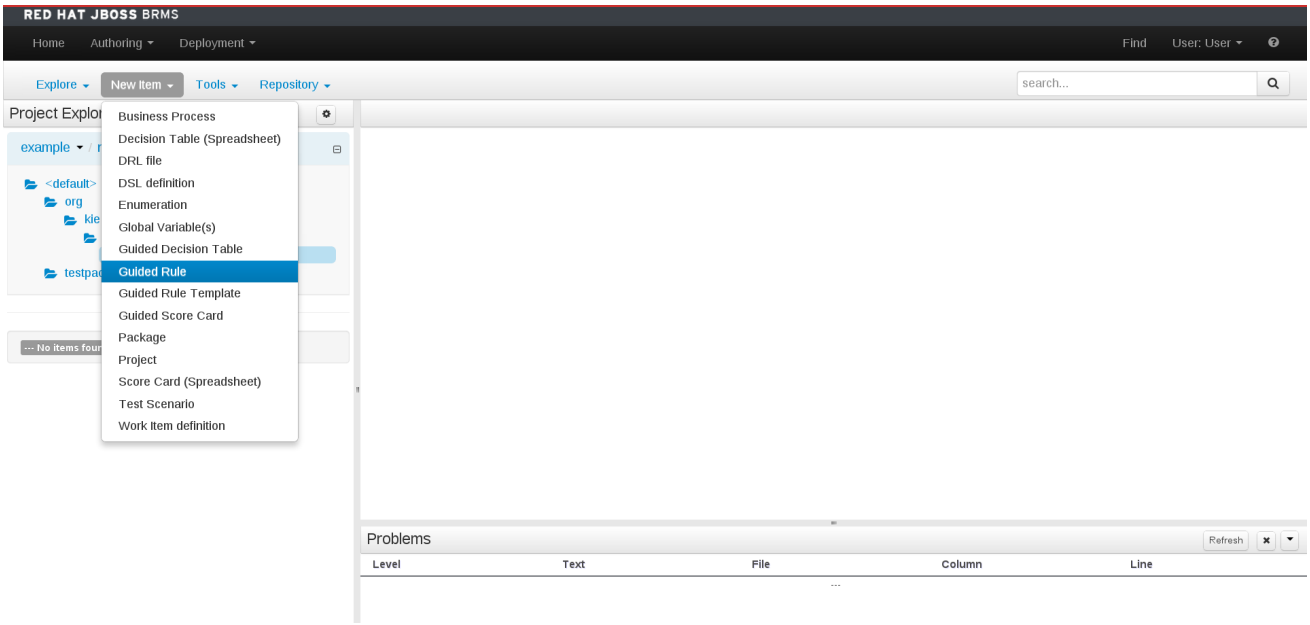


Figure 2.3. Creating new Asset screen

Clicking on a Asset from the **`New Item`** menu will open a **`Create new (Asset-type)`** pop-up dialog where a user can enter the name of the Asset.

Figure 2.4. Create new pop-up dialog

2.5.3. Asset Metadata and Versioning

Most assets within Business Central have some metadata and versioning information associated with them. In this section, we will go through the metadata screens and version management for one such asset (a DRL asset). Similar steps can be used to view and edit metadata and versions for other assets.

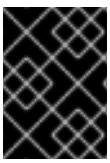
Metadata Management

To open up the metadata screen for a DRL asset, click on the **`Overview`** tab. If an asset doesn't have an **`Overview`** tab, it means that there is no metadata associated with that asset.

The **`Overview`** section opens up in the **`Version history`** tab, and you can switch to the actual metadata by clicking on the **`Metadata`** tab.

The metadata section allows you to view or edit the **`Categories`**, **`Subject`**, **`Type`**, **`External Link`** and **`Source metadata`** for that asset. However, the most interesting metadata is the description of the asset that you can view/edit in the description field and the comments that you and other people with access to this asset can enter and view.

Comments can be entered in the text box provided in the comments section. Once you have finished entering a comment, press enter for it to appear in the comments section.



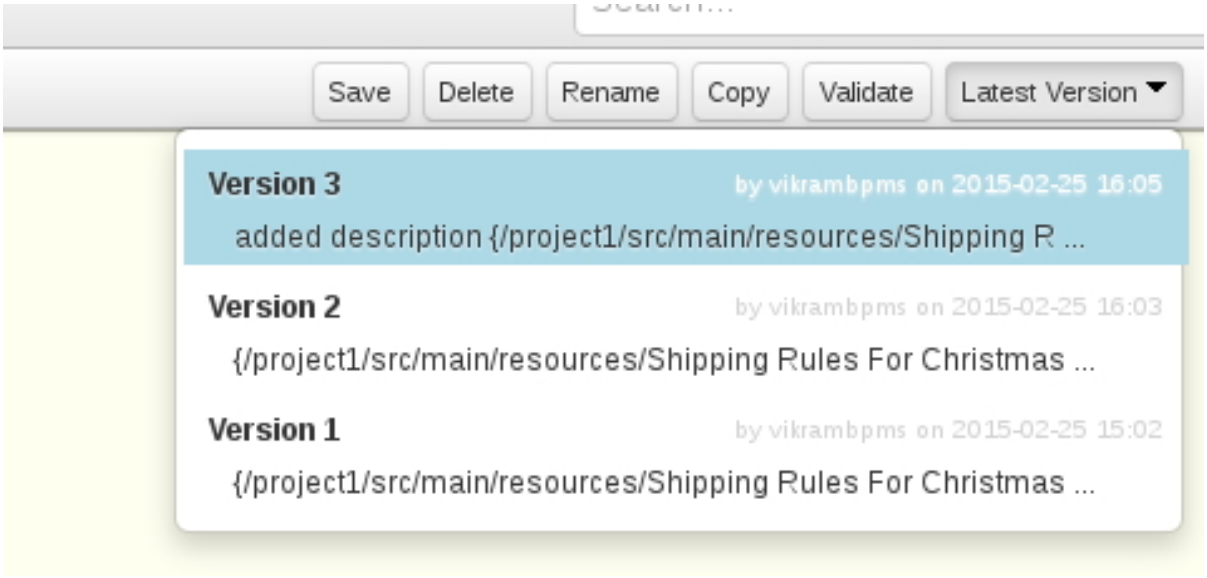
IMPORTANT

You must hit the **`Save`** button for all metadata changes to be persisted, including the comments.

Version Management

Every time you make a change in an asset and save it, a new version of the asset is created. You can switch between different versions of an asset in one of two ways:

- Click the **`Latest Version`** button in the asset toolbar and select the version that you are interested in. Business Central will load this version of the asset.



- Alternatively, open up the **`Overview`** section. The **`Version history`** section shows you all the available versions. **`Select`** the version that you want to restore.

In both cases, the **`Save`** button will change to **`Restore`**. Click this button to persist changes.

2.6. PROJECT EDITOR

2.6.1. The Project Editor

The Project Editor helps a user to build and deploy projects. This view provides access to the various properties of a Red Hat JBoss BRMS Project that can be edited via the Web interface. Properties like Group artifact version, Dependencies, Metadata, Knowledge Base Settings and Imports can be managed from this view. The Project Editor is accessible from the menu bar **Tools** → **Project Editor**. The editor shows the configuration options for the current active project and the content changes when you move around in your code repository.

2.6.2. Project Settings

Project General Settings

The Project settings screen allows a user to set the Group, Artifact, and Version ID's for a project. It edits the **`pom.xml`** setting file since we use Maven to build our projects.

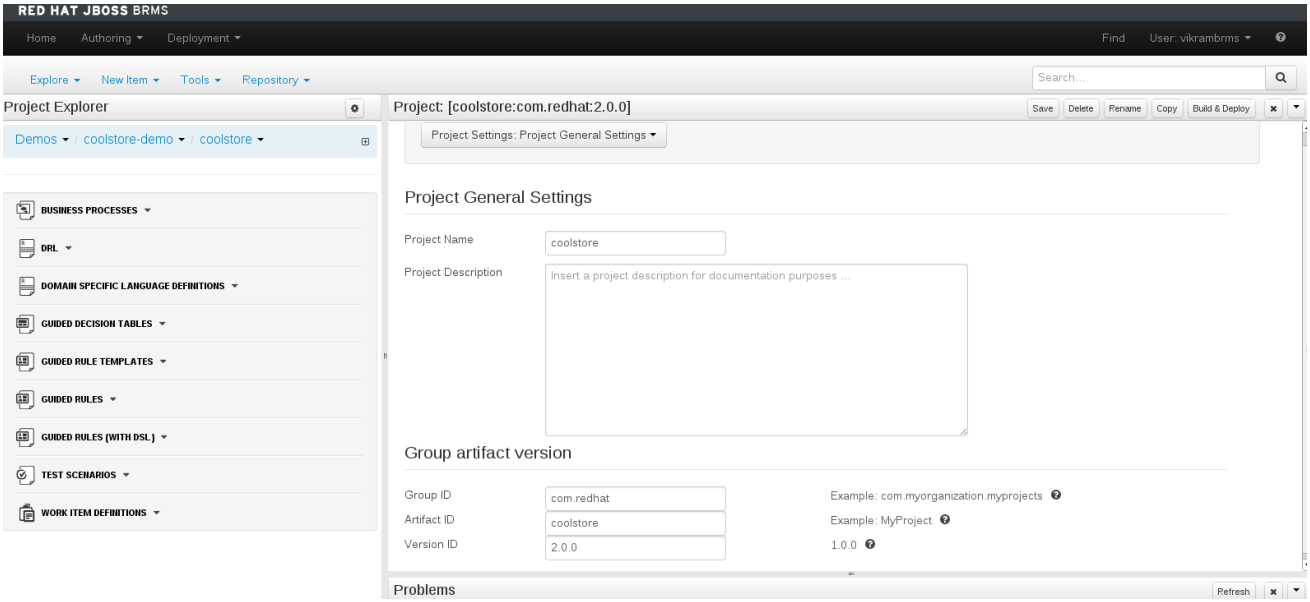


Figure 2.5. Project Editor - Project Settings

Dependencies

The Dependencies option allows you to set the dependencies for the current project. You access the dependencies by using **Project Settings → Dependencies** option. You can add dependencies from the Artifact repository by clicking the **`Add from repository`** button or by entering the Group ID, Artifact ID and Version ID of a project directly by clicking on the **Add** button.

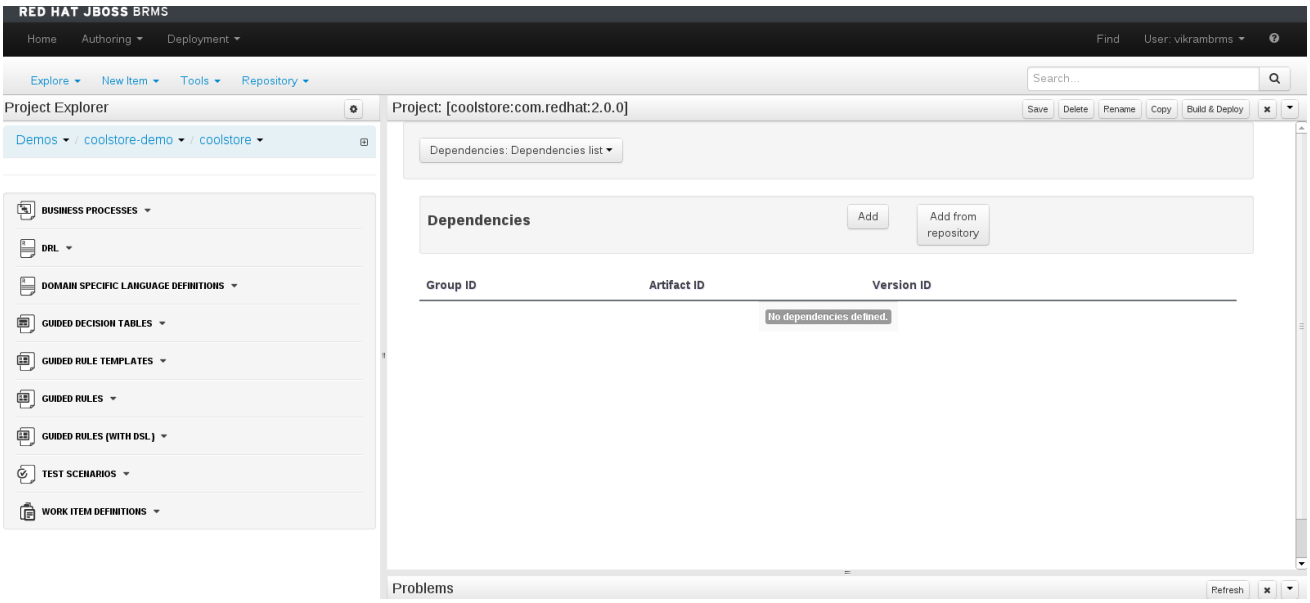


Figure 2.6. Project Editor - Project Dependencies

Metadata

The Metadata screen displays the generic data and version history of a project. It allows a user to edit other metadata details, add descriptions, and participate in discussions which are specific to a selected asset. You can assign categories to assets by clicking the plus icon  next to the **Metadata → Categories** option. By opening an asset, you will see a list of the categories it currently belongs to. If you edit the asset, you will need to save your changes for future execution. Within the **Categories** section is a list of attributes:

- **by** - Who made the last change.

- **`Note`** - Comment from the last asset update.
- **`Created on`** - The date and time the asset was created.
- **`Created by`** - The name of the original asset author.
- **`Format`** - Short format name of the asset type.
- **`URI`** - Unique identifier of the asset.

The other metadata options provide **`Subject`**, **`Type`**, **`External Link`** and **`Source`** options for the asset.

2.6.3. Knowledge Base Settings

Knowledge Bases and Sessions

The Knowledge Base Settings allows the user to create the KIE bases and sessions using the **`kmodule.xml`** project descriptor file of your project. Accordingly, it edits the **`kmodule.xml`** project setting file.

The Knowledge bases and sessions page lists all the knowledge bases by name. It contains the Add, Rename, Delete, and Make Default options above the list. Only one knowledge base can be set as default at a time.

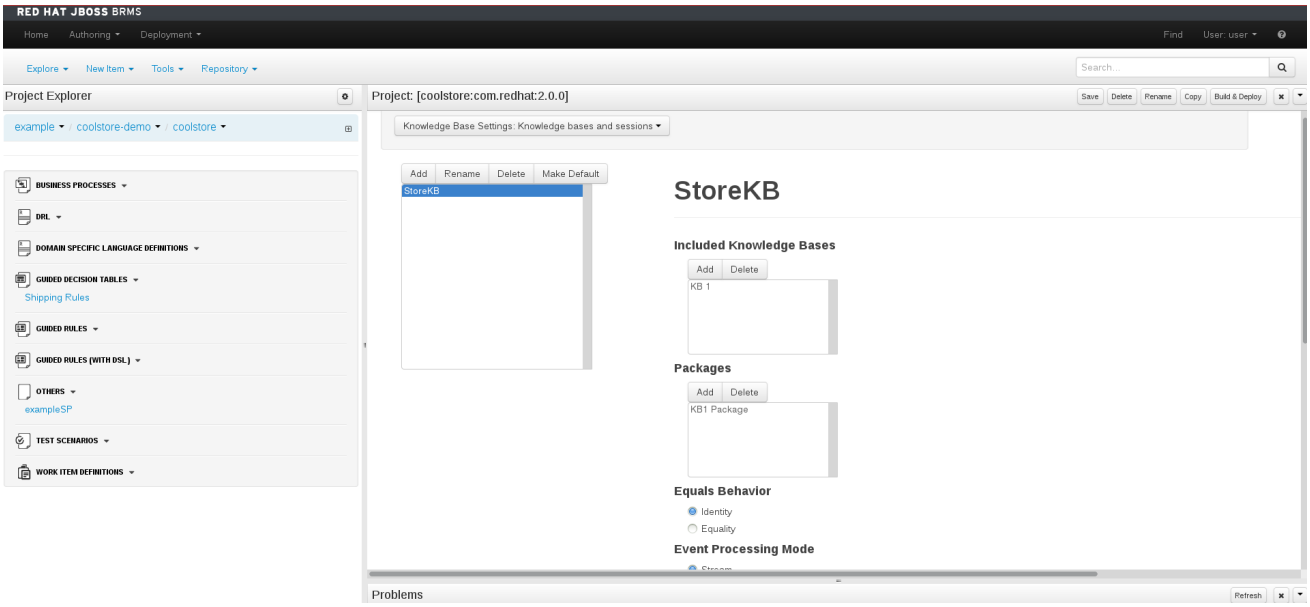


Figure 2.7. Project Editor - Knowledge Base Settings

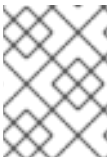
The **Included Knowledge Bases** section displays the models, rules, and any other content in the included knowledge base. It will only be visible and usable by selecting the knowledge base from the Knowledge Base list to the left. You can Add and Delete content from this list.

The **Packages** section allows users to Add and Delete packages which are specified to the knowledge base.

The **Equals Behavior** section allows the user to choose between **`Identity`** or **`Equality`** assertion modes.

- **`Identity`** uses an **`IdentityHashMap`** to store all asserted objects.

- **`Equality`** uses a **`HashMap`** to store all asserted objects.

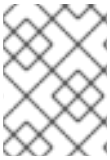


NOTE

Please refer to the kbase attributes section of the Red Hat JBoss BRMS Development Guide for further details about **`Identity`** and **`Equality`** assertion modes.

The **Event Processing Mode** section allows the user to choose between **`Cloud`** and **`Stream`** processing modes.

- **`Cloud`** processing mode is the default processing mode. It behaves in the same manner as any pure forward-chaining rules engine.
- **`Stream`** processing mode is ideal when the application needs to process streams of events.



NOTE

Please refer to the kbase attributes section of the Red Hat JBoss BRMS Development Guide for further details about **`Cloud`** and **`Stream`** processing modes.

The **Knowledge Sessions** table lists all the knowledge sessions in the selected knowledge base. By clicking the 

button, you are able to add a new knowledge session to the table.

- The **`Name`** field displays the name of the session.
- The **`Default`** option can only be allocated to one of each type of session.
- The **`State`** drop-down allows either Stateless or Stateful types.
- The **`Clock`** drop-down allows either Realtime or Pseudo choices.
- Clicking the  opens a pop-up that displays more properties for the knowledge session.

Metadata

Please refer to Section 2.6.2, “Project Settings” for more information about Metadata.

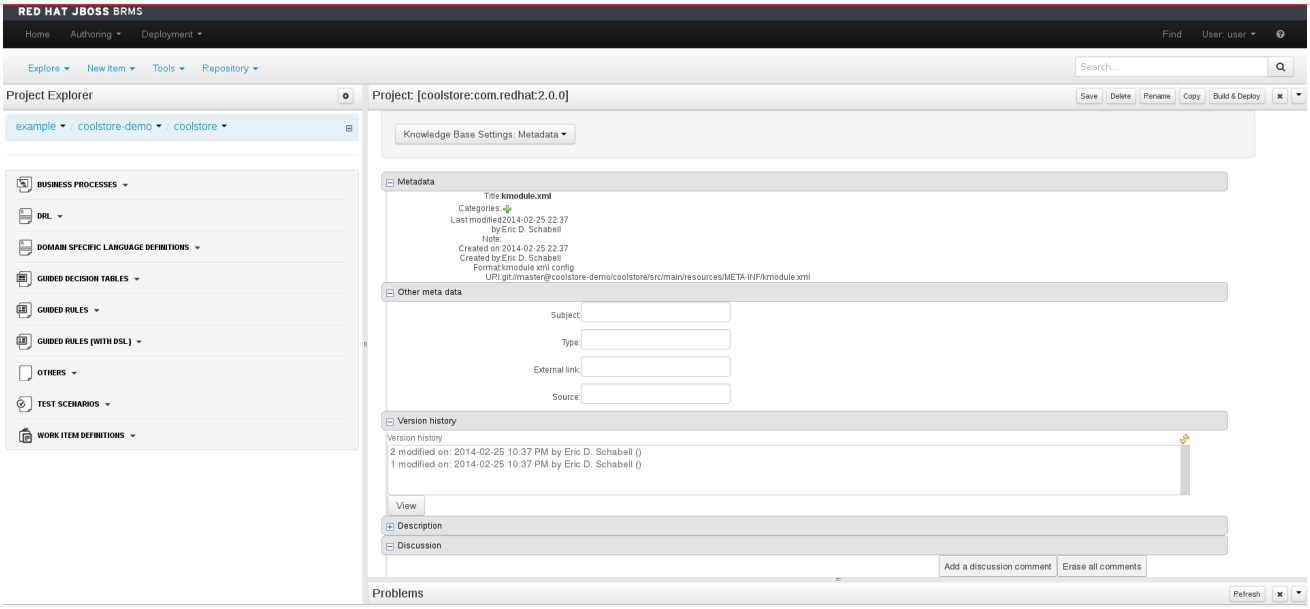


Figure 2.8. Knowledge Base Settings - Metadata

2.6.4. Imports

Import Suggestions

The Import suggestions specify a set of imports used in the project. Each asset in a project has its own imports. The imports are used as suggestions when using the guided editors the workbench offers; accordingly, this makes it easier to work with the workbench as there is no need to type each import in every file that uses it. By changing the Import settings, the **`project.imports`** setting files are edited.

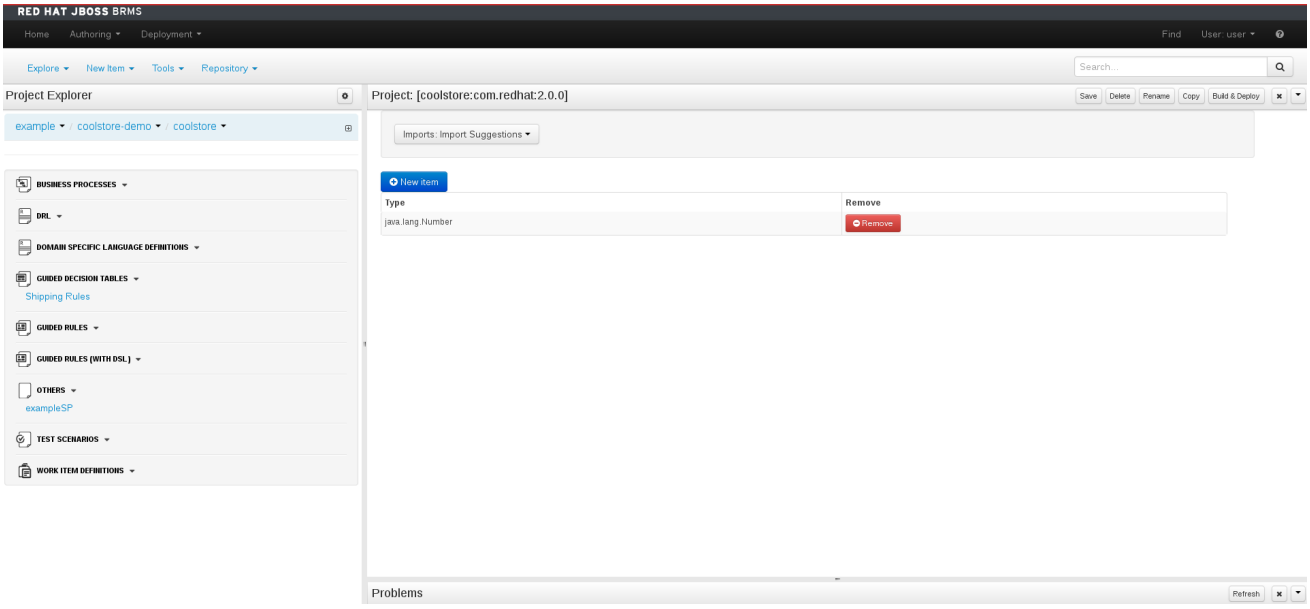


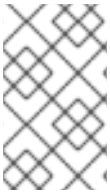
Figure 2.9. Project Editor - Imports

To add a fact model to the imports section, press the 

button. This displays a pop-up dialog to Add Import information. Once the Import Type has been entered, press OK.

To remove a fact model from the imports section, select the fact and click the 

button.



NOTE

The imports listed in the import suggestions are not automatically added into the knowledge base or into the packages of the workbench. Each import needs to be added into each file.

Metadata

Please refer to Section 2.6.2, “Project Settings” for more information about Metadata.

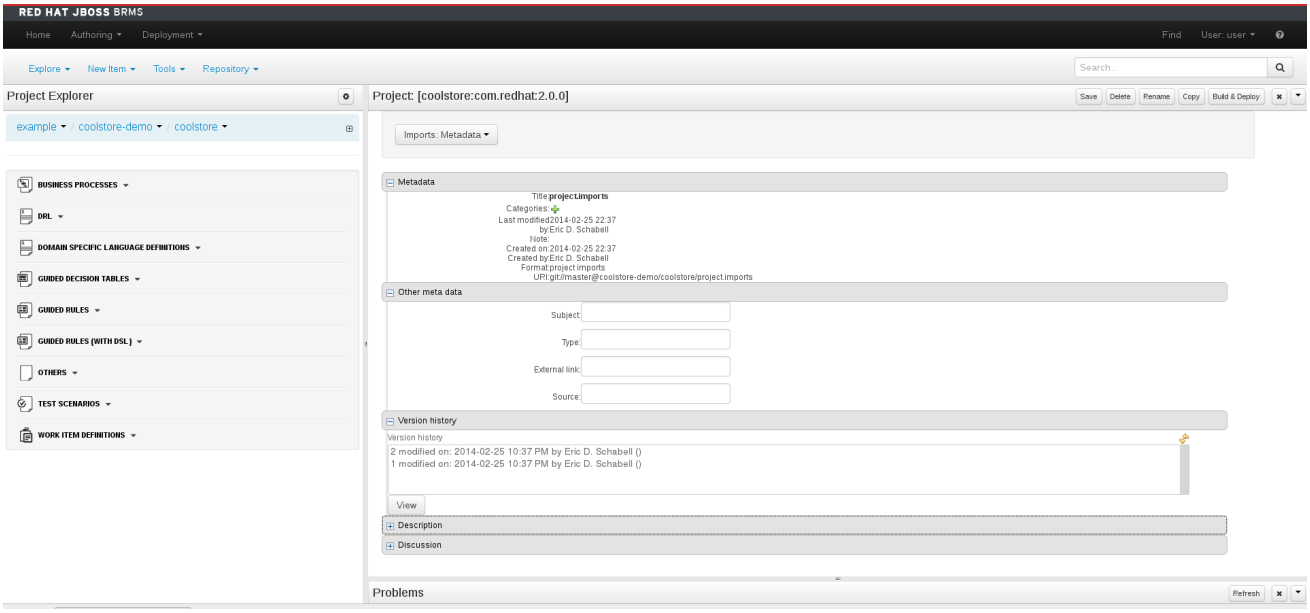


Figure 2.10. Imports - Metadata

2.7. ADMINISTRATION MENU

You can manage Organizational Units and Repositories from the Administration view. Click **Authoring → Administration** to get to this view. For more details on creating and managing these assets, please see Chapter 3, *Setting up a new project*.

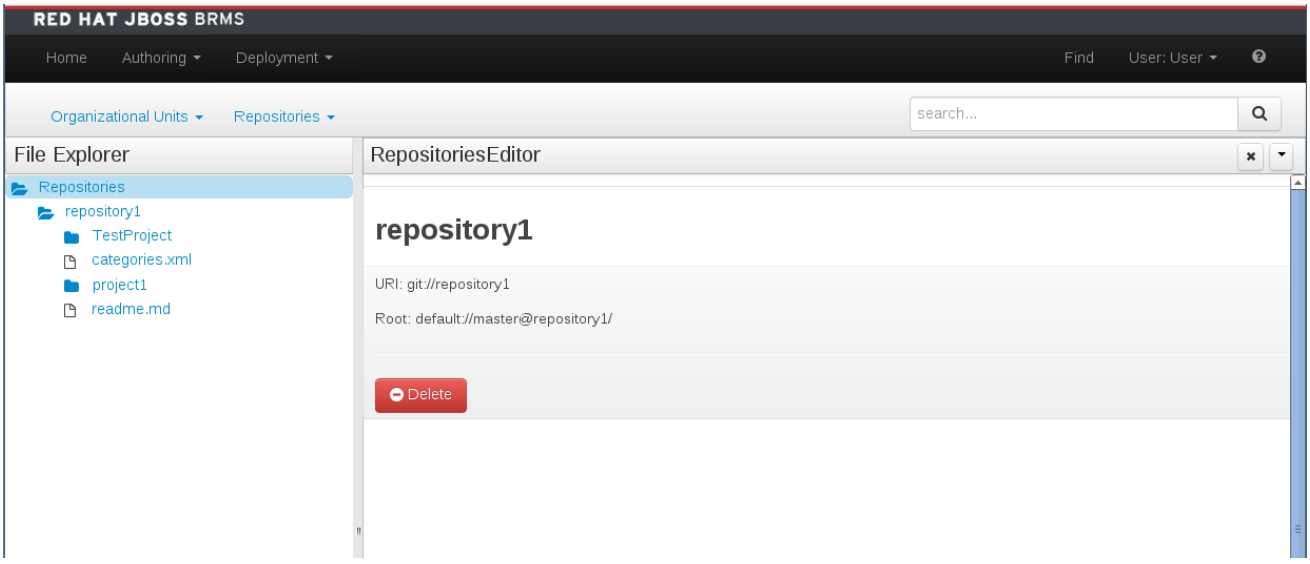


Figure 2.11. The Administration Screen

2.8. RENAME, COPY, DELETE ASSETS

2.8.1. Renaming a file or folder

Users can rename a file or a folder directly in Project Explorer.

1. To rename a file or a folder, open **`Project Explorer`** by selecting **Authoring → Project Authoring**.
2. Click the Gear  icon in the upper right hand corner of the **`Project Explorer`** view and in the menu that opens, select **`Repository View`**. Click the gear icon again to select the option **`Show as Links`** (if not already selected).
3. Click the Rename  icon to the right of the file or folder you want to rename. In the displayed **`Rename this item`** dialog box, enter the new name and click the **`Rename item`** button.

2.8.2. Deleting a file or folder

Users can delete a file or a folder directly in Project Explorer.

1. To delete a file or a folder, open **`Project Explorer`** by selecting **Authoring → Project Authoring**.
2. Click the Gear icon () in the upper right hand corner of the **`Project Explorer`** view and in the menu that opens, select **`Repository View`**. Click the gear icon again to select the option **`Show as Links`** (if not already selected).
3. Click the Delete icon () to the right of the file or folder you want to rename. In the displayed **`Delete this item`** dialog box, click the **`Delete item`** button.

2.8.3. Copying a file or folder

Users can copy a file or a folder directly in Project Explorer.

1. To copy a file or a folder, open **`Project Explorer`** by selecting **Authoring → Project Authoring**.
2. Click the Gear  icon in the upper right hand corner of the **`Project Explorer`** view and in the menu that opens, select **`Repository View`**. Click the gear icon again to select the option **`Show as Links`** (if not already selected).
3. Click the Copy  icon to the right of the file or folder you want to copy. In the displayed **`Copy this item`** dialog box, enter the new name and click the **`Create copy`** button.

2.9. DEPLOYMENT MENU: THE ARTIFACT REPOSITORY

The **`Artifact Repository`** explores the Guvnor M2 repository. It shows the list of available kjar files used by the existing projects and allows a user to upload, download and manage the kjar files. It can be accessed by clicking on the **Authoring → Artifact Repository** menu on the toolbar.

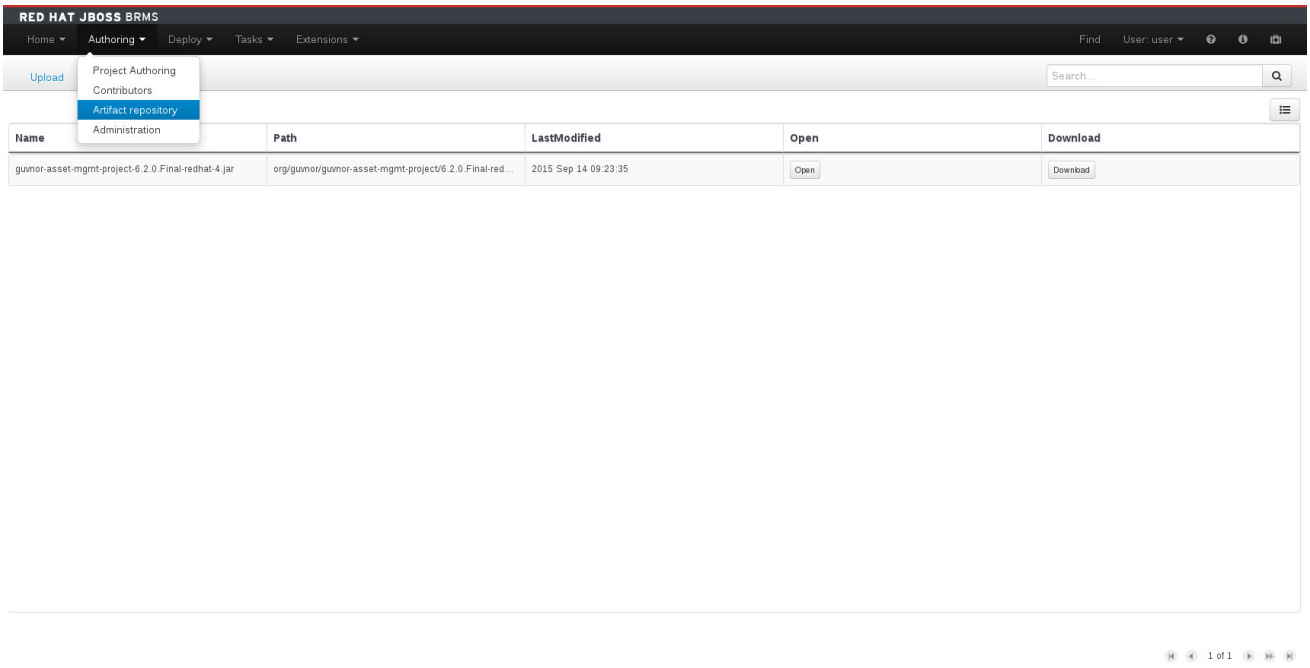


Figure 2.12. The Artifact Repository Screen

CHAPTER 3. SETTING UP A NEW PROJECT

To create a project a business user has to create an organizational unit. An organizational unit is based on any domain in a particular business sector. It holds the repositories, where projects and packages can be created. Packages are deployable collections of assets like rules, fact models, decision tables and so on, that can be validated and compiled for deployment.

3.1. CREATING AN ORGANIZATIONAL UNIT

IMPORTANT

Note that only user with the **`ADMIN`** role can create an organizational unit.

Procedure 3.1. Creating an organizational unit

1. Open the **`Administration`** perspective: on the main menu, click **Authoring** → **Administration**.
2. On the perspective menu, click **Organizational Units** → **Manage Organizational Units**.
3. In the **`Organization Unit Manager`** view, click on the **`Add`** button. The **`Add New Organizational Unit`** pop-up window opens.

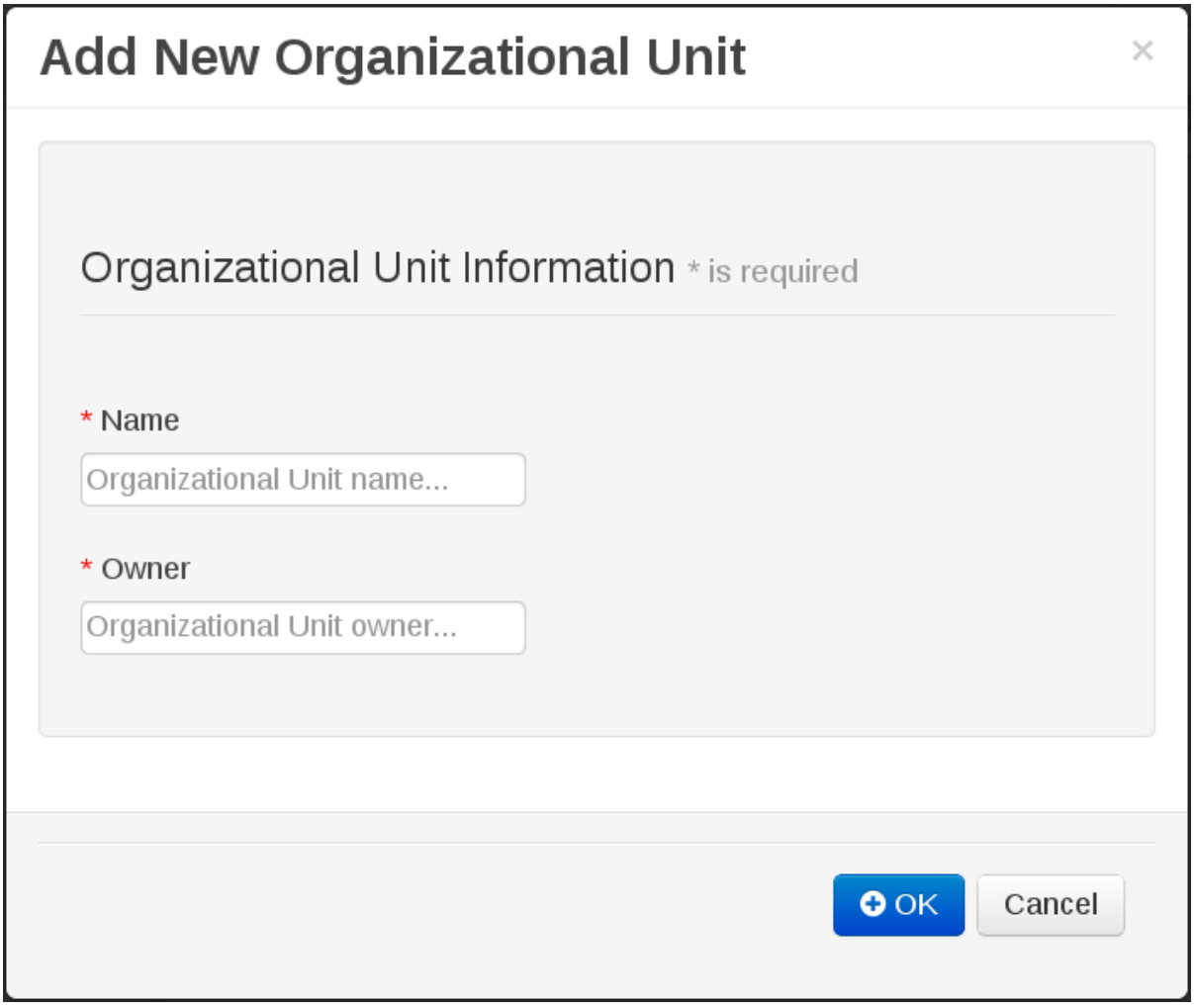


Figure 3.1. Add New Organizational Unit Pop-up

4. Enter the mandatory details:
   - Organizational unit name.
   - Owner.
5. Click **OK** to create the unit.

3.2. CREATING A REPOSITORY

IMPORTANT

Note that only user with the **ADMIN** role can create a repository.

Procedure 3.2. Creating a New Repository

1. Open the **Administration** perspective: on the main menu, click **Authoring** → **Administration**.
2. On the perspective menu, click **Repositories** → **New Repository**.
3. The **Create Repository** pop-up window is displayed.

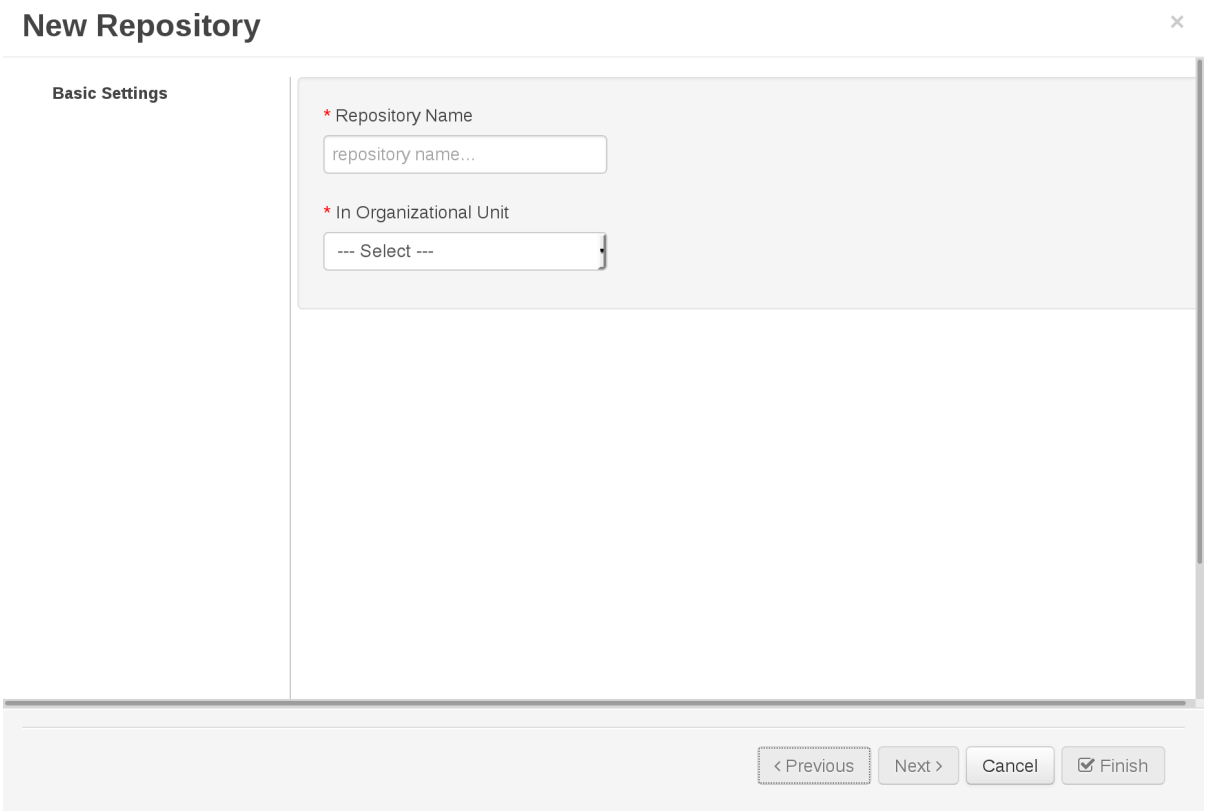


Figure 3.2. Create Repository Pop-up

4. Enter the mandatory details:
   - Repository name.

NOTE

Note that the repository name should be a valid filename. Avoid using a space or any special character that might lead to an invalid folder name.

- Select an organizational unit in which the repository is to be created from the **`Organizational Unit`** drop-down option.

5. Click **`Finish`**

The new repository can be viewed either in the **`File Explorer`** or **`Project Explorer`** views.

3.3. CLONING A REPOSITORY

IMPORTANT

Note that only user with the **`ADMIN`** role can clone a repository.

Procedure 3.3. Cloning a repository

1. Open the **`Administration`** perspective.
2. On the **`Repositories`** menu, select **Clone repository**.
3. The **`Clone Repository`** pop-up window is displayed.

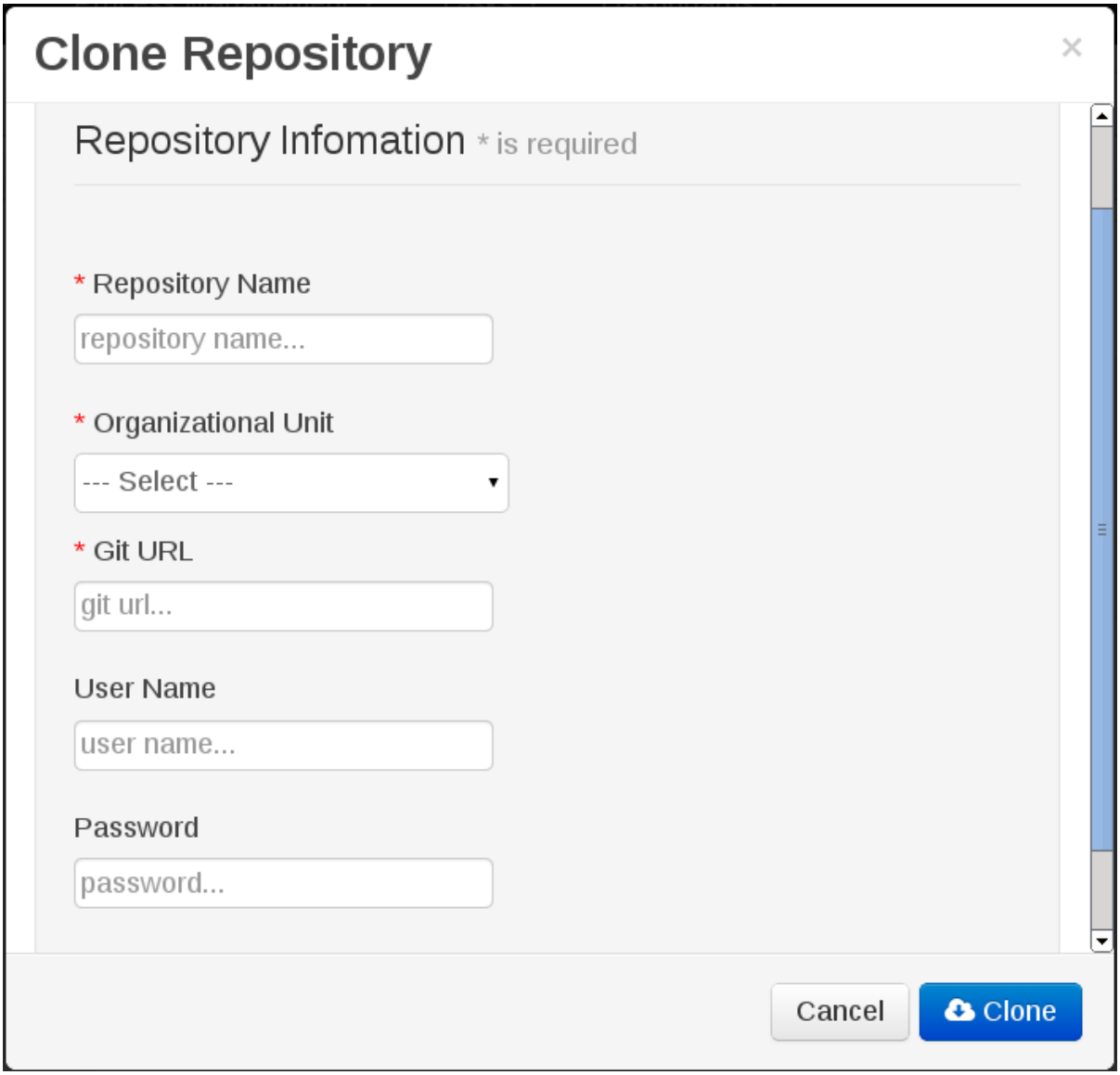


Figure 3.3. Clone Repository Pop-up

4. In the **Clone Repository** dialog window, enter the repository details:

   a. Enter the **Repository Name** to be used as the repository identifier in the Asset repository and select the **Organizational Unit** it should be added to.

   b. Enter the URL of the GIT repository:

      - For a Local Repository: **file:///*path-to-repository/reponame***
      - For a Remote or preexisting Repository: **git://*hostname/reponame***

      NOTE

      The file protocol is only supported for 'READ' operations. 'WRITE' operations are *not* supported.

   c. If applicable, enter the **User Name** and **Password** to be used for authentication when cloning the repository.

5. Click **Clone**.

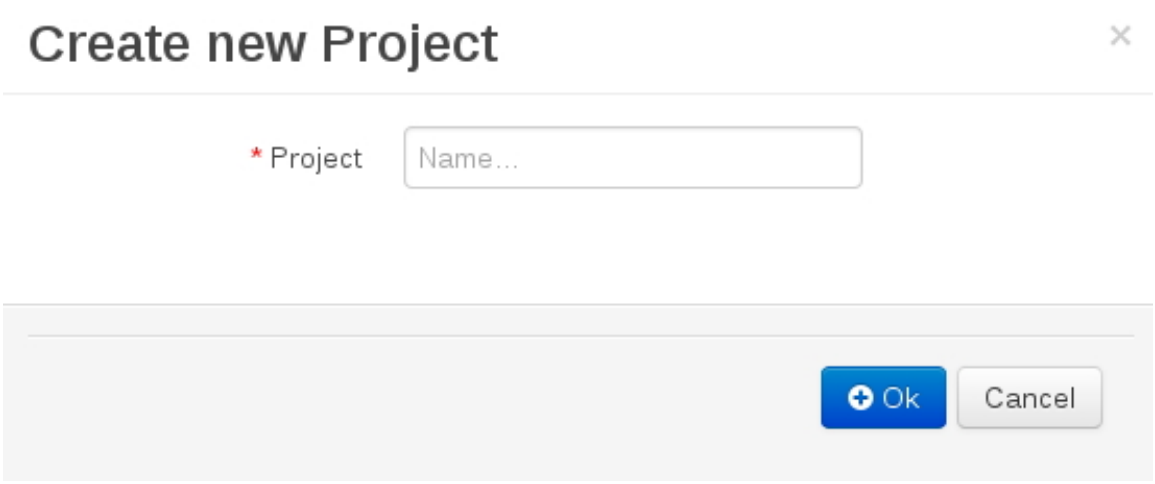
6. A confirmation prompt with an **OK** button is displayed which notifies the user that the repository is created successfully. Click **OK**.The repository will be indexed. Some workbench features may be unavailable until indexing has completed.

The cloned repository can be checked either in the **File Explorer** or **Project Explorer** views.

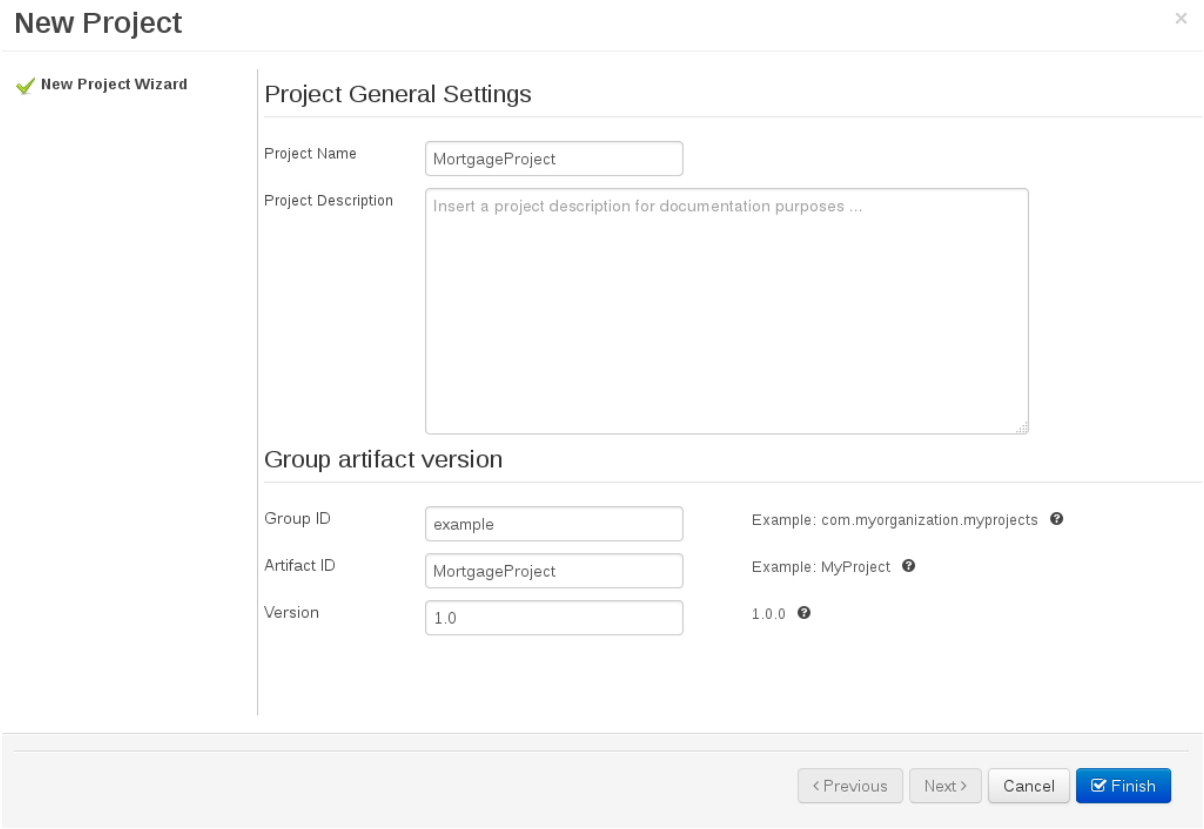
3.4. CREATING A PROJECT

To create a project, do the following:

1. Open the **Project Authoring** perspective: on the main menu, click **Authoring** → **Project Authoring**.
2. In the **Project Explorer**, select the organizational unit and the repository where you want to create the project.
3. In the perspective menu, go to **New Item** → **Project**.
4. In the **Create new Project** dialog window, define the project details:
   a. In the **Project** text box, enter the project name.



5. The explorer refreshes to show a **New Project Wizard** pop-up window.



6. Define the **Project General Settings** and **Group artifact version** details for this new project. These parameters are stored inside the **pom.xml** Maven configuration file.
   - **Project Name**: The name for the project; for example **MortgageProject**
   - **Project Description**: The description of the project which may be useful for the project documentation purpose.
   - **Group ID**: group ID of the project; for example **org.mycompany.commons**
   - **Artifact ID**: artifact ID unique in the group; for example **myframework**. Avoid using a space or any special character that might lead to an invalid name.
   - **Version ID**: version of the project; for example **2.1.1**

   The **Project Screen** view is updated with the new project details as defined in the **pom.xml** file. Note, that you can switch between project descriptor files in the drop down-box with **Project Settings** and **Knowledge Base Setting**, and edit their contents.

3.5. CREATING A NEW PACKAGE

Procedure 3.4. Creating a new package

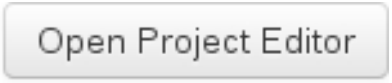
1. Open the **Project Authoring** perspective: on the main menu, click **Authoring → Project Authoring**.
2. In the **Project Explorer** view, do the following:
   - If in the Project view of Project Explorer, select the organizational unit, repository and the project where you want to create the package.

- If in the Repository view of Project Explorer, navigate to the project root, where you want to create the package.

3. In the perspective menu, go to **New Item** → **Package**.
4. In the **`Create new Package`** dialog window, define the package details:
   a. In the **`Resource Name`** text box, enter the package name and click **`OK`**.
5. The new package is now created under the selected project.

3.6. ADDING DEPENDENCIES

To add dependencies to your project, do the following:

1. Open the Project Editor for the given project:
   a. In the **`Project Explorer`** view of the **`Project Authoring`** perspective, open the project directory.
   b. Click on the 

button to open the project view.
2. In the **`Project Screen`** view, select in the **`Project Settings`** drop-down box the **`Dependencies`** item.
3. On the updated **`Project Screen`**, click the **`Add`** button to add a maven dependency or click the **`Add from repository`** button to add a dependency from the Knowledge Store (Artifact repository):
   - When adding a maven dependency, a user has to define the **`Group ID`**, **`Artifact ID`** and the **`Version ID`** in the new row which is created in the dependency table.
   - When adding a dependency from the Knowledge Store, select the dependency in the displayed dialog box: the dependency will be added to the dependency table.
4. To apply the various changes, the dependencies must be saved.

WARNING

If working with modified artifacts, do not re-upload modified non-snapshot artifacts as Maven will not know these artifacts have been updated, and it will not work if it is deployed in this manner.

3.7. DEFINING KIE BASES AND SESSIONS

You can create KIE bases and sessions by editing the **`kmodule.xml`** project descriptor file of your project. You can edit this file via the user interface provided in the Project Editor or by directly editing this file under the **`src/main/resources/META-INF/`** folder by navigating through the **`Repository`** view.

Defining KIE bases and sessions in the Project Editor

To define a KIE base or session in the web environment in the **kmodule.xml** file, do the following:

1. Open your project properties with the Project Editor: in the Project Explorer, locate your project root. In the top menu, go to **Tools** → **Project Editor**.
2. In the **Project Screen** tab on the right, select "Knowledge Base Settings: Knowledge bases and sessions" option. This brings up a user interface for editing the **kmodule.xml** file.
3. Click on **Add** button to define and add your bases. Scroll down in this screen to add your knowledge sessions.
4. When **KieBases** are defined in **kmodule.xml**, one of them must be marked as default. This is also true for **KieSessions** and **StatelessKieSessions**. This is achieved by the default="true/false" attribute.
5. Click the **save** button in the top right once done.

Defining KIE bases and sessions in the kmodule.xml file directly.

To define a KIE base or session in the **kmodule.xml** file directly, do the following:

1. Open up the repository view for your project by clicking the gear icon.

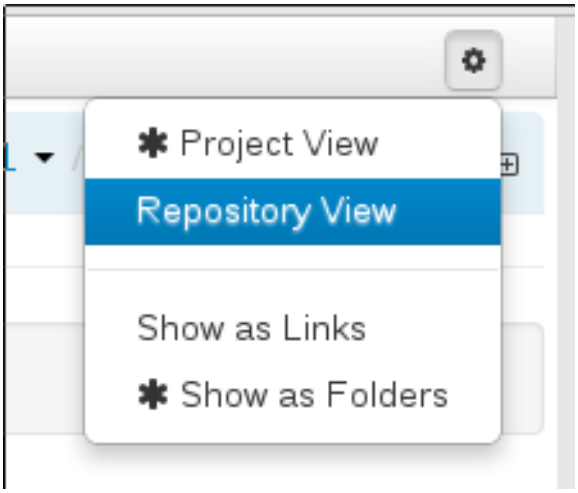


Figure 3.4. Changing to repository view

2. Click on **src** folder, and then drill down to locate the **META-INF** folder (**/src/main/resources/META-INF**). You will see the **kmodule.xml** file. Click on this file and its raw view will open up in a panel on the right hand side.
3. Define your **kbases** and **ksessions** in this file panel. The example code snippet below illustrates this process; however, it should be added to an existing **kmodule.xml** file and not used to replace the content entirely.

```
<kbase name="TestBase" default="false" eventProcessingMode="stream"
equalsBehavior="equality"> <ksession name="TestSession"
type="stateless" default="false" clockType="realtime"/> </kbase>
```

4. Click **save** button in the top right hand screen once done.

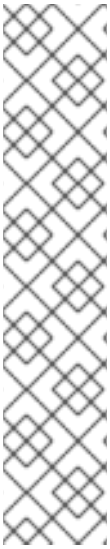
NOTE

You can switch between the Project Editor view and the Repository view to look at the changes you make in each view, but to do so, you must close and reopen the view each time a change is made.

3.8. CREATING A RESOURCE

A Project may contain an arbitrary number of packages, which contain files with resources, such as Process definition, Work Item definition, Form definition, Business Rule definition, etc.

To create a resource, select the Project and the package in the **`Project Explorer`** and click **New Item** on the perspective menu and select the resource you want to create.



NOTE

It is recommended to create your resources, such as Process definitions, Work Item definitions, Data Models, etc., inside a package of a Project to allow importing of resources and referencing their content.

To create a package, do the following:

- In the **`Repository`** view of the Project Explorer, navigate to the ***`REPOSITORY/PROJECT`*`/src/main/resources/`** directory.
- Go to **New Item** → **Package**.
- In the **`New resource`** dialog, define the package name and check the location of the package in the repository.

3.9. SYSTEM PROPERTIES

The following is a list of all system properties:

Table 3.1. System Properties

Property	Description
org.uberfire.nio.git.dir	Location of the directory **`.niogit`**. Default: working directory
org.uberfire.nio.git.daemon.enabled	Enables/disables git daemon. Default: **`true`**
org.uberfire.nio.git.daemon.host	If git daemon enabled, uses this property as local host identifier. Default: **`localhost`**
org.uberfire.nio.git.daemon.port	If git daemon enabled, uses this property as port number. Default: **`9418`**
org.uberfire.nio.git.ssh.enabled	Enables/disables ssh daemon. Default: **`true`**

Property	Description
org.uberfire.nio.git.ssh.host	If ssh daemon enabled, uses this property as local host identifier. Default: **`localhost`**
org.uberfire.nio.git.ssh.port	If ssh daemon enabled, uses this property as port number. Default: **`8001`**
org.uberfire.nio.git.ssh.cert.dir	Location of the directory **`.security`** where local certtificates will be stored. Default: working directory
org.uberfire.metadata.index.dir	Place where Lucene **`.index`** folder will be stored. Default: working directory
org.uberfire.cluster.id	Name of the helix cluster, for example: **`kie-cluster`**
org.uberfire.cluster.zk	Connection string to zookeeper. This is of the form **`host1:port1`**,**`host2:port2`**,**`host3:port3`**, for example: **`localhost:2188`**
org.uberfire.cluster.vfs.lock	Name of the resource defined on helix cluster, for example: **`kie-vfs`**
org.uberfire.cluster.autostart	Delays VFS clustering until the application is fully initialized to avoid conflicts when all cluster members create local clones. Default: **`false`**
org.uberfire.sys.repo.monitor.disabled	Disable configuration monitor (do not disable unless you know what you're doing). Default: **`false`**
org.uberfire.secure.key	Secret password used by password encryption. Default: **`org.uberfire.admin`**
org.uberfire.secure.alg	Crypto algorithm used by password encryption. Default: **`PBEWithMD5AndDES`**
org.uberfire.domain	Security-domain name used by uberfire. Default: **`ApplicationRealm`**
org.guvnor.m2repo.dir	Place where Maven repository folder will be stored. Default: working-directory/repositories/kie
org.kie.example.repositories	Folder from where demo repositories will be cloned. The demo repositories need to have been obtained and placed in this folder. Demo repositories can be obtained from the kie-wb-6.2.0-SNAPSHOT-example-repositories.zip artifact. This System Property takes precedence over org.kie.demo and org.kie.example. Default: Not used.

Property	Description
org.kie.demo	Enables external clone of a demo application from GitHub. This System Property takes precedence over org.kie.example. Default: **`true`**
org.kie.example	Enables example structure composed by Repository, Organization Unit and Project. Default: **`false`**

To change one of these system properties in a WildFly or JBoss EAP cluster:

- Edit the file $ **`JBOSS_HOME/domain/configuration/host.xml`**.
- Locate the XML elements server that belong to the ***`main-server-group`*** and add a system property, for example:

```
<system-properties>
  <property name="org.uberfire.nio.git.dir" value="..." boot-
time="false"/>
  ...
</system-properties>
```

CHAPTER 4. SOCIAL EVENTS

In Red Hat JBoss BRMS 6.1, users can follow other users and gain an insight into what activities are being performed by that user. They can also listen for and follow timelines of regular events. This capability comes via the implementation of a Social Activities framework. This framework ensures that event notifications are generated by different activities within the system and that these notifications are broadcast for registered actors to view.

Multiple activities trigger events. These include: new repository creation, adding and updating resources and adding and updating processes. With the right credentials, a user can view these notifications once they are logged into Business Central.

FOLLOW USER

To follow a user, search for the user by entering his name in the search box in the *People* perspective. You get to this perspective by navigating to it from **Home** → **People**.

You must know the login name of the other user that you want to follow. As you enter the name in the search box, the system will try and auto-complete the name for you and display matches based on your partial entry. Select the user that you want to follow from these matches and the perspective will update to display more details about this user.

You can choose to follow the user by clicking on the **`Follow`** button. The perspective refreshes to showcase the user details and their recent activities.

ACTIVITY TIMELINE

Click on **Home** → **Timeline** to see a list of recent assets that have been modified (in the left hand window) and a list of changes made in the selected repository in the right hand side. You can click on the assets to directly open the editor for the assets (if you have the right permissions).

CHAPTER 5. DATA MODELS

Data models are models of data objects. A data object is a custom complex data type (for example, a Person object with data fields Name, Address, and Date of Birth).

Data models are saved in data models definitions stored in your Project. Red Hat JBoss BRMS provides the Data modeler, a custom graphical editor, for defining data objects.

IMPORTANT

Every data object is implemented as a POJO and you need to import its class explicitly into your Process definition to allow the Process definition to see the data object.

5.1. DATA MODELER

The Data Modeler is the built-in editor for creating facts or data objects as part of a Project data model from the Business Central. Data objects are custom data types implemented as POJOs. These custom data types can be then used in any resource (such as a Guided Decision Table) after they have been imported.

To open the editor, open the Project Authoring perspective, click **New Item** → **Data Object** on the perspective menu. If you want to edit an existing model, these files are located under **`Data Objects`** in **`Project Explorer`**.

You will be prompted to enter the name of this model object, when creating a new model, and asked to select a location for it (in terms of the package). On successful addition, it will bring up the editor where you can create fields for your model object.

The Data Modeler supports roundtrips between the **`Editor`** and **`Source`** tabs, along with source code preservation. This allows you to make changes to your model in external tools, like JBDS, and the Data Modeler updates the necessary code blocks automatically.

5.2. CREATING A DATA OBJECT

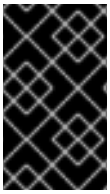
1. Open the Data Modeler: in the Project Authoring perspective, click **New Item** → **Data Object** on the perspective menu. Enter the name (unique across the whole project and not just the package) and the location and press the **`Ok`** button.

2. Create fields of the data object:

   a. In the **`Create new field`** part of the **`Fields`** panel, define the field properties:

   - **`Id`**: field ID unique within the data object
   - **`Label`**: label to be used in the **`Fields`** panel
   - **`Type`**: data type of the field

   b. Click **`Create`**.



IMPORTANT

To use a data object, make sure you import the data model into your resource. This is necessary even if the data model lives in the same Project as your resource (Business Process).

5.3. ANNOTATIONS IN DATA MODELER

The Data Modeler in Business Central supports the editing of pre-defined annotations of fact model classes and attributes. For the fact model, the annotations supported are: TypeSafe, Role, Timestamp, Duration and Expires. For the fields within the fact model, the position annotation is supported.

If you want to add/edit custom or pre-defined annotations, you can switch to the source tab and modify the source code directly (in JBDS or Business Central).

CHAPTER 6. WRITING RULES

6.1. CREATING A RULE

Procedure 6.1. Creating a new rule

1. In the **`Project Explorer`** view, do the following:
   - If in the Project view of Project Explorer, select the organizational unit, repository and the project where you want to create the rule.
   - If in the Repository view of Project Explorer, navigate to src/main/resources/ and the SUBFOLDER/PACKAGE where you want to create the project folder for the rule template.
2. In the perspective menu, go to **New Item** → **Guided Rule**.
3. In the **`Create new Guided Rule`** dialog window, define the package details:
   a. In the **`Resource Name`** text box, enter the Guided Rule name and click **`OK`**.
4. The new Guided Rule is now created under the selected project.

6.2. THE ASSET EDITOR

6.2.1. The asset editor

The asset editor provides access to information about assets and gives users the ability to edit assets.

The editor contains Edit, Source, Config and metadata tabs.

Edit tab

The edit tab is where assets can be edited. The available options in the edit tab will depend on the type of asset being edited.

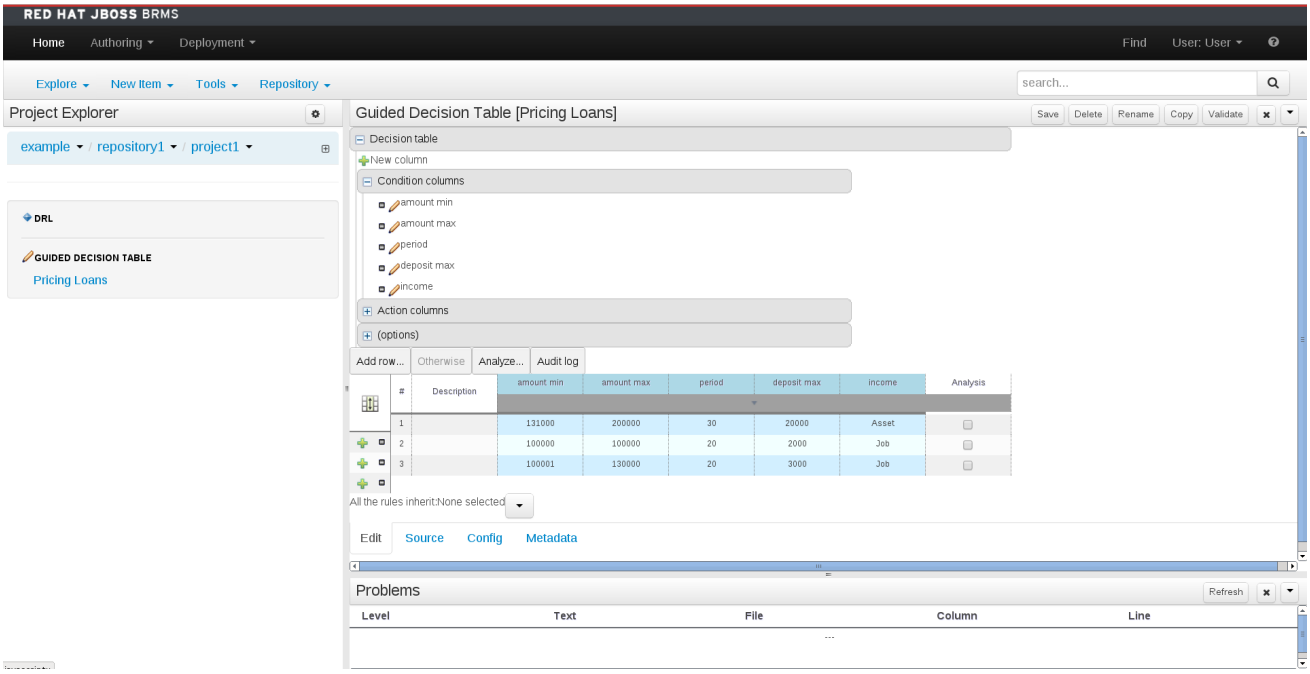


Figure 6.1. The asset editor - Edit tab

Source tab

Source tab shows the DRL source for a selected asset.

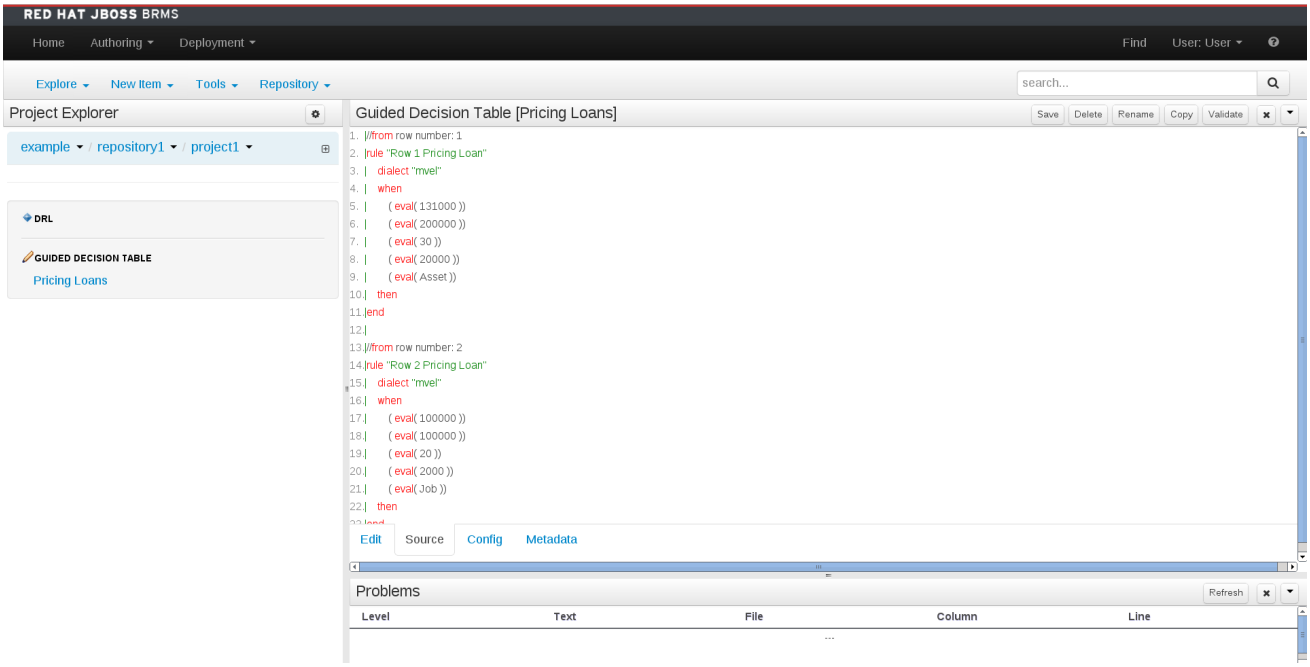


Figure 6.2. The asset editor - Source tab

Config tab

The config tab suggests the set of imports used in the project. Each asset has its own imports and suggest fact types that the user might want to use.

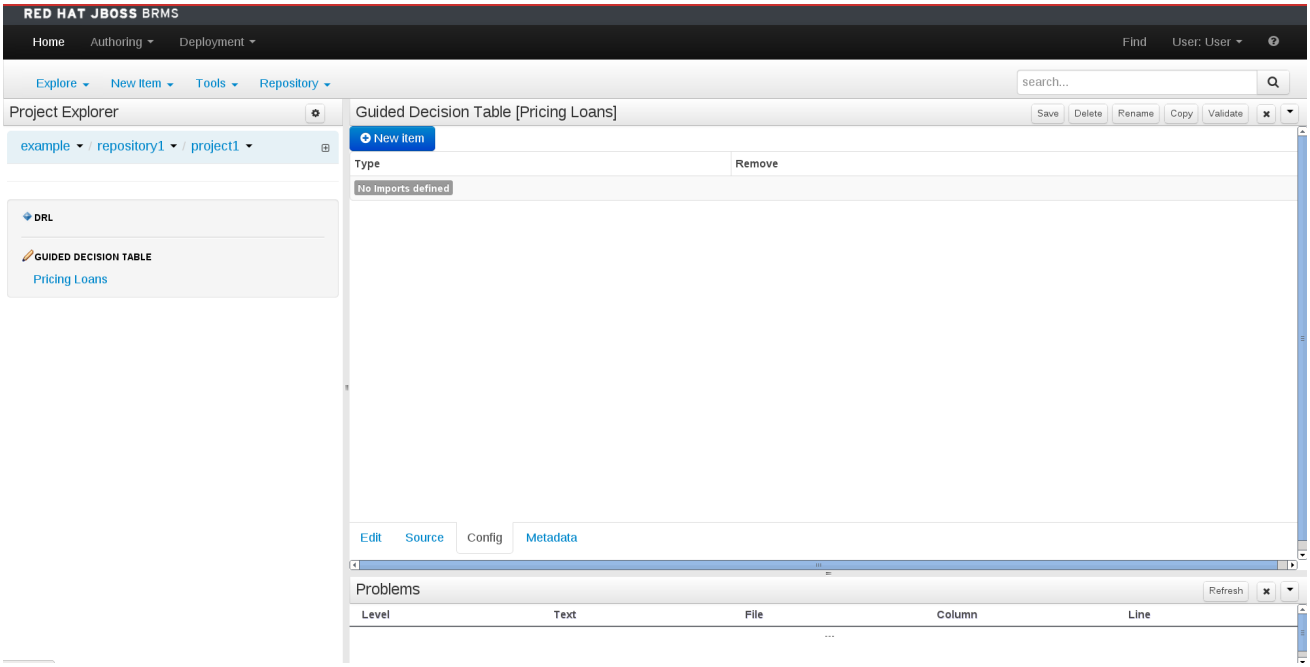


Figure 6.3. The asset editor - Config tab

Metadata tab

The Metadata screen displays the generic data and version history of an asset. It allows a user to edit other metadata details, add descriptions and discussions which are specific to a selected asset.

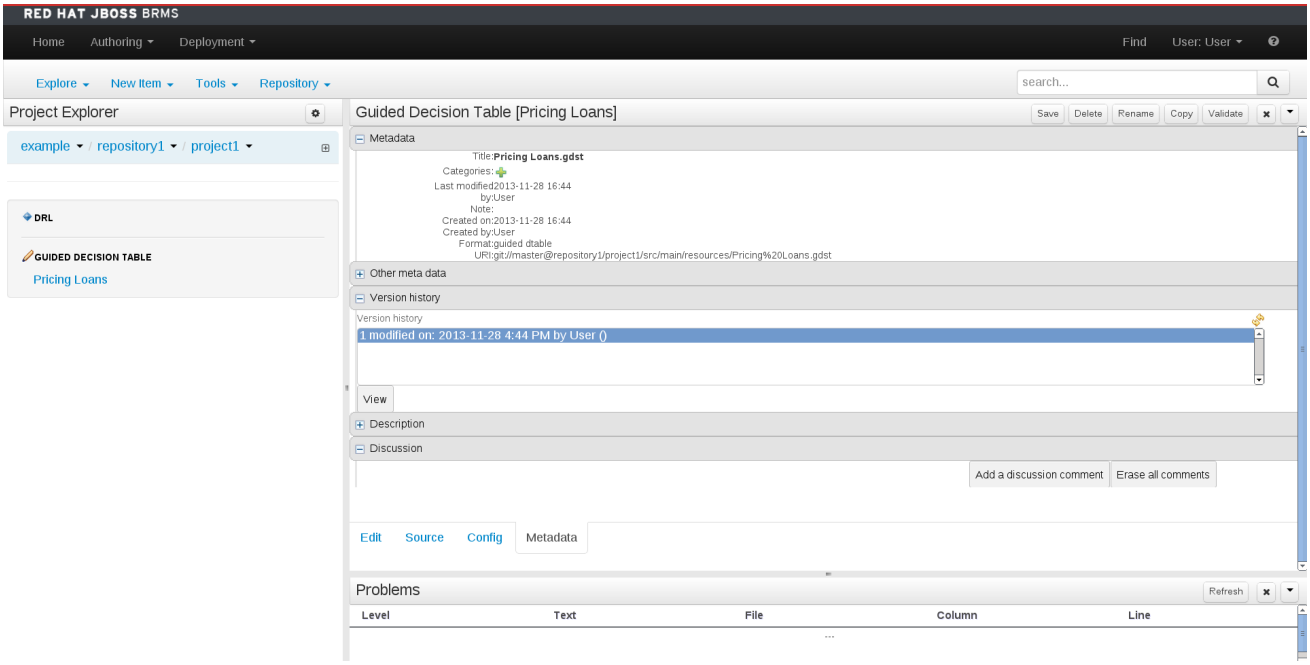


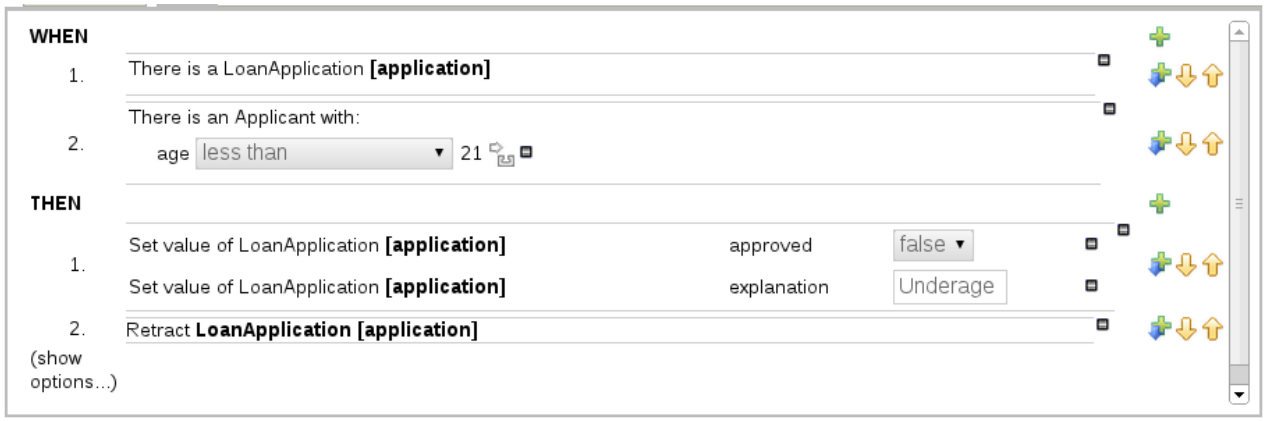
Figure 6.4. The asset editor - Metadata tab

6.2.2. Business rules with the guided editor

Business rules are edited in the guided editor. Rules edited in the guided editor use the Business Rules Language (BRL) format. The guided editor prompts users for input based on the object model of the rule being edited.

A package must exist for assets to be added to before rules can be created. Package access must be configured before users can use the BRL guided editor.

Example 6.1. The guided editor



6.2.3. Narrowing Facts Using Package White List

Starting with 6.1, the facts available during rule creation and modification can be narrowed down using a file called the **`package-names-white-list`**.

The use of this file allows a developer to narrow down the group of facts that are loaded and are therefore, visible. This helps in speeding up the loading of these facts while creating new rules.

This file is created automatically on the creation of a new project in the root directory, along with the **`pom.xml`** and **`project.imports`** project files. For existing projects, you may create this file manually.

In Business Central, you can view this file by opening up the repository view of a project in the Project Explorer. As expected, the default file that is automatically created is empty. An empty file doesn't restrict any facts.

Rules for Defining Packages

The **`package-names-white-list`** file is a text file that accepts single package names on each line. Packages can contain wildcards as defined below:

com.redhat.finance: allows facts from *only* the com.redhat.finance package. Thus, **`com.redhat.finance.Person`** and **`com.redhat.finance.Salary`** are allowed, but **`com.redhat.finance.senior.Management`** are not allowed.
com.redhat.finance.*: allows facts from the sub-packages of the com.redhat.finance package *only*. Thus, **`com.redhat.finance.senior.Management`** and **`com.redhat.finance.junior.Management`** are allowed, but *not* **`com.redhat.finance.Person`**.
com.redhat.finance.**: this is a combination of the above two rules. Allows **`com.redhat.finance.Person`** and **`com.redhat.finance.senior.Management`** and even, **`com.redhat.finance.really.senior.Management`** classes.

6.2.4. The Anatomy of a Rule

A rule consists of multiple parts:

- When

The *When* part of the rule is the condition that must be met. For instance, a bank providing credit in the form of a loan may specify that customers must be over twenty-one years of age. This would be represented by using *when* to determine if the customer is over twenty-one years of age.

- Then

  The *Then* part of the rule is the action to be performed when the conditional part of the rule has been met. For instance, *when* the customer is under twenty-one years of age, *then* decline the loan because the applicant is under age.

- Optional

  Optional attributes such as salience can be defined on rules.

With the guided editor, it is possible to add more conditions to the When (or conditional) part of the rule and more actions to the Then (or action) part of the rule. For instance, if an applicant under the age of 21 had a guarantor for a loan application, the bank may decide to approve the loan application.

6.2.5. Salience

Each rule has a salience value which is an integer value that defaults to zero. The salience value represents the priority of the rule with higher salience values representing higher priority. Salience values can be positive or negative.

6.2.6. Adding Conditions or Actions to Rules

Procedure 6.2. Adding Conditions or Actions to Rules

1. Click the plus icon in the When section of the guided editor to add a condition, or click the plus icon in the Then section of the guided editor to add an action.

2. Select the condition or action from the menu and click **Ok**. If the package the rule belongs to has been configured to include DSL (Domain Specific Language) sentences, DSL sentences can be chosen from the menu.

3. If the condition or action requires input, i.e., a date, true or false, am integer, or other input type, enter the required value.

6.2.7. Adding a Field to a Fact Type

With the guided editor, it is possible to add more conditions to the 'when' (or conditional) part of the rule and more actions to the 'then' (or action) part of the rule. For instance, if a loan applicant under the age of 21 had a guarantor for a loan application, the bank may decide to approve the loan application.

To add the guarantor to the condition, it is first necessary to add the **`guarantor`** field to the application fact type for the mortgage model.

Procedure 6.3. Adding a Field to a Fact Type

1. **Select the Model**
   From the Project Explorer, select the Project and expand the package that contains the model.

   Select and Open the model from the list by clicking over it.

2. **Add the Field**
   Expand the fact type by clicking the plus sign next to it and select **Add Field**.

3. **Enter the Field Details**
   Add the details to the pop up dialogue. In this case, enter the name *guarantor* in the **Field name** field and select **True or False** from the **Type** drop down menu.

   Save the changes made to the model by selecting **File** and **Save changes**.

With the guarantor field now added to the applicant fact type, it is possible to modify the rule to include a guarantor.

6.2.8. Technical Rules (DRL)

Technical (DRL) rules are stored as text and can be managed in the Red Hat JBoss BRMS user interface. A DRL file can contain one or more rules. If the file contains only a single rule, then the package, imports and rule statements are not required. The condition and the action of the rule can be marked with "when" and "then" respectively.

Red Hat JBoss Developer Studio provides tools for creating, editing, and debugging DRL files, and it should be used for these purposes. However, DRL rules can be managed within the JBoss BRMS user interface.

```
salience 100 #this can short circuit any processing
when
      a : Approve()
      p : Policy()
then
      p.setApproved(true);
      System.out.println("APPROVED: "  + a.getReason());
```

Figure 6.5. Technical Rule (DRL)

6.3. DECISION TABLES

6.3.1. Spreadsheet Decision Tables

Rules can be stored in spreadsheet decision tables. Each row in the spreadsheet is a rule, and each column is either a condition, an action, or an option. The *Red Hat JBoss BRMS Development Guide* provides details for using decision tables.

6.3.2. Uploading Spreadsheet Decision Tables

Procedure 6.4. Uploading a Spreadsheet Decision Table

1. To upload an existing spreadsheet, select **New Item** → **Decision Table (Spreadsheet)**.

2. Enter a name for the spreadsheet and then click **Choose file...** button to browse and select the spreadsheet. You can only select **.xls** files. Click the **Ok** button when done.

To convert the uploaded spreadsheet to a Guided Decision table:

1. Validate the uploaded spreadsheet by clicking on the **`Validate`** button located on the project screen menu bar.
2. Now click on the **`Other`** drop down menu on the project screen toolbar and select the **`Convert to Guided Decision Table`** option.

6.3.3. Spreadsheet Decision Table Examples

We are here considering a simple example for an online shopping site which lists out the shipping charges for the ordered items. The site agrees for a FREE shipping with the following conditions:

- If the number of items ordered is 4 or more and totaling $300 or over and
- If delivered at the standard shipping day from the day they were purchased which would be 4 to 5 working days.

The listed shipping rates are as follows:

Table 6.1. For orders less than $300

Number of items	Delivery Day	Shipping Charge, N = Number of Items
3 or less	Next Day	$35
	2nd Day	$15
	Standard	$10
4 or more	Next Day	N*$7.50
	2nd Day	N*3.50
	Standard	N*2.50

Table 6.2. For orders more than $300

Number of items	Delivery Day	Shipping Charge, N = Number of Items
3 or less	Next Day	$25
	2nd Day	$10
	Standard	N*$1.50
4 or more	Next Day	N*$5
	2nd Day	N*2

Number of items	Delivery Day	Shipping Charge, N = Number of Items
	Standard	FREE

The above conditions can be presented in a spreadsheet as:

Calculating Shipping Charges												
Purchase Amount	**Over $300**						**Less than $300**					
Number of Items	**3 or less**			**4 or more**			**3 or less**			**4 or more**		
Delivery Day	Next	2nd day	Standard	Next	2nd day	Standard	Next	2nd day	Standard	Next	2nd day	Standard
Shipping Charges ($)	25	10	N * 1.50	N * 5	N * 2	*FREE*	35	15	10	N * 7.50	N * 3.50	N * 2.50

Decision Tables can also be mapped based on actions and conditions as shown in the following format:

Conditions	Condition Alternatives
Actions	Action Entries

To explain the above format in detail we use a simple example of sending an email to a recipient with following conditions:

- Send an email when Recipient address is present, subject is present and before 5:30pm
- If after 5:30pm, then put it in the pending folder
- If Recipient address is missing, give a warning message
- If all fields are present and before 5:30, send the email

The Decision table can be mapped as follows:

Table 6.3. Decision Table Mapping

Criteria		Mapped Entries			
`Conditions (IF)`	Address Present	Y	Y		Y
	Subject Present	Y		Y	Y
	Before 5:30	Y	Y	Y	
`Actions (THEN)`	Send Mail	X			
	Error Message		X	X	
	Make Pending				X

The above example is a limited-entry decision table where the condition alternatives are simple boolean values and the action entries are marked to reprsent which actions in the given columns are to be

performed. For example if the condition alternatives are mapped for all 3 conditions - Address present, Subject Present and Before 5:30, the action entry is marked for action as Send Mail. So if all the three conditions are satisfied, the action will be to send the mail.

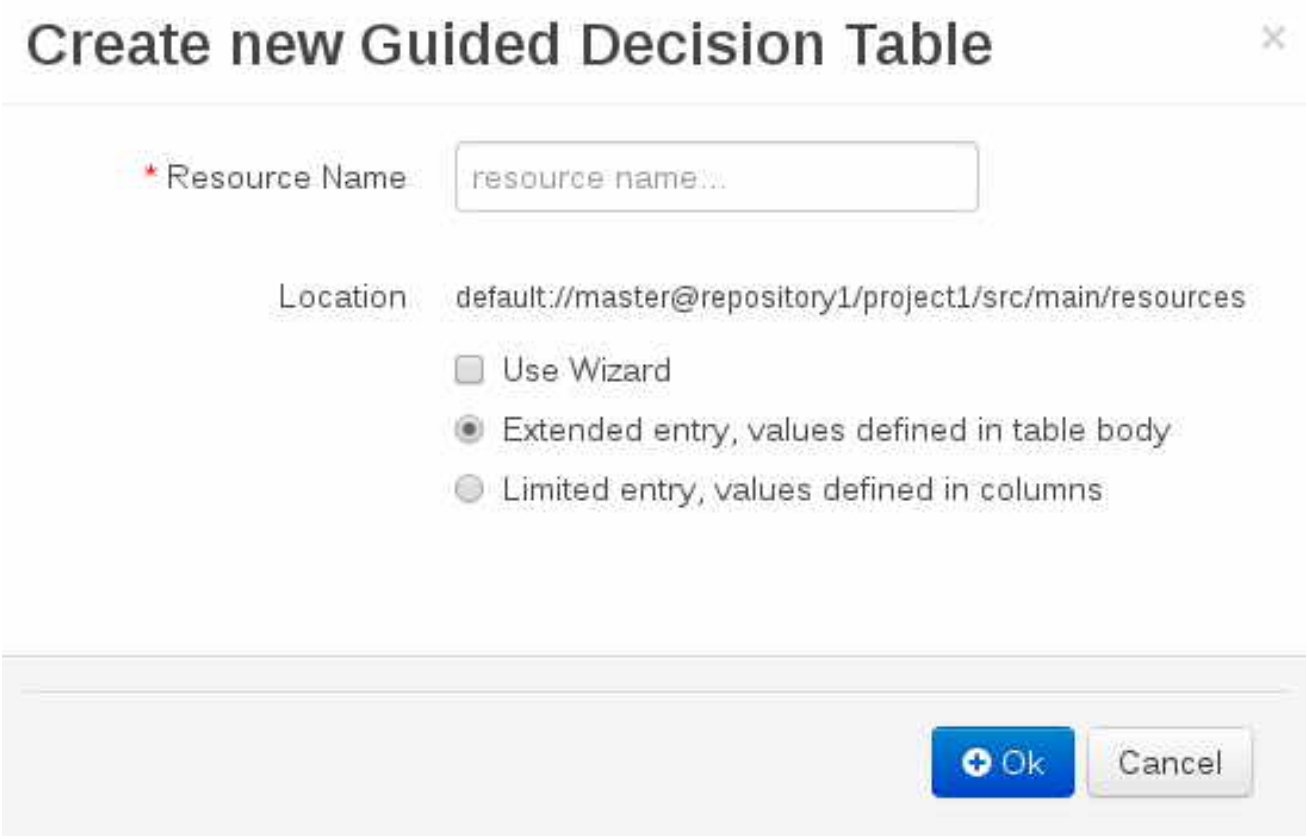
6.4. WEB BASED GUIDED DECISION TABLES

6.4.1. Web Based Guided Decision Tables

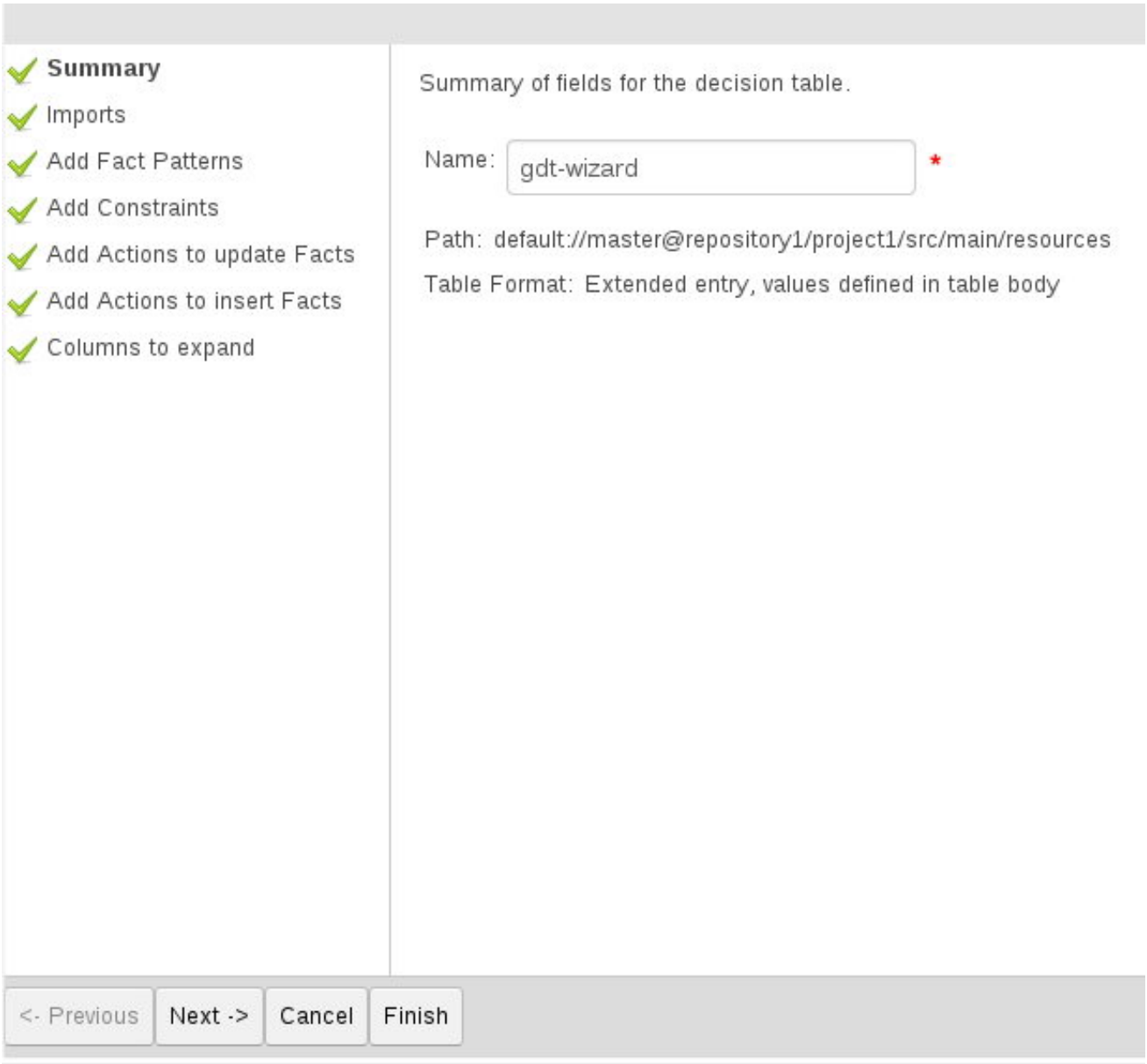
The (web based) Guided Decision Table feature works similar to the Guided Editor by introspecting what facts and fields are available to guide the creation of a decision table.

Rule attributes, meta-data, conditions and actions can be defined in a tabular format thus facilitating rapid entry of large sets of related rules. Web based decision table rules are compiled into DRL like all other rule assets.

To create a new decision table, click on **New Item** → **Guided Decision Table**. Enter the name of the table and select whether you want the extended entry or limited entry table (Section 6.4.2, “Types of decision tables”). Optionally select to use the Guided Decision Table Wizard.



Click **OK** when done. If you didn't select the wizard, you will be presented with the editor for Guided Decision Tables. If you selected the wizard, you will be presented with the first screen of the wizard.



The wizard helps you define your imports, facts, patterns and columns, but not the rows. Rows are added in the Guided Decision Table Editor, which is what you are presented with at the end of the wizard (or directly if you didn't use the wizard).

A brand new empty decision table

6.4.2. Types of decision tables

There are broadly two types of decision tables, both of which are supported:

- Extended Entry
- Limited Entry

Extended entry

An Extended Entry decision table is one for which the column definitions specify Pattern, Field and operator but not value. The values, or states, are themselves held in the body of the decision table. It is normal, but not essential, for the range of possible values to be restricted by limiting entry to values from a list. Business central supports use of Java enumerations or decision table "optional value lists" to restrict value entry.

Limited entry

A Limited Entry decision table is one for which the column definitions specify value in addition to Pattern, Field and operator. The decision table states, held in the body of the table, are boolean where a positive value (a checked tick-box) has the effect of meaning the column should apply, or be matched. A negative value (a cleared tick-box) means the column does not apply.

6.4.3. Column Configuration

Columns can have the following types of constraint:

- Literal

  The value in the cell will be compared with the field using the operator.

- Formula

  The expression in the cell will be evaluated and then compared with the field.

- Predicate

  No field is needed, the expression will be evaluated to true or false.

You can set a default value, but normally if there is no value in the cell, that constraint will not apply.

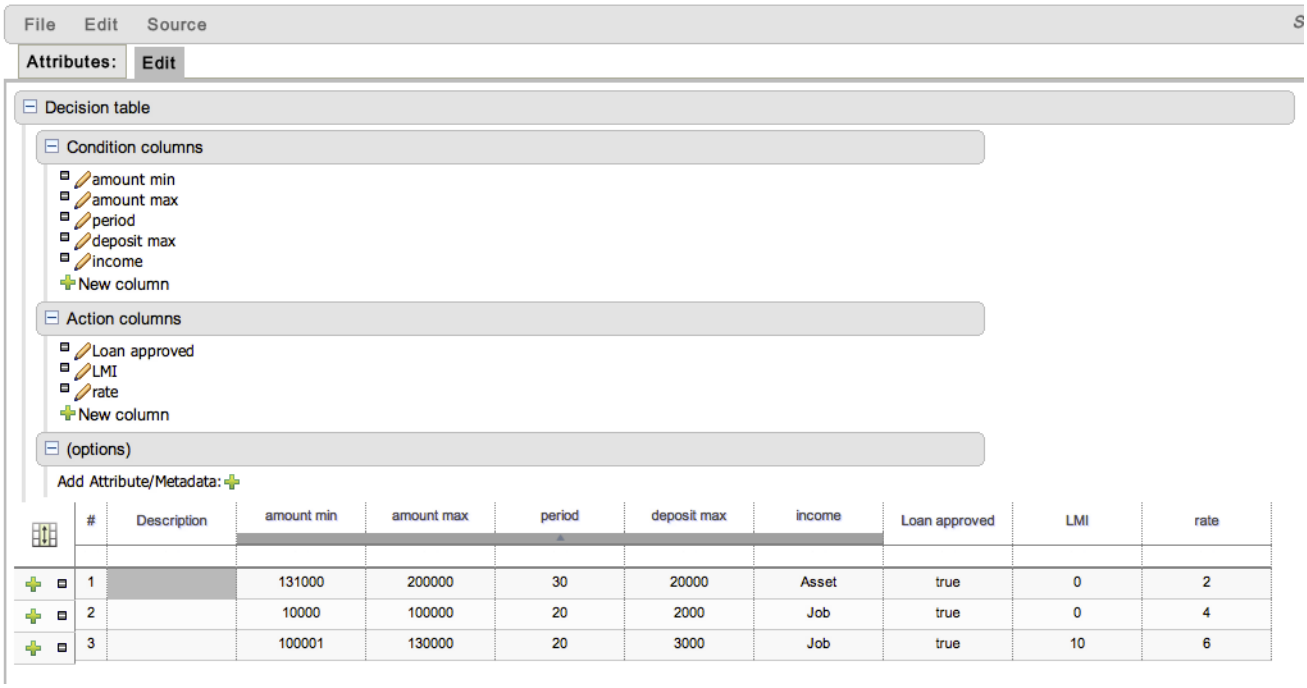
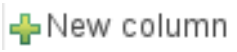


Figure 6.6. Column Configuration

6.4.4. Adding Columns

To add a column within the Guided Decision Table Editor, click on the  icon.

The following column type selection dialog appears:

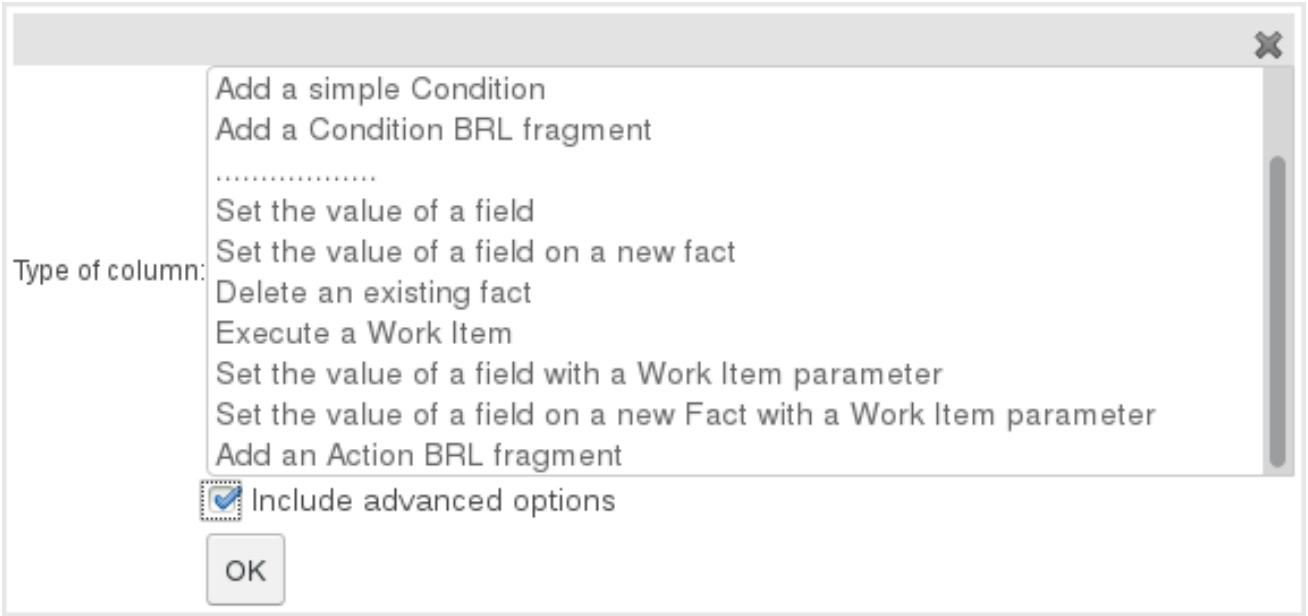


Figure 6.7. Advanced Column Options

By default, the column type dialog shows the following types:

- Add a new Metadata\Attribute column
- Add a simple Condition
- Set the value of a field

- Set the value of a field on a new fact
- Delete an existing fact

Clicking on "Include advanced options" adds the following options:

- Add a Condition BRL fragment
- Execute a Work Item
- Set the value of a field with a Work Item parameter
- Set the value of a field on a new Fact with a Work Item parameter
- Add an Action BRL fragment

6.4.5. Column Types

6.4.5.1. Attribute Columns

Zero or more attribute columns representing any of the DRL rule attributes can be added. An additional pseudo attribute is provided in the guided decision table editor to "negate" a rule. Use of this attribute allows complete rules to be negated. For example, the following simple rule can be negated as also shown.

```
when
  $c : Cheese( name == "Cheddar" )
then
  ...
end
```

```
when
  not Cheese( name == "Cheddar" )
then
  ...
end
```

6.4.5.2. Metadata Columns

Zero or more meta-data columns can be defined, each represents the normal meta-data annotation on DRL rules.

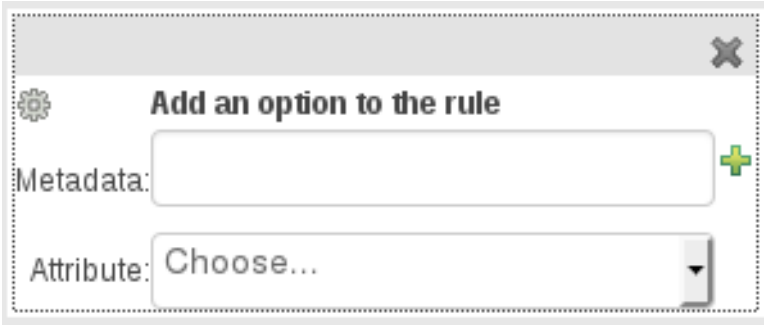


Figure 6.8. Attribute and MetaData Option

6.4.5.3. Condition Columns

Conditions represent fact patterns defined in the right-hand side, or "when" portion, of a rule. To define a condition column, you must define a binding to a model class or select one that has previously been defined. You can choose to negate the pattern. Once this has been completed, you can define field constraints. If two or more columns are defined using the same fact pattern binding, the field constraints become composite field constraints on the same pattern. If you define multiple bindings for a single model class, each binding becomes a separate model class in the right-hand side of the rule.

When you edit or create a new column, you will be given a choice of the type of constraint:

- Literal : The value in the cell will be compared with the field using the operator.
- Formula: The expression in the cell will be evaluated and then compared with the field.
- Predicate : No field is needed, the expression will be evaluated to true or false.

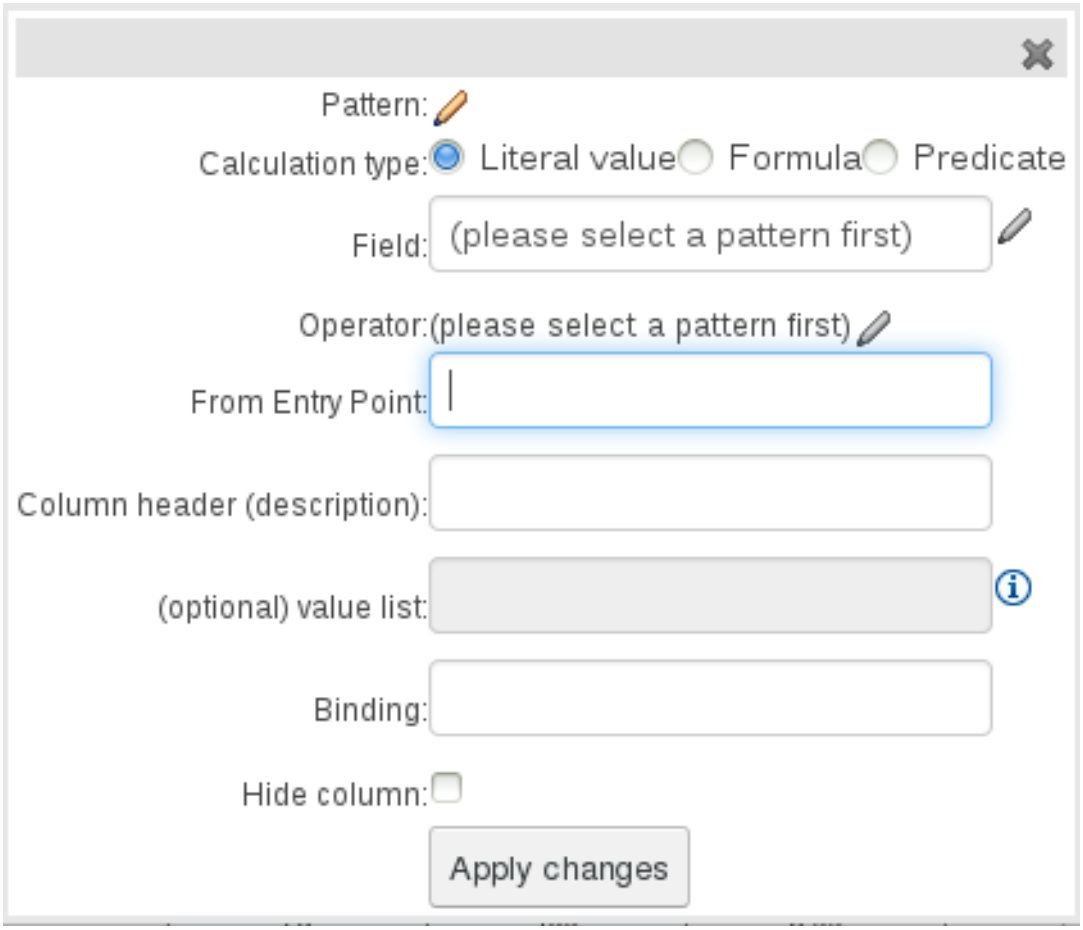


Figure 6.9. Simple Condition Column

6.4.5.4. Field Value Columns

Creates an Action to set the value of a field on a previously bound fact. You have the option to notify the Rule Engine of the modified values which could lead to other rules being re-activated.

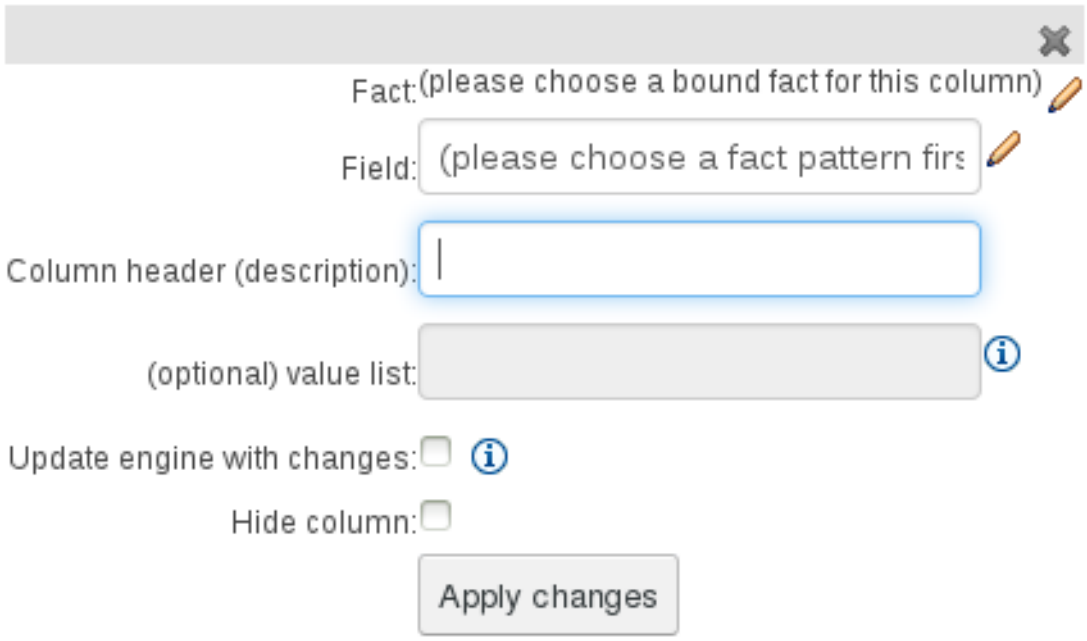


Figure 6.10. Set the value of a field

6.4.5.5. New Fact Field Value Columns

This column allows an Action to insert a new Fact into the Rule Engine Working Memory, and it sets the value of one of the new Facts' fields. You can choose to have the new Fact "logically inserted;" that is, it will automatically be deleted should the conditions leading to the action being invoked cease to be true. Please refer to the Red Hat JBoss Development Guide for information about truth maintenance and logical insertions.

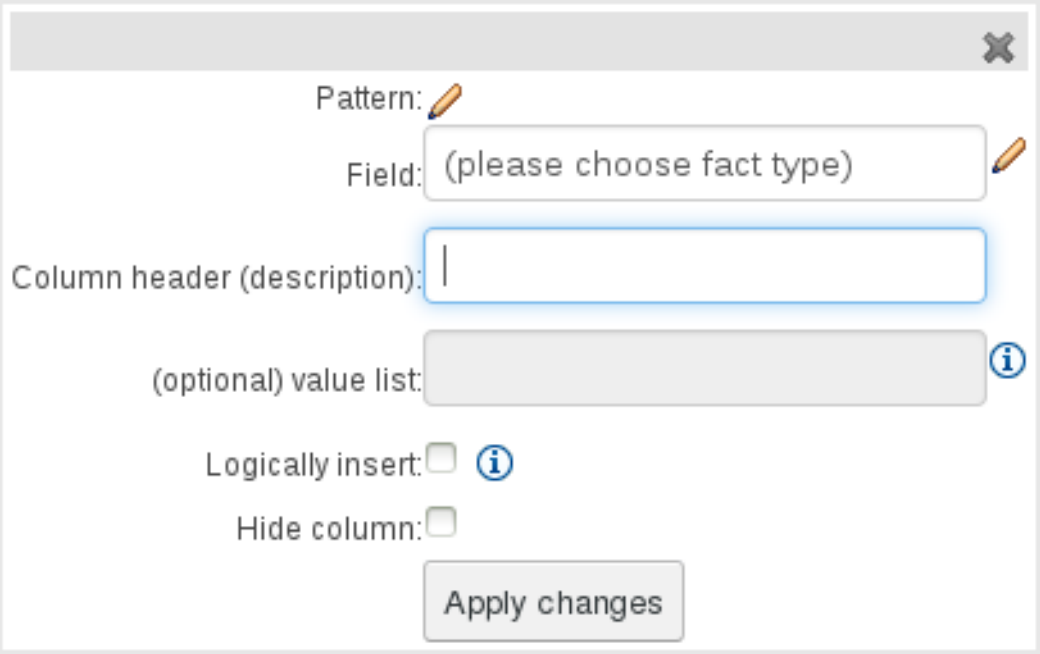


Figure 6.11. Set the value of a field on a new fact

6.4.5.6. Delete Existing Fact Columns

The implementation of an Action to delete a bound Fact.

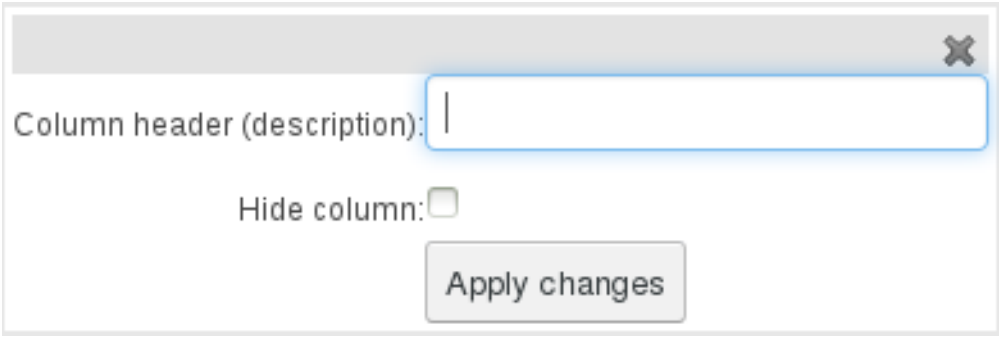


Figure 6.12. Delete an existing fact

6.4.6. Advanced Column Types

6.4.6.1. Condition BRL Fragment Columns

A construct that allows a BRL fragment to be used in the left-hand side of a rule. A BRL fragment is authored using the Guided Rule Editor and hence all features available in that editor can be used to define a decision table column such as the following: "from", "collect", and "accumulate". When using the embedded Guided Rule Editor, field values defined as "Template Keys" will form columns in the decision table. Facts and Fact's fields bound in the BRL fragment can be referenced by the simpler column types and vice-versa.

The following example displays a BRL Condition for a shopping tier.

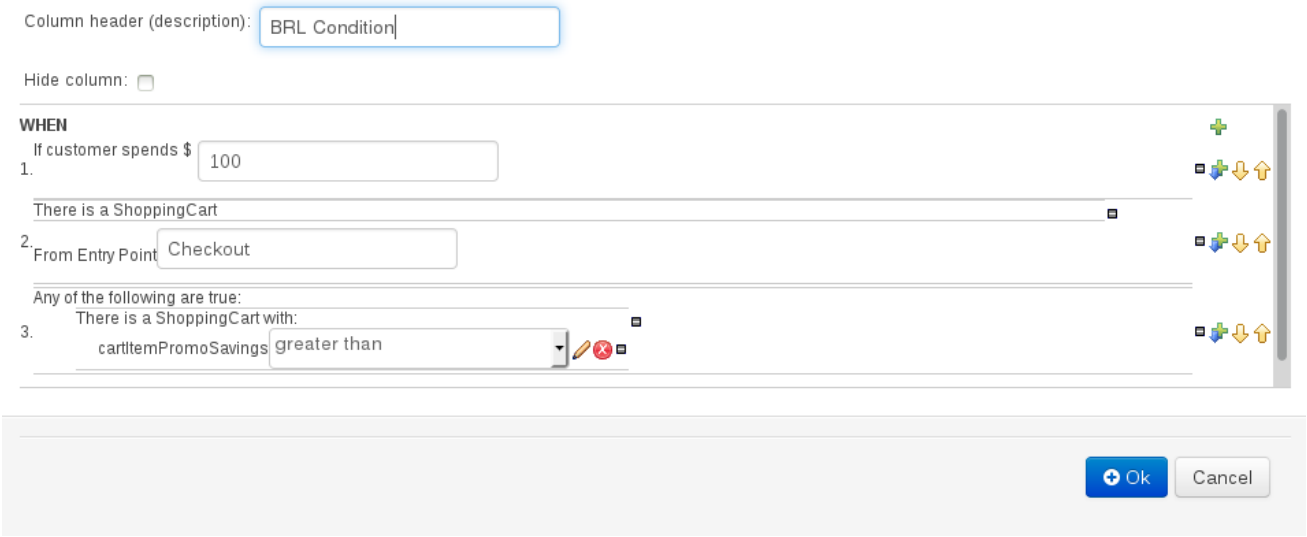


Figure 6.13. Add a Condition BRL fragment

6.4.6.2. Execute Work Item Columns

This implements an Action to invoke a Red Hat JBoss Business Process Management Suite Work Item Handler. It sets its input parameters to bound Facts/Facts' fields values. This works for any Work Item Definition.

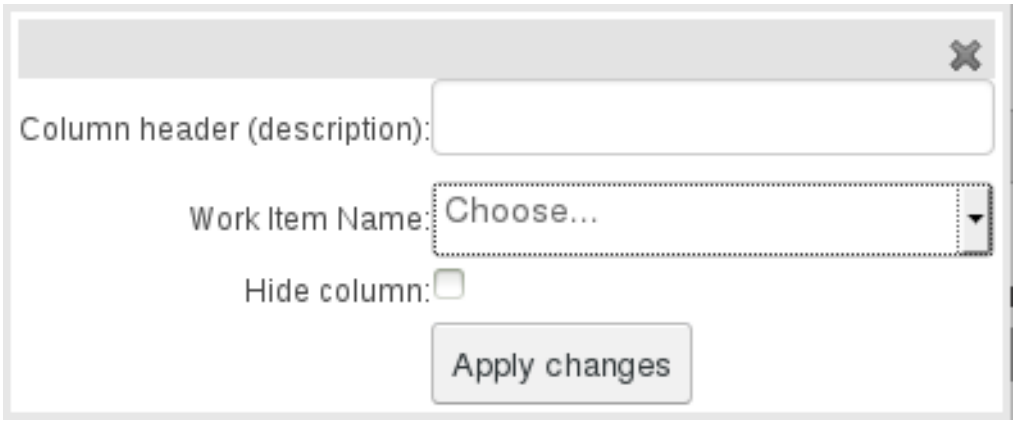


Figure 6.14. Execute a Work Item

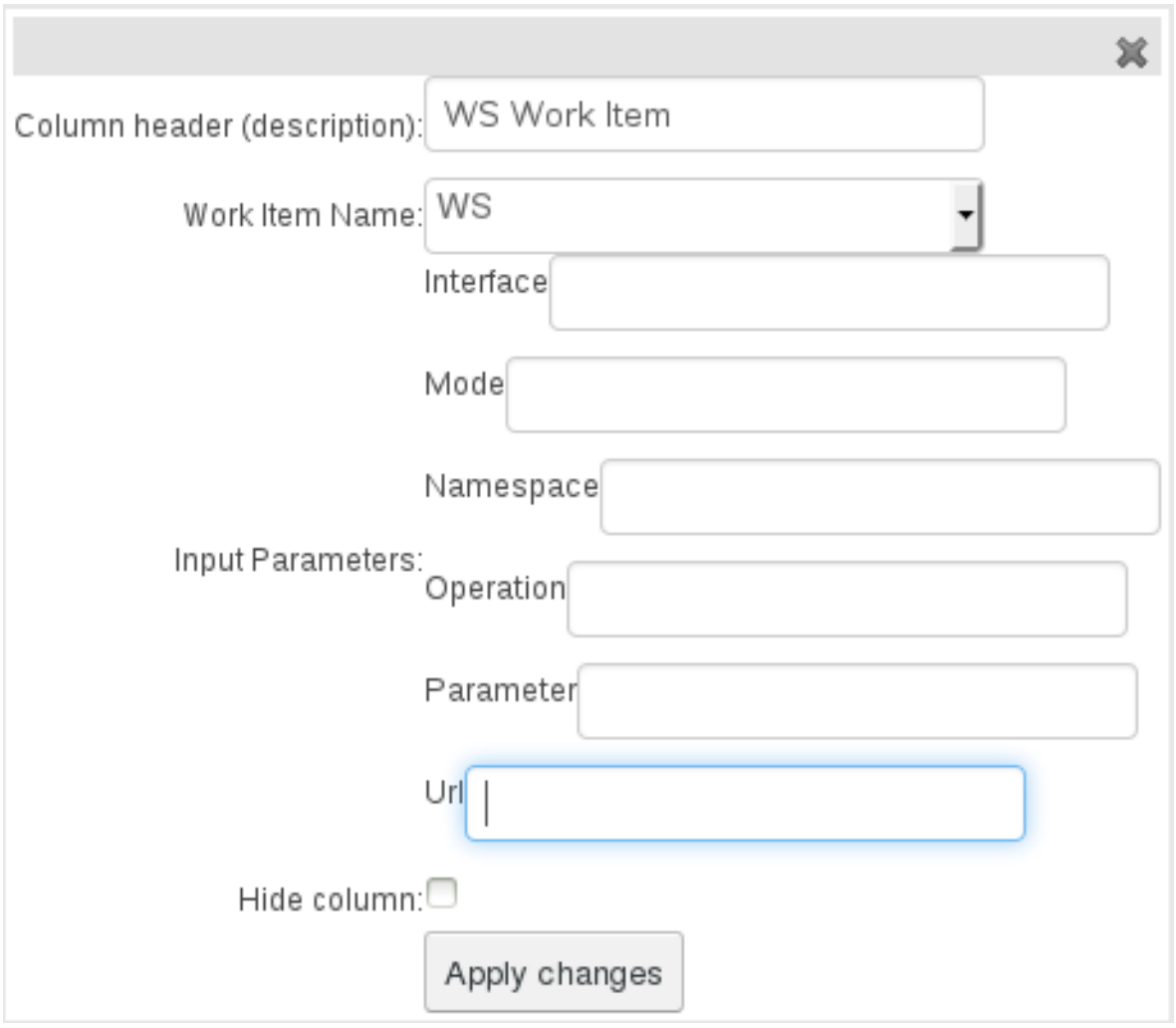


Figure 6.15. WS Work Item

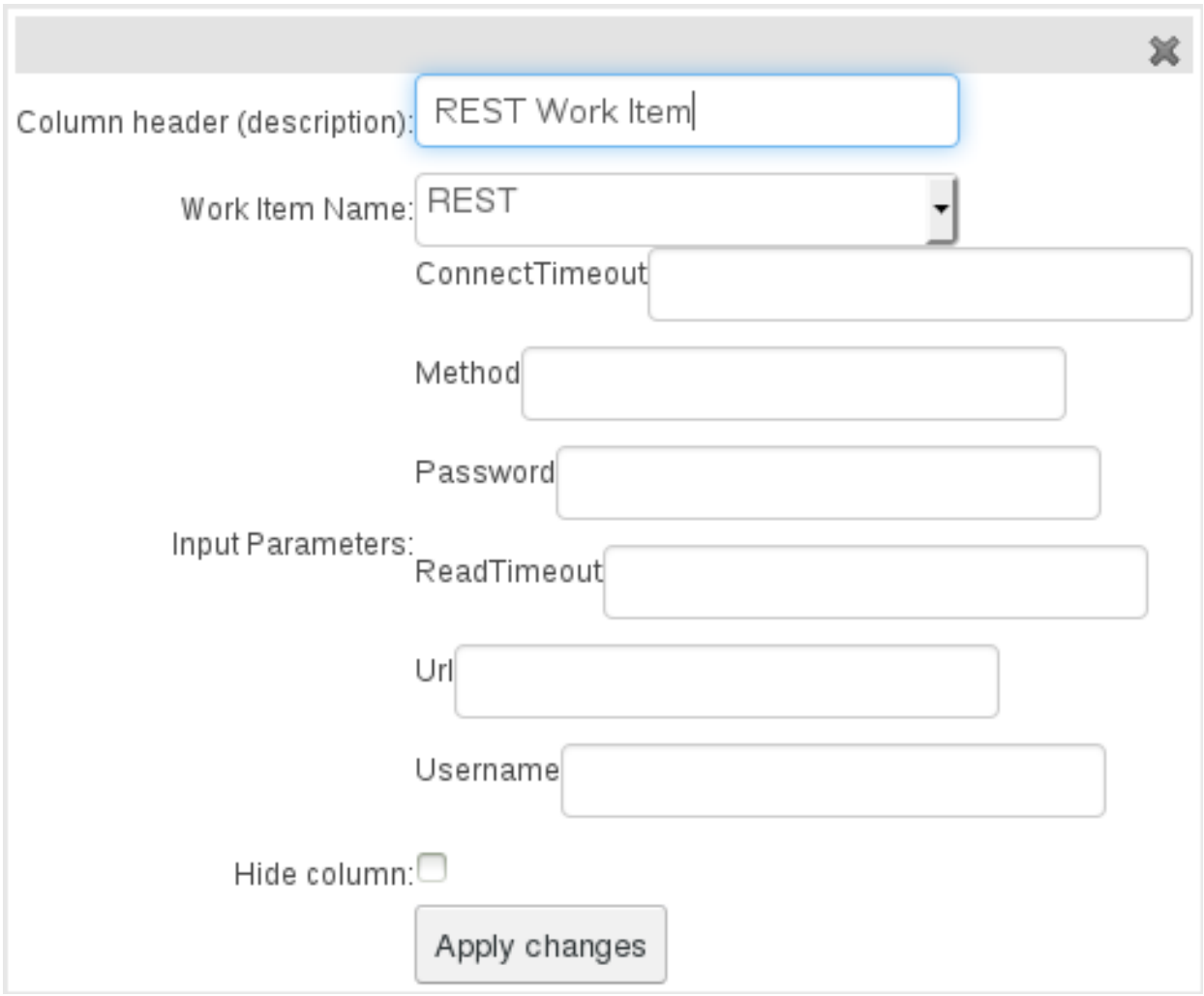


Figure 6.16. REST Work Item

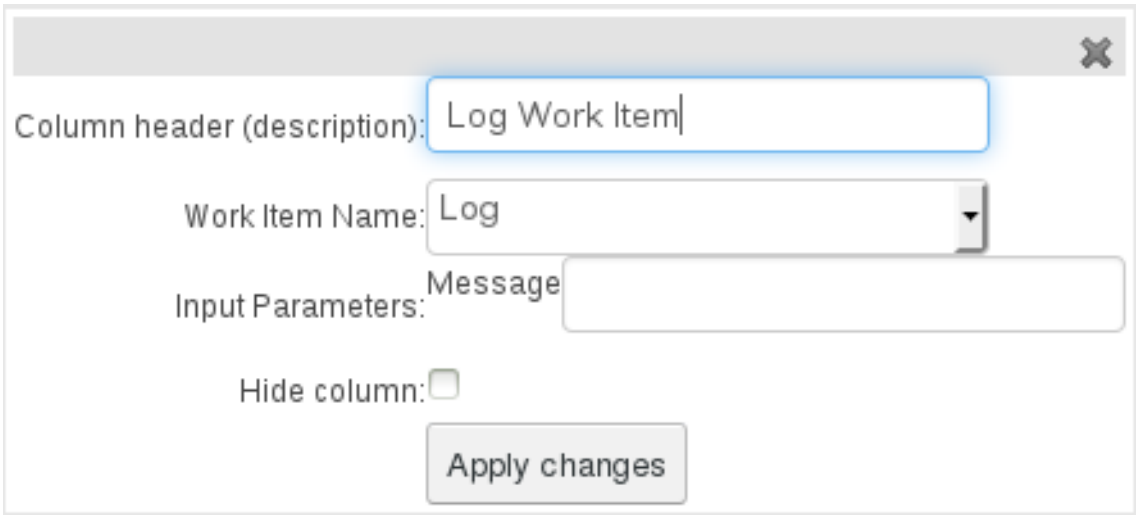


Figure 6.17. Log Work Item

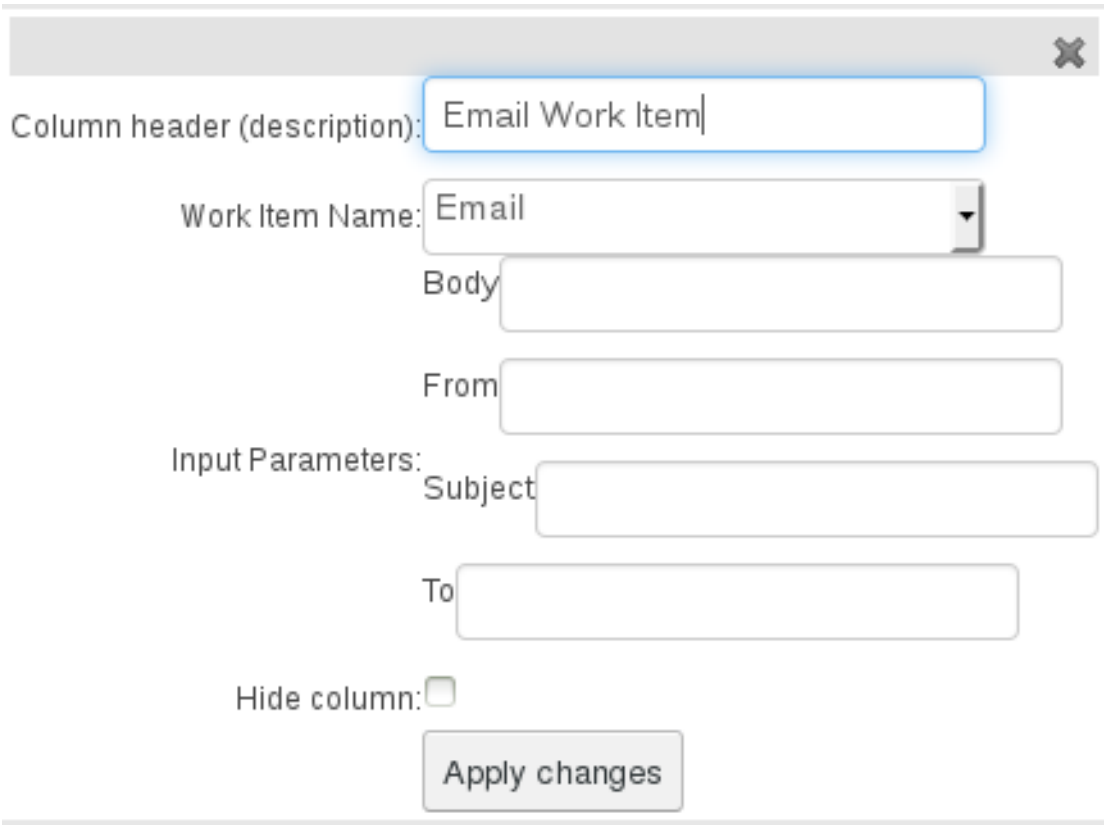


Figure 6.18. Email Work Item

6.4.6.3. Field Value with Work Item Parameter Columns

This implements an Action to set the value of a Fact's field to that of a Red Hat JBoss Business Process Management Suite Work Item Handler's result parameter. The Work Item needs to define a result parameter of the same data-type as a field on a bound fact; this will allow you to set the field to the return parameter.

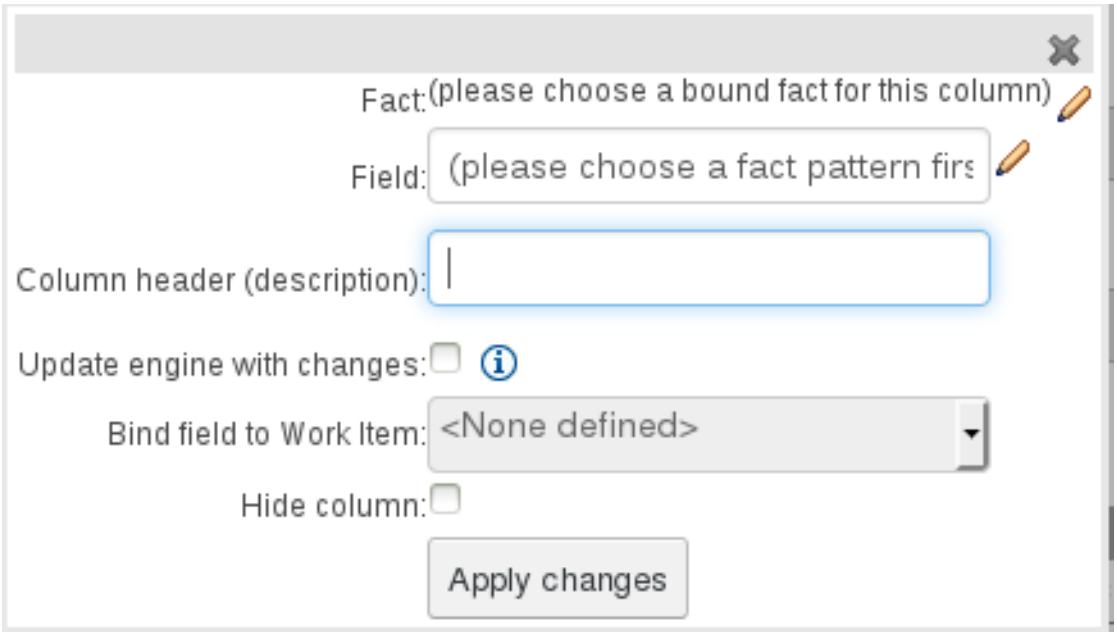
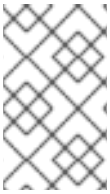


Figure 6.19. Set the value of a field with a Work Item parameter



NOTE

Please note that in order to set the "Bind field to Work Item" option, you first need to have an Action executing a Work Item. This will allow you to set the field of an existing Fact based on a Work Item's results.

6.4.6.4. New Fact Field Value with Work Item Parameter Columns

This implements an Action to set the value of a new Fact's field to that of a Red Hat JBoss Business Process Management Suite Work Item Handler's result parameter. Again, this Work Item needs to define a result parameter of the same data-type as a field on a bound fact type. You should then be able to set the field to the return parameter.

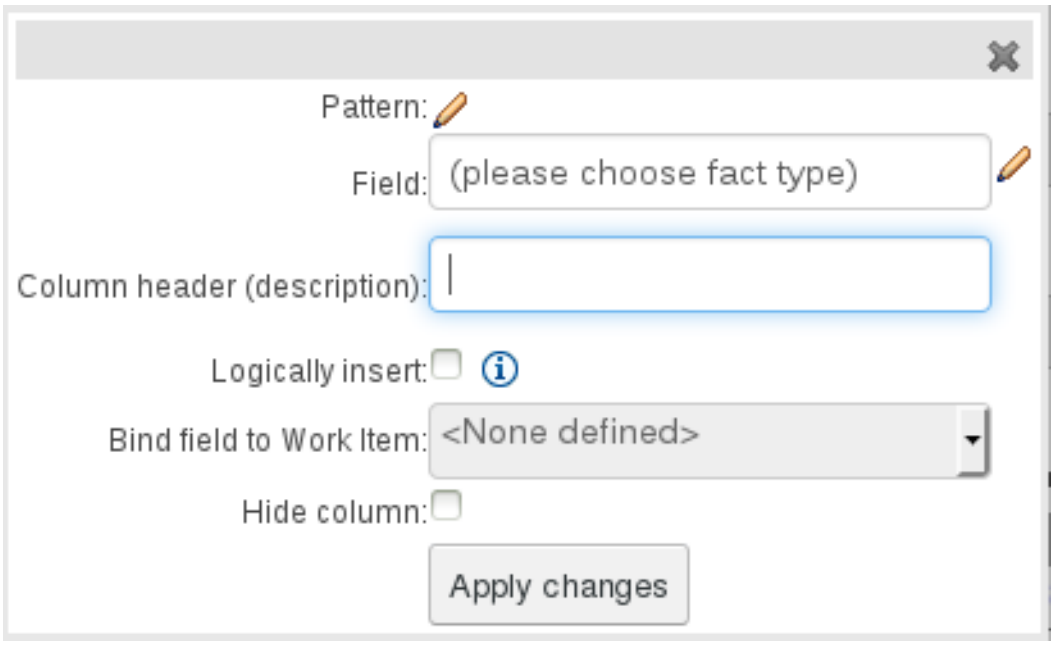
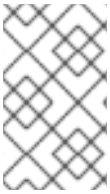


Figure 6.20. Set the value of a field on a new Fact with a Work Item parameter.



NOTE

Please note that in order to set the "Bind field to Work Item" option, you first need to have an Action executing a Work Item. This will allow you to insert a new Fact with a field value from a Work Item's Results.

6.4.6.5. Action BRL Fragment Columns

A construct that allows a BRL fragment to be used in the right-hand side of a rule. A BRL fragment is authored using the Guided Rule Editor and hence all features available in that editor can be used to define a decision table column. When using the embedded Guided Rule Editor, field values defined as "Template Keys" will form columns in the decision table. Facts bound in the BRL fragment can be referenced by the simpler column types and vice-versa.

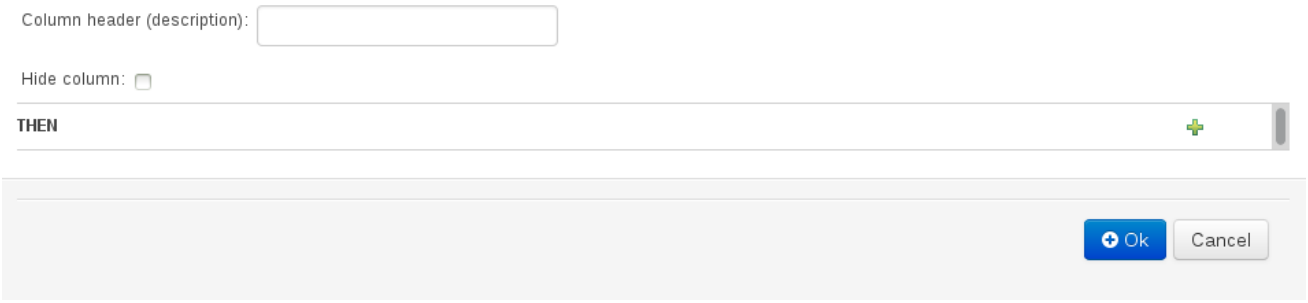


Figure 6.21. Simple layout for Adding an Action BRL fragment

6.4.7. Rule Definition

Rules are created in the main body of the decision table using the columns that have already been defined.

Rows of rules can be added or deleted by clicking the plus or minus symbols respectively.

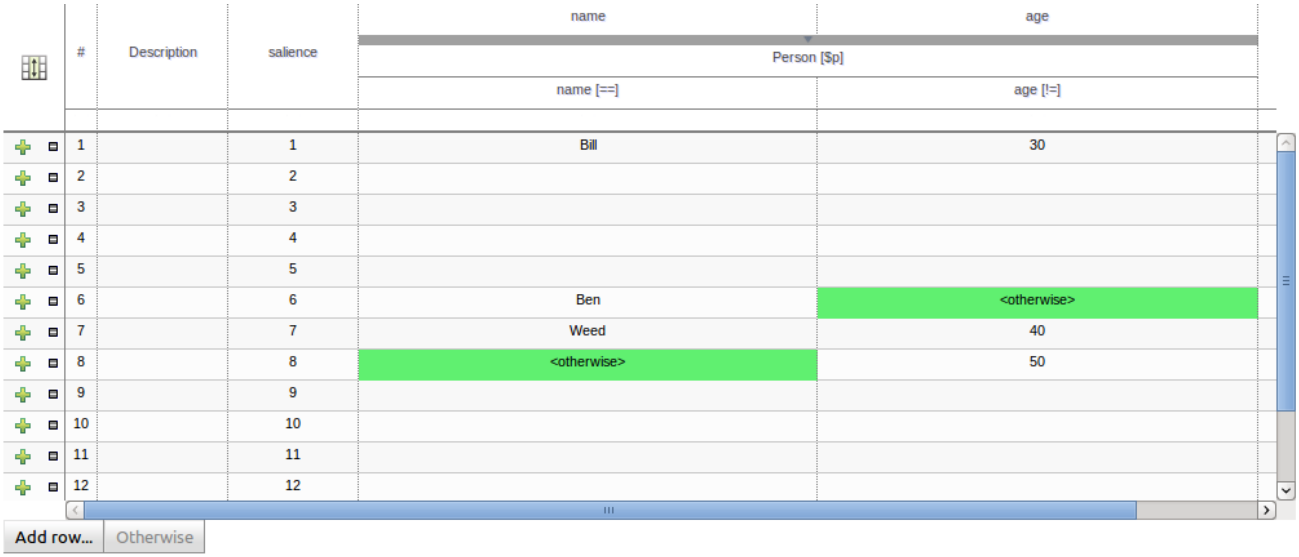


Figure 6.22. Rule Definition

6.4.8. Cell Merging

The icon in the top left of the decision table toggles cell merging on and off. When cells are merged, those in the same column with identical values are merged into a single cell. This simplifies changing the value of multiple cells that shared the same original value. When cells are merged, they also gain an icon in the top-left of the cell that allows rows spanning the merged cell to be grouped.

#	Description	salience	name	age	age
1		1	Bill	30	12345
2		2	Ben	<otherwise>	
3		3			
4		4			
5		5			
6		6	Weed	40	12345
7		7	<otherwise>	50	

Figure 6.23. Cell Merging

6.4.9. Cell Grouping

Cells that have been merged can be further collapsed into a single row. Clicking the [+\-] icon in the top left of a merged cell collapses the corresponding rows into a single entry. Cells in other columns spanning the collapsed rows that have identical values are shown unchanged. Cells in other columns spanning the collapsed rows that have different values are highlighted and the first value displayed.

#	Description	salience	name	age	age
1		1	Bill	30	12345
2		2	Ben	<otherwise>	12345
6		6	Weed	40	12345
7		7	<otherwise>	50	

Figure 6.24. Cell Grouping

When the value of a grouped cell is altered, all cells that have been collapsed also have their values updated.

6.4.10. Otherwise Operations

Condition columns defined with literal values that use either the equality **==** or inequality **!=** operators can take advantage of a special decision table cell value of **otherwise**. This special value allows a rule to be defined that matches on all values not explicitly defined in all other rules defined in the table. This is best illustrated with an example:

```
when
  Cheese( name not in ("Cheddar", "Edam", "Brie") )
  ...
then
  ...
end
```

```
when
```

```
  Cheese( name in ("Cheddar", "Edam", "Brie") )
  ...
then
  ...
end
```

6.5. RULE TEMPLATES

6.5.1. The Guided Rule Template

Rule Templates allow the user to define a rule structure. They provide a place-holder for values and data, and they populate templates to generate many rules. From the user's perspective, Guided Rule Templates are a parametrized guided rule with a data table which provides parameter values. This can allow for more flexible decision tables and it can enhance the flexibility of rules in existing databases. For more information about Rule Templates, see the Red Hat JBoss BRMS Development Guide.

Procedure 6.5. Creating a new Guided Rule Template

1. In the **`Project Explorer`** view, do one of the following:
   - If you are in the Project view, select the organizational unit, repository, and the project where you want to create the template.
   - If you are in the Repository view, navigate to **`src/main/resources/`** and the **`SUBFOLDER/PACKAGE`** where you want to create the project folder for the rule template.
2. In the perspective menu, go to **New Item** → **Guided Rule Template**.
3. In the **`Create new Guided Rule Template`** dialog window, specify the rule template name:
   a. In the **`Resource Name`** text box, enter the Guided Rule Template name and click **`OK`**.
4. The new Guided Rule Template is now created and under the selected project.

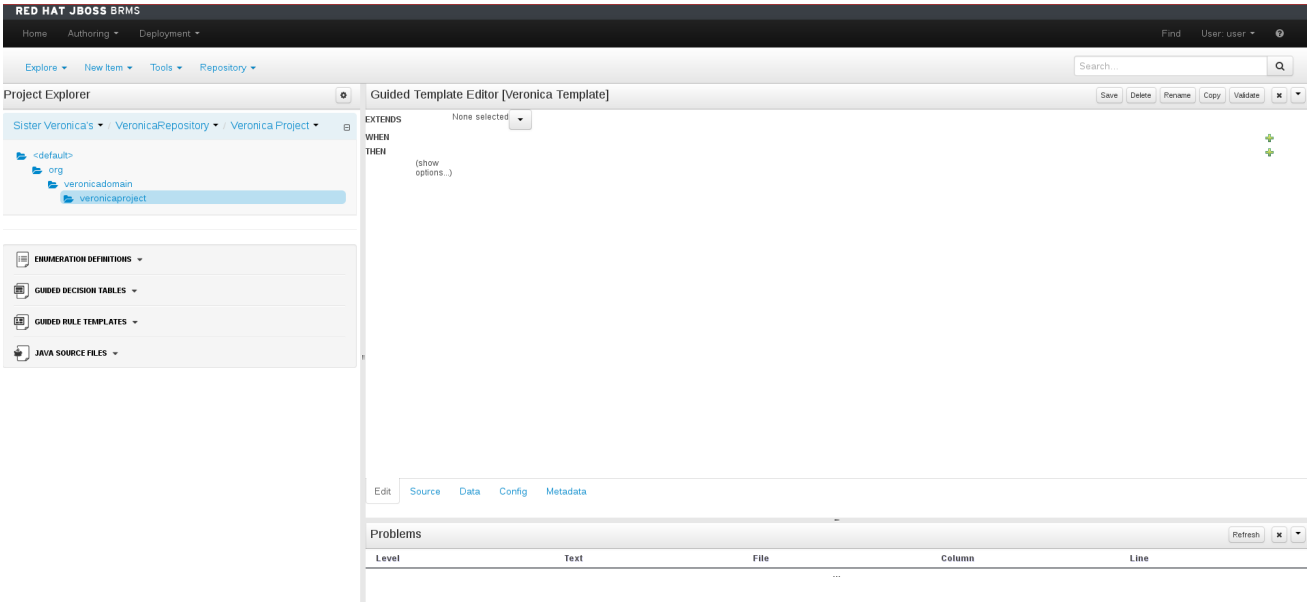


Figure 6.25. Guided Template Editor

NOTE

Rule templates created within Business Central are supported but only when created and used through Business Central. Explicit use of drools-templates api and jar is not.

6.5.2. WHEN conditions in the Guided Rule Template

The **`Guided Template Editor`** within Business Central allows users to set rule templates where the data is completely separate from the rules.

The editor contains Edit, Source, Data, Config, and Metadata tabs.

Within this section, we will explore how to alter the **`WHEN`** constrains within the Edit tab.

NOTE

In the Guided Rule Template example procedures below, a Nurse Rostering data model was created for a fictitious hospital, Sister Veronica's .

Procedure 6.6. Using the Guided Template Editor with WHEN constraints

1. Assuming you have already set up a Data Model for your project (as described in Section 5.2, “Creating a data object”), select the plus icon  to the right of the **`WHEN`** section of the Guided Template Editor.
2. A dialog window will appear with available condition templates to choose from. In the example below, we select the **`Nurse...`** condition from the list.

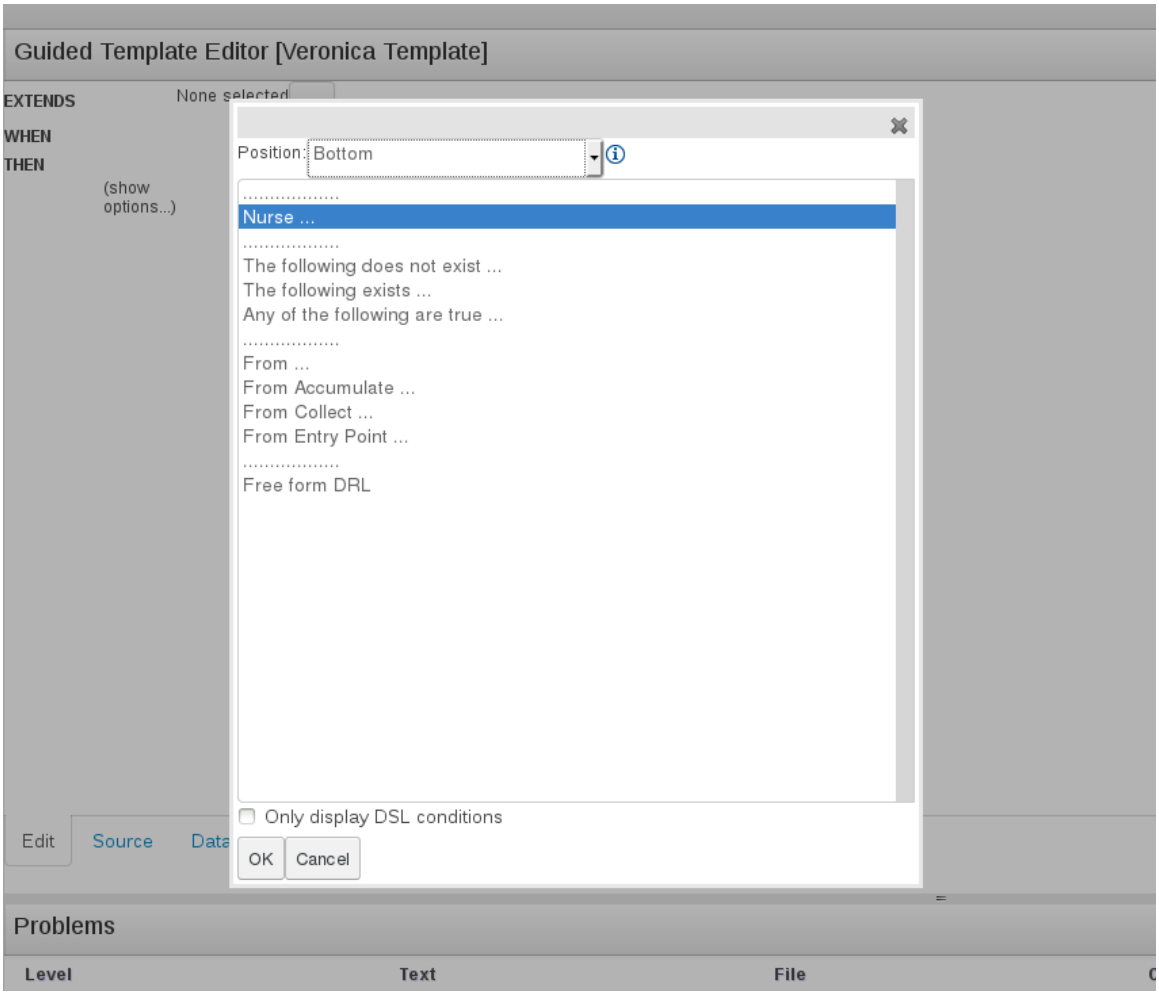


Figure 6.26. Nurse Roster WHEN Dialog Window

3. Click **OK** and the Guided Template Editor will display your **WHEN** condition.

4. Click on the newly added **WHEN** condition. In the example below, it is the "There is a Nurse" condition. A "Modify constraints for Nurse" dialog appears.

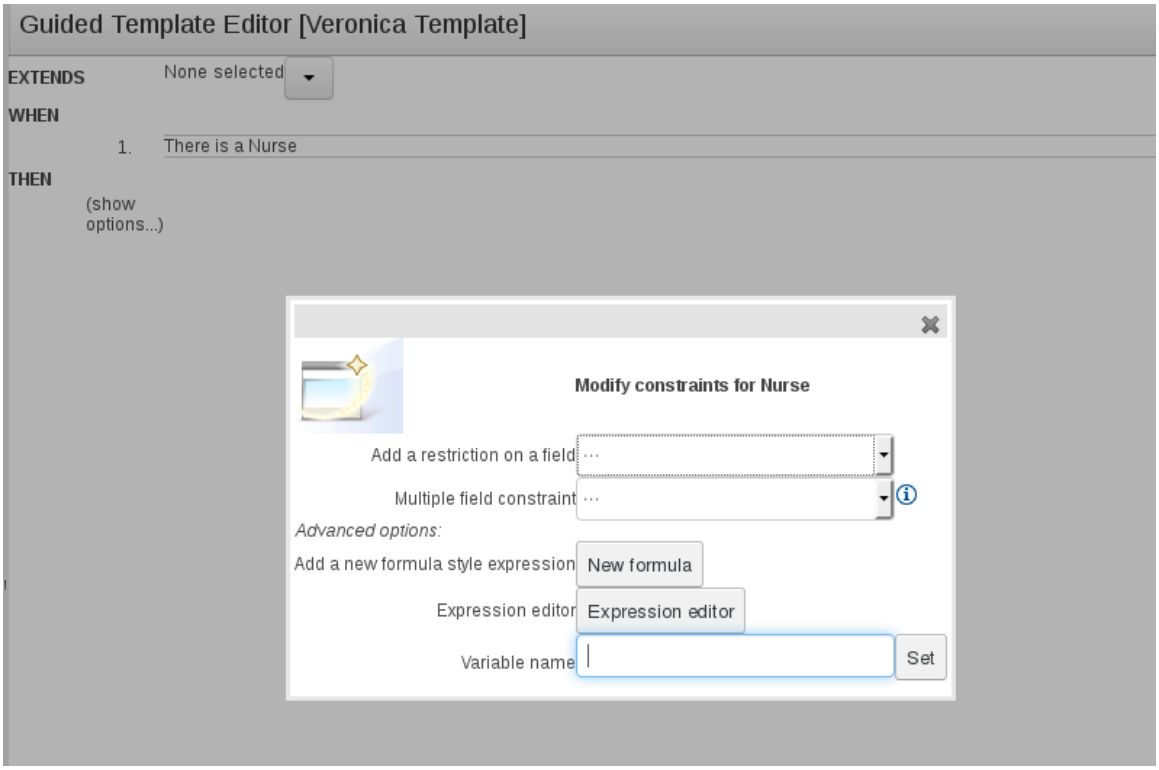


Figure 6.27. Modify Constraints Dialog

5. From here you can add a restriction on a field, apply multiple field constraints, add a new formula style expression, apply an expression editor, or set a variable name.

6. In the example below, we will add a restriction of **`servicelength`** to the condition.

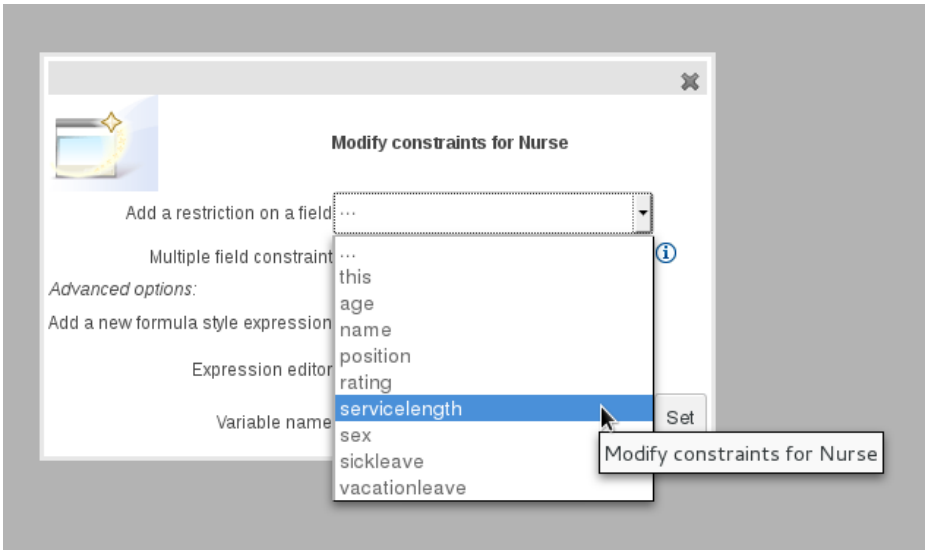


Figure 6.28. Title

7. Once selected, the dialog window closes automatically.

8. Next to the newly selected restriction will be a drop down box to choose an operator. In the example below, we have chosen an operator of "less than."

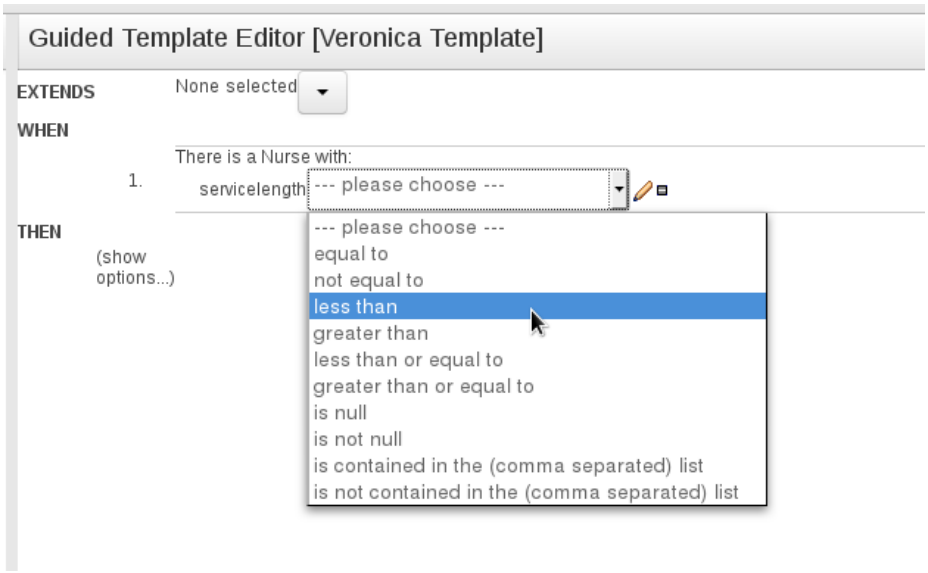


Figure 6.29. Restriction drop-down menu

9. By selecting the **`Edit Icon`**  within the restrictions field, you will be able to define the field value with a literal value, template key, a formula, or expression editor.

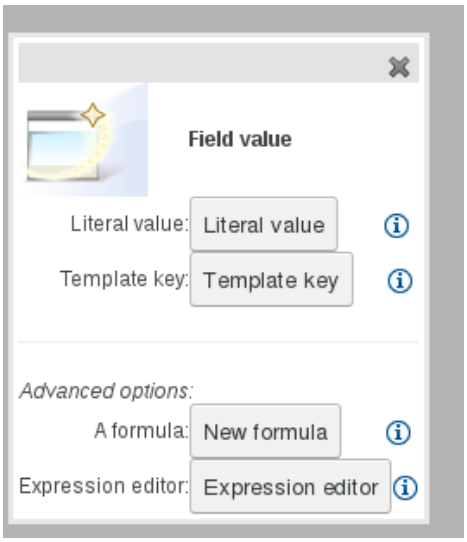


Figure 6.30. Field Value Options

10. By clicking on the **`WHEN`** condition again, we can supply a variable name to help define the restriction. In the example below, we name it "nurse" and click the Set button.

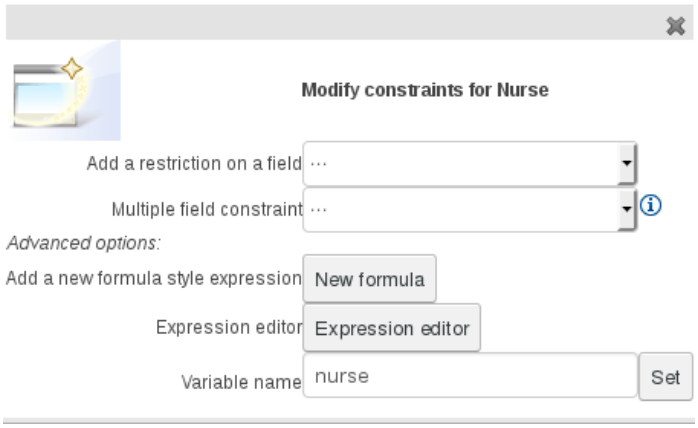


Figure 6.31. Setting a variable name

11. Continue to add **WHEN** conditions as appropriate for the project. The example below demonstrates "servicelength" and "rating" constraints.

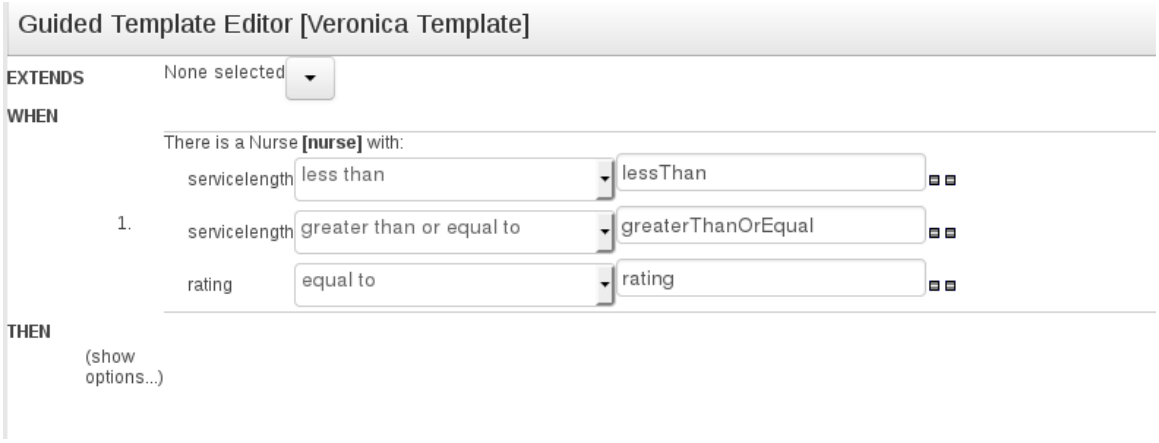


Figure 6.32. WHEN Constraints

6.5.3. THEN actions in the Guided Rule Template

The **THEN** section of a rule holds the actions to be executed when it matches the **WHEN** section.This section explores how to alter the **THEN** actions within the Edit tab of the Guided Template Editor.

Procedure 6.7. Using the Guided Template Editor with THEN actions

1. Select the plus icon to the right of the **THEN** section of the Guided Template Editor to input **THEN** actions.

2. A dialog window will appear with available action templates to choose from. In the example below, we select the **`Modify nurse...`** action from the list.

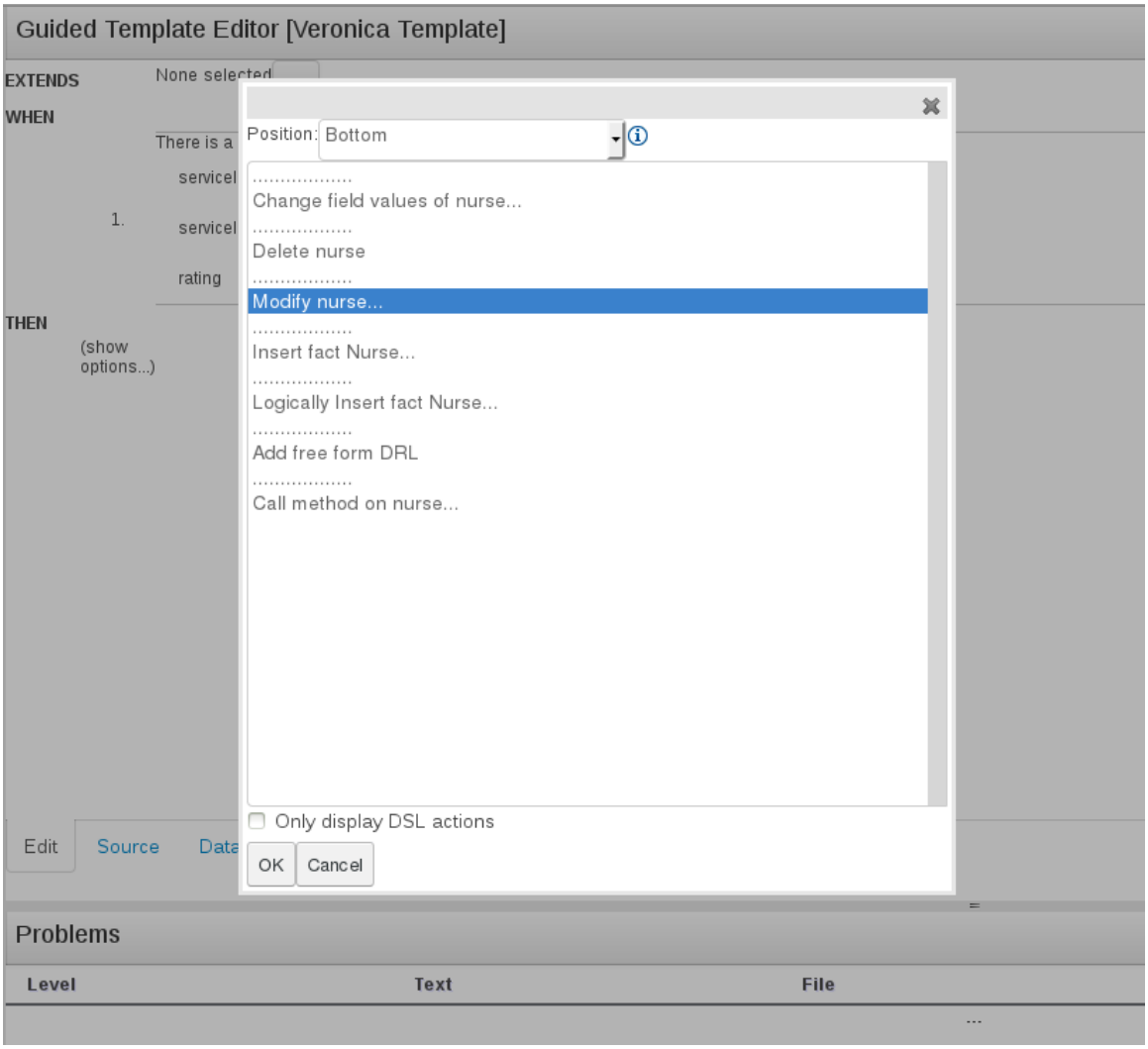


Figure 6.33. Nurse Roster THEN Dialog Window

3. Click **OK** and the Guided Template Editor will display your **THEN** action.
4. Click on the newly added **THEN** action. In the example below, it is the "Modify value of Nurse `[nurse]`" action. An "Add a field" dialog appears.

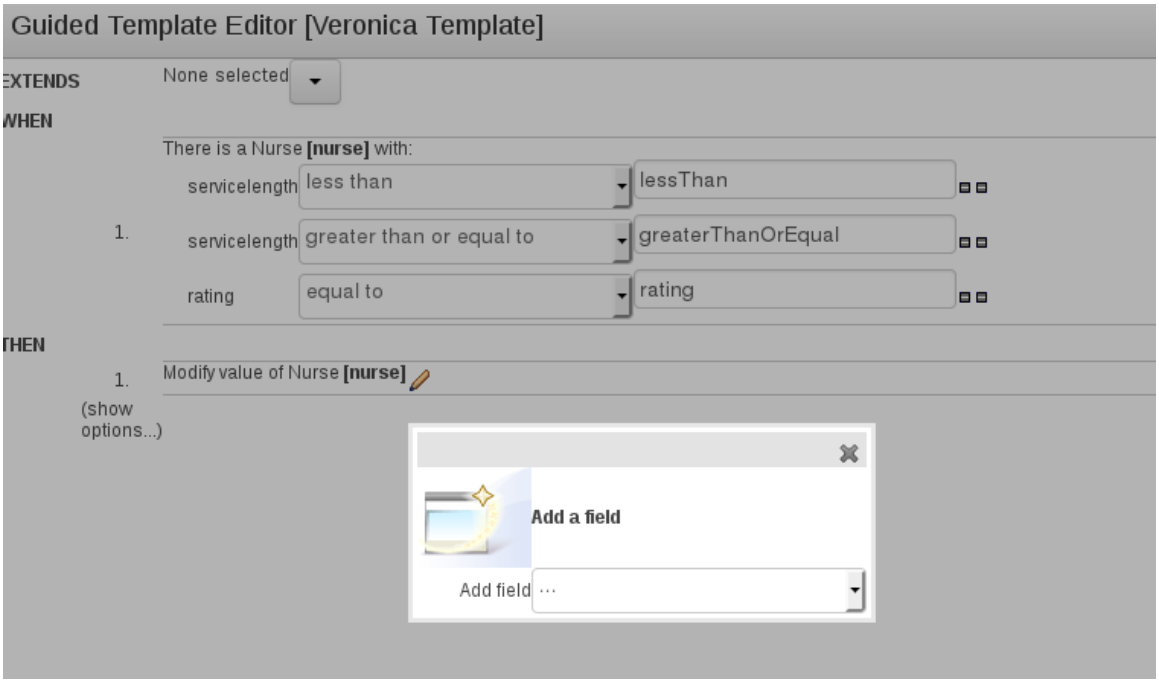


Figure 6.34. Add a field Dialog

5. Within this dialog, you can choose a field from the Add field drop-down menu.
6. Once selected, the dialog window closes automatically.
7. By selecting the **`Edit Icon`** within the item field, you will be able to define the field value with a literal value, template key, or a formula.

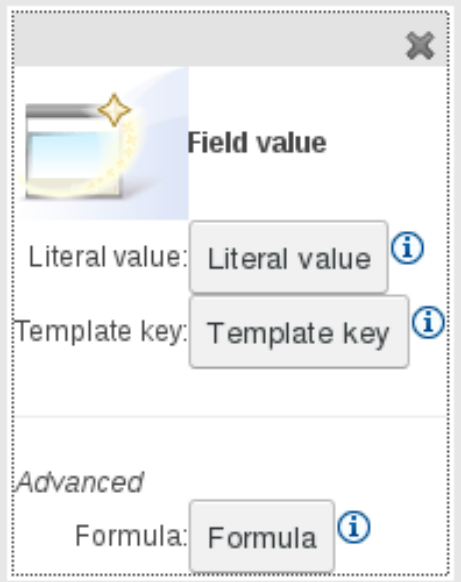


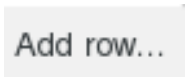
Figure 6.35. Field Value Options

6.5.4. Data Tables in the Guided Rule Template

Data tables can be altered within the Guided Template Editor directly by clicking on the Data tab. The procedure below illustrates how to alter the data created within Guided Template Editor itself.

Procedure 6.8. Using the Guided Template Editor with Data Tables

1. Click on the Data tab at the bottom of the Guided Template Editor in order to access the newly created data table.

2. Click the **`Add row...`**  button to add more table rows.

3. Input additional data into the table. In the example below, we see the ServiceLessThan, ServiceGreaterThan, EmployeeRating, and VacationTime column options and supply data to each field.

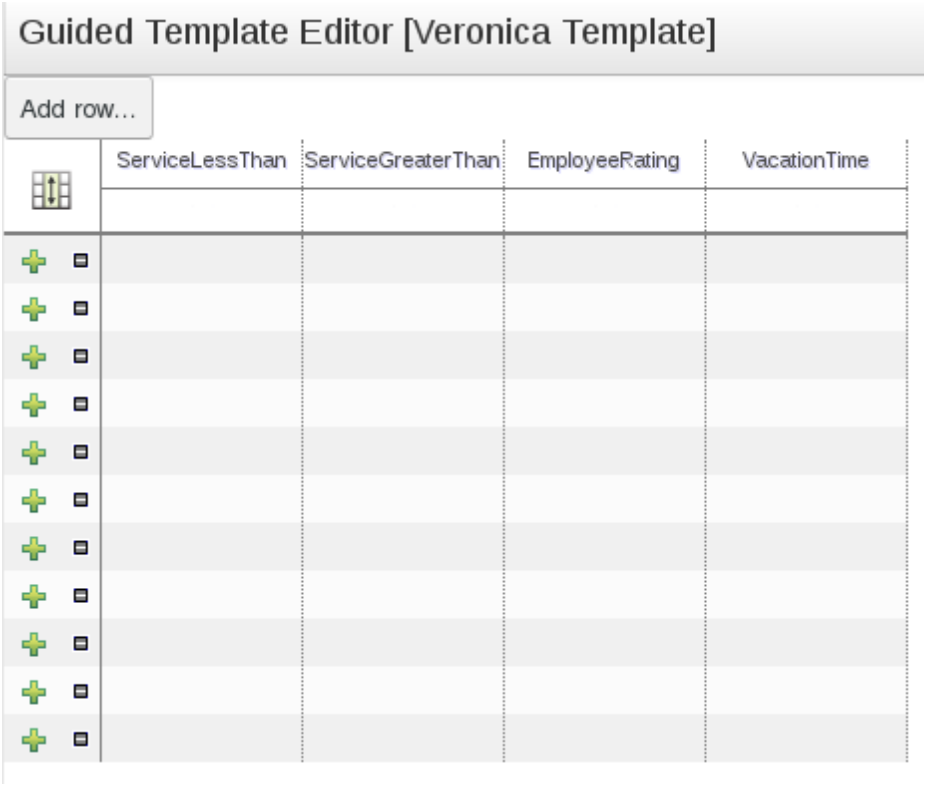


Figure 6.36. Data Table for Guided Template Editor

4. To view the code source, click the Source tab at the bottom of the Guided Template Editor. Illustrated below is the source code for the Nurse example in the Veronica Template.

Figure 6.37. Source Code for Nurse Example

5. Save the template when you are finished working in the Guided Template Editor.

6.6. THE DOMAIN SPECIFIC LANGUAGE EDITOR

Sentence constructed from domain specific languages (or DSL sentences) can be edited in the DSL editor. Please refer to the *JBoss Development Guide* for more information about domain specific languages. The DSL syntax is extended to provides hints to control how the DSL variables are rendered. The following hints are supported:

- {<varName>:<regular expression>}

  This will render a text field in place of the DSL variable when the DSL sentence is used in the guided editor. The content of the text field will be validated against the regular expression.

- {<varName>:ENUM:<factType.fieldName>}

  This will render an enumeration in place of the DSL variable when the DSL sentence is used in the guided editor. <factType.fieldName> binds the enumeration to the model fact and field enumeration definition. This could be either a Knowledge Base enumeration or a Java enumeration, i.e., defined in a model POJO JAR file.

- {<varName>:DATE:<dateFormat>}

This will render a date selector in place of the DSL variable when the DSL sentence is used in the guided editor.

- {<varName>:BOOLEAN:<[checked | unchecked]>}

  This will render a dropdown selector in place of the DSL variable, providing boolean choices, when the DSL sentence is used in the guided editor.

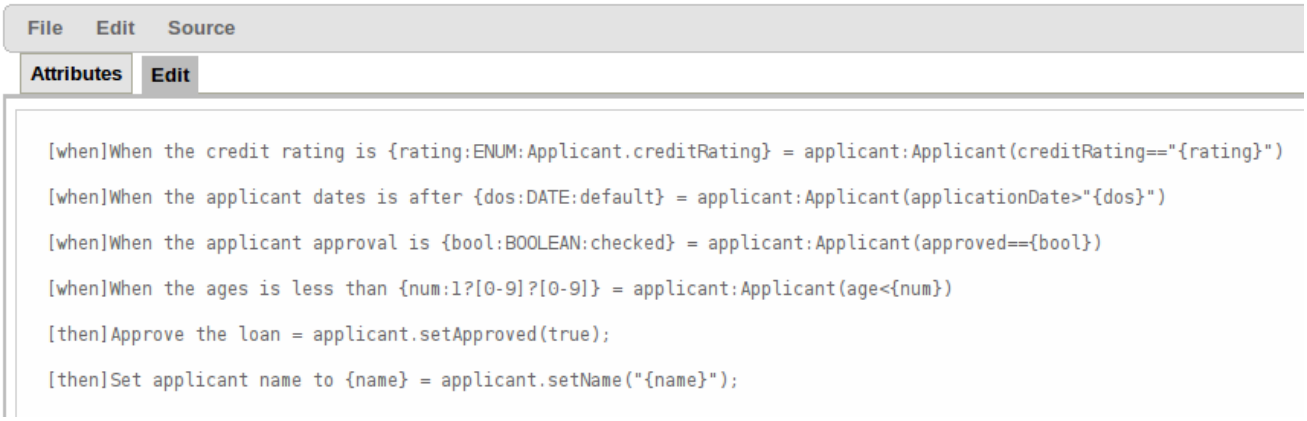


Figure 6.38. DSL Editor

6.7. DATA ENUMERATIONS

6.7.1. Data Enumerations Drop Down List Configuration

Data enumerations are an optional type of asset that can be configured to provide drop-down lists for the guided editor. They are stored and edited just like any other asset and only apply to the package they are created in.

The contents of an enumeration configuration are the mapping of a **`fact.field`** to a list of values. These values are used to populate the drop-down menu. The list can either be literal or use a utility class (which must be added to the classpath) to load the strings. The strings contain either a value to be shown in the drop-down menu or a mapping from the code value (which is what is used in the rule) and a display value, e.g., M=Mini.

Example 6.2. An Example Enumeration Configuration

```
'Board.type' : [ 'Short', 'Long', 'M=Mini', 'Boogie']
'Person.age' : [ '20', '25', '30', '35' ]
```

6.7.2. Advanced Enumeration Concepts

Drop-down lists are dependent on field values. With enumerations it is possible to define multiple options based on other field values.

A fact model for insurance policies could have a class called Insurance, consisting of the fields, **`policyType`** and **`coverage`**. The choices for **`policyType`** could be **`Home`** or **`Car`**. The type of insurance policy will determine the type of coverage that will be available. A home insurance policy could include **`property`** or **`liability`**. A car insurance policy could include **`collision`** or **`fullCoverage`**.

The field value policyType determines which options will be presented for coverage, and it is expressed as follows:

```
'Insurance.policyType' : ['Home', 'Car']
'Insurance.coverage[policyType=Home]' : ['property', 'liability']
'Insurance.coverage[policyType=Car]' : ['collision', 'fullCoverage']
```

6.7.3. Obtaining Data Lists from External Sources

A list of Strings from an external source can be retrieved and used in an enumeration menu. This is achieved by adding code to the classpath that returns a **`java.util.List`** (of strings). Instead of specifying a list of values in the user interface, the code can return the list of strings. (As normal, you can use the "=" sign inside the strings if you want to use a different display value to the rule value.) For example, you could use the following:

```
'Person.age' : ['20','25', '30', '35']
```

To:

```
'Person.age' : (new com.yourco.DataHelper()).getListOfAges()
```

This assumes you have a class called **`DataHelper`** which has a method **`getListOfAges()`** which returns a list of strings. The data enumerations are loaded the first time the guided editor is used in a session. To check the enumeration has loaded, go to the package configuration screen. You can "save and validate" the package; this will check it and provide feedback about any errors.

6.8. SCORECARDS

6.8.1. Scorecards

Scorecard is a Risk Management tool which is a graphical representation of a formula used to calculate an overall score. It is mostly used by financial institutions or banks to calculate the risk they can take to sell a product in market. Thus it can predict the likelihood or probability of a certain outcome. JBoss BRMS now supports additive scorecards that calculates an overall score by adding all partial scores assigned to individual rule conditions.

Additionally, Drools Scorecards will allows for reason codes to be set, which help in identifying the specific rules (buckets) that have contributed to the overall score. Drools Scorecards will be based on the PMML 4.1 Standard.

In general, a scorecard can be created more or less in this way:

1. A statistical analysis is performed on the historical data which is usually collected from the existing customer database.
2. A predictive or probable characteristics (attributes or pieces of information) are identified based on this analysis.
3. Each characteristics are then broken down into ranges of possible values which are then given a score.

To explain it in detail, following is an example:

Characteristics	Expected Score	Range	Score
Family Income	50	1 - 30000	10
		30001 - 60000	25
		60001 - 90000	40
		90001 - 120000	65
		Over 120000	75

Figure 6.39. Scorecard Example

6.8.2. Creating a Scorecard

Procedure 6.9. Creating a new Score Card (Spreadsheet)

1. Open the **`Project Authoring`** perspective: on the main menu, click **Authoring** → **Project Authoring**.
2. In the **`Project Explorer`** view, do the following:
   - If in the Project view of Project Explorer, select the organizational unit, repository and the project where you want to create the score card.
   - If in the Repository view of Project Explorer, navigate to the project root, where you want to create the score card.
3. In the perspective menu, go to **New Item** → **Score Card (Spreadsheet)**.
4. In the **`Create new Score Card (Spreadsheet)`** dialog window, define the package details:
   a. In the **`Resource Name`** text box, enter the score card name.
   b. Click on **`Choose File`** and browse to the location to select the spreadsheet in which the score card is initially created.
5. Click **`OK`**.
6. The new score card spreadsheet is created under the selected project.

CHAPTER 7. BUILDING AND DEPLOYING ASSETS

Packages or assets can be build and deployed by clicking on the **`Build & Deploy`** button as shown in the following screen:

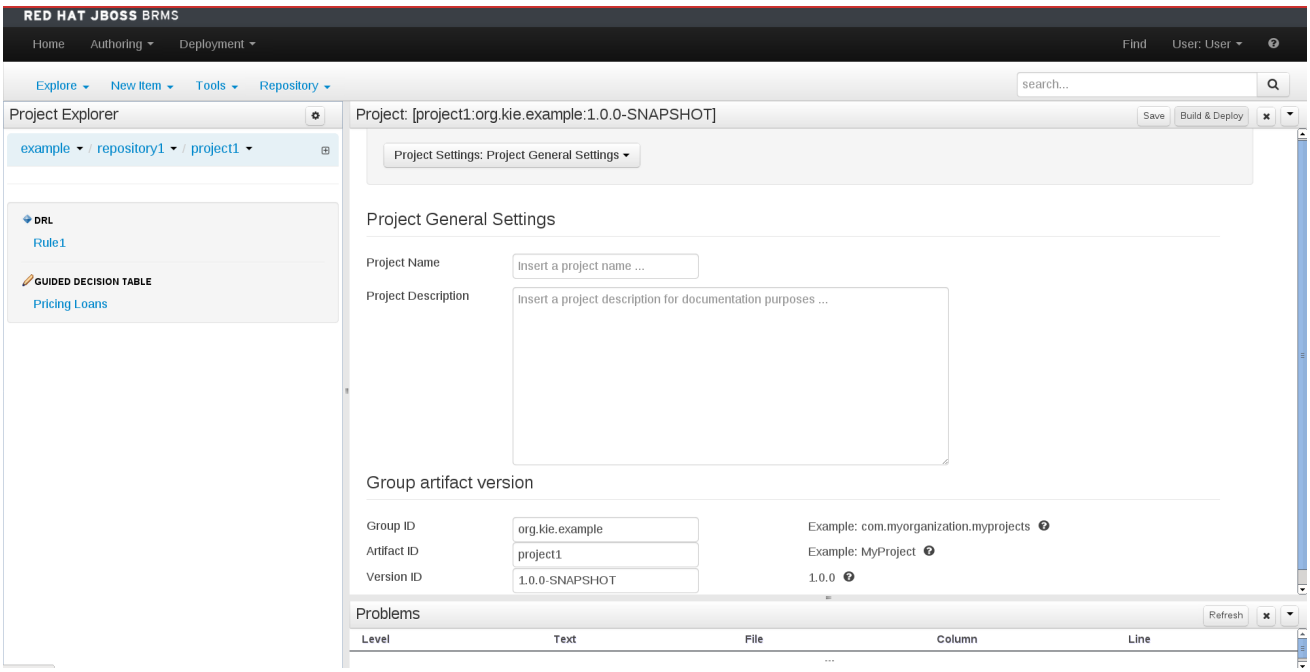


Figure 7.1. Build & Deploy Option

The **`Problems`** window below the project screen displays the errors like compilation errors that can cause a failure in the build.

A user can choose to build a whole package or a subset of it.

In case of large number of rules, the build might take some time. On successful build, a user can download the binary package as a pkg file. The DRL that a built package results in can be viewed by clicking on the package source link.

CHAPTER 8. MANAGING ASSETS

8.1. VERSIONS AND STORAGE

Business Rules, Process definition files and other assets and resources created in Business Central are stored in Artifact repository (Knowledge Store), which is accessed by the Execution Server.

Knowledge store is a centralized repository for your business knowledge. It connects multiple repositories (currently only GIT repositories are supported) so that you can access them from a single environment while allowing you to store different kinds of knowledge and artifacts in different locations. Business Central provides a web front-end that allows users to view and update the store content. You can access it using the **Project Editor** `Project Explorer` from the unified environment of Red Hat JBoss BRMS.

Git is a distributed version control system and it implements revisions as commit objects. Every time when you commit your changes into a repository this creates a new commit object in the Git repository. Similary, the user can also copy an existing repository. This copying process is typically called cloning and the resulting repository can be referred to as clone. Every clone contains the full history of the collection of files and a cloned repository has the same functionality as the original repository.

CHAPTER 9. TESTING

9.1. TEST SCENARIOS

Test Scenarios is a powerful feature that provides the ability for developers to validate the functionality of rules, models, and events. In short, Test Scenarios provide you the ability to test your knowledge base before deploying it and putting it into production.

Test Scenarios can be executed one at the time or as a group. The group execution contains all the Scenarios from one package. Test Scenarios are independent, one Scenario can not affect or modify the other.

After running all the Test Scenarios a report panel is shown. It contains either a success message or a failure message for test scenarios that were run.

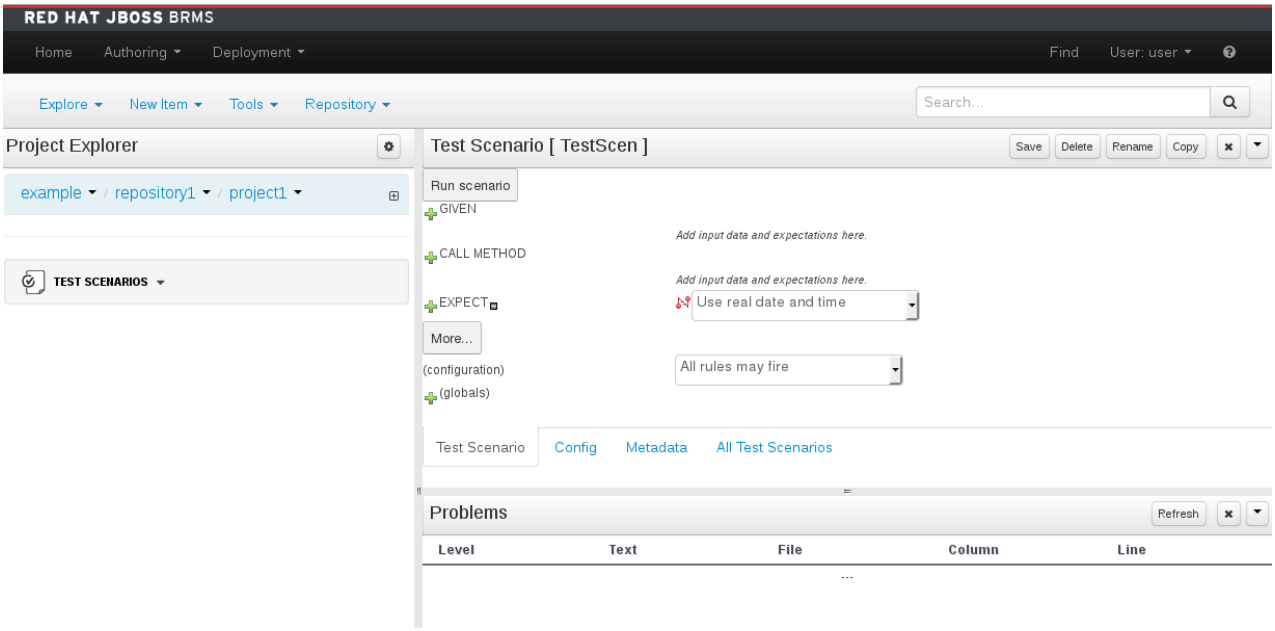


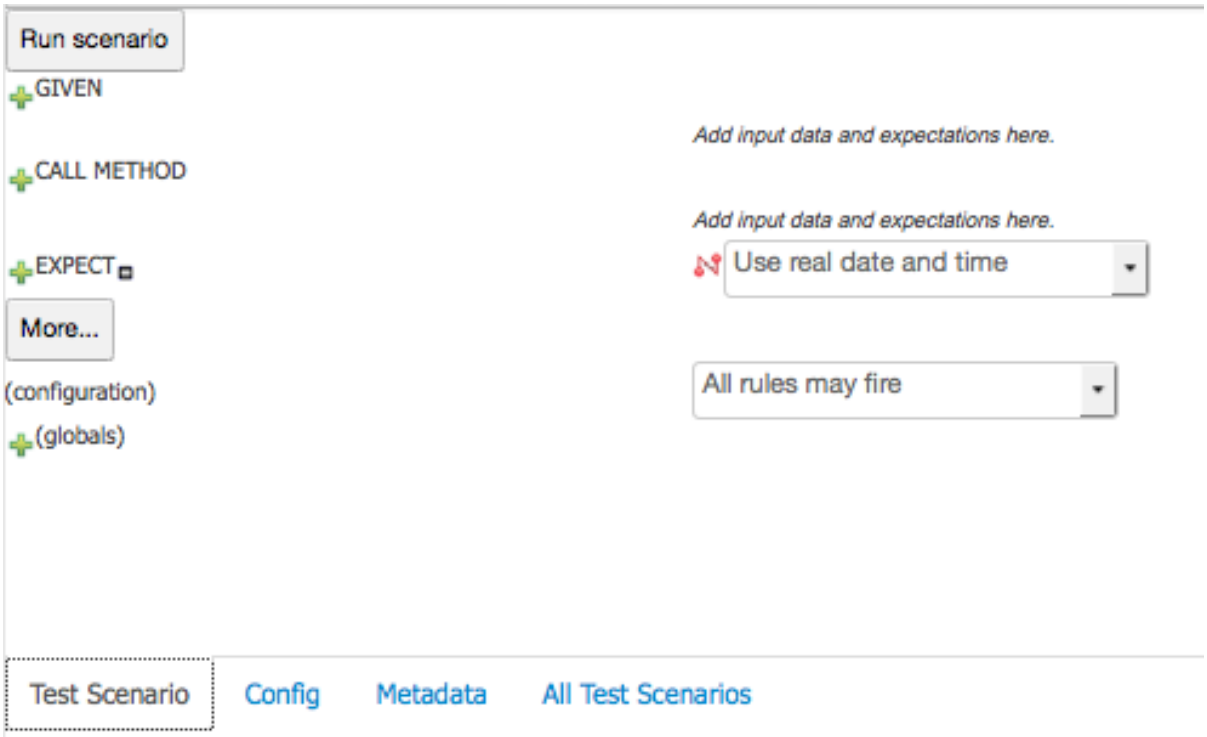
Figure 9.1. Test Scenario Screen

9.2. CREATING A TEST SCENARIO

Creating a Test Scenario requires you to provide data for conditions which resemble an instance of your fact or project model. This is matched against a given set of rules and if the expected results are matched against the actual results, the Test Scenario is deemed to have passed.

Procedure 9.1. Creating a new Test Scenario

1. Open the **`Projects`** view from the **Project Authoring** menu.
2. Select the project where the test scenario is to be created.
3. From the **`New Item`** dropdown menu on the toolbar, select **`Test Scenario`** from the listed options.
4. Enter the Test Scenario name in the pop-up dialog box and click **`OK`**.
5. You will be presented with the Test Scenario edit screen.



Procedure 9.2. Importing a model for the Test Scenario

1. Use the tabs at the bottom of the screen to move between the **`Test Scenario`** edit screen and the **`Config`** edit screen.
2. The **`Config`** screen allows you to specify the model objects that you will be using in this Test Scenario.
3. Data objects from the same package are available by default. For example, if you have a package structure **`org.company.project`**, and you have a data object Fact1 in package **`org.company`** and a Fact2 in package **`org.company.project`**, you will not have to import model objects for Fact1 if you want to create a Test Scenario in package **`org.company`**; however, you will need to import Fact2 if you want to use it.
4. To import the data objects that are required for your Scenario, click on the **`New Item`** button in the **`Config`** screen.
5. These imports can be specific to your project's data model or generic ones like **`String`** or **`Double`** objects.

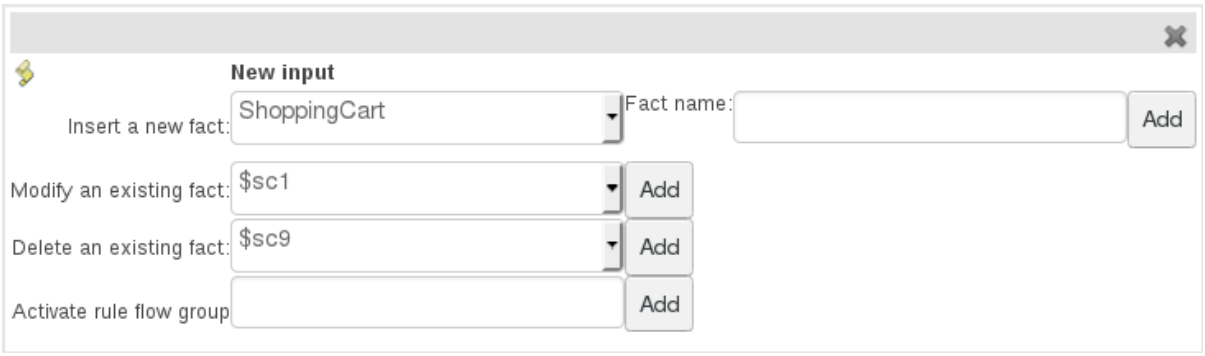
Procedure 9.3. Providing Test Scenario Facts

1. After you have imported the data objects, come back to the Test Scenario screen and enter the variables for your Scenario.
2. At the minimum, there are two sections that require input: **`GIVEN`** and **`EXPECT`**
   - GIVEN: What are the input facts for this Test Scenario?
   - EXPECT: What are the expected results given the input facts from the **`GIVEN`** section?
3. **`GIVEN`** these input parameters, **`EXPECT`** these rules to be activated or fired. You can also **`EXPECT`** facts to be present and to have specific field values or **`EXPECT`** rules not to fire at all.

If the expectations are met, then the Test Scenario has passed and your rules have been created correctly. If the expectations are not met, then the Test Scenario has failed and you need to check your rules.

Procedure 9.4. Providing Given Facts

1. To add a new fact, click on the green + button next to the **GIVEN** label. This will bring up the **New Input** dialog box. Provide your fact data in this window based on the data models that you have imported in the **Config** screen.



You can select a particular data object from the model and give it a variable name (called **Fact Name** in the window), or choose to activate a rule flow group instead. If you choose to activate a rule flow group, you are allowing rules from a rule flow group to be tested by activating the group in advance. If you want to add a given fact and activate a rule flow group, you have to add the given fact, click the green + button again, and then add the activation.

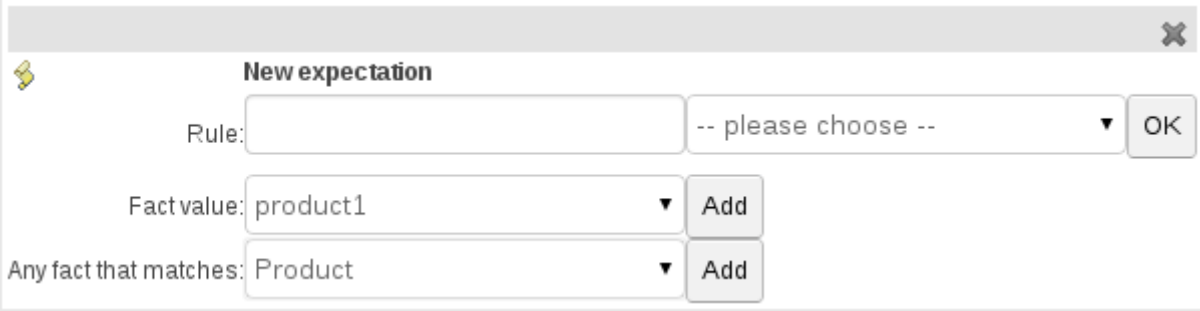
2. Depending upon the model that you select, you will be able to provide values to its editable properties as part of your GIVEN fact. For example, if your model was a **Product**, you might have properties like **itemID**, **productName** and **price**. You get to these properties by clicking on the text **Insert Product**.

By clicking on the pencil icon next to the property, you can edit the property to provide either a literal value for that property that should form part of your GIVEN fact data or you can provide advanced fact data. See Section 9.3, “Additional Test Scenario Features” for more information.

Procedure 9.5. Providing Expected Rules

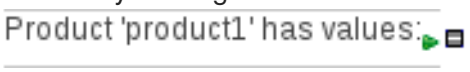
1. Once you are satisfied with the Given rule conditions, you can expect that rules that will be fired if the Given rule conditions are met when the Test Scenario is run.

2. To do so, click on the green + button next to the **EXPECT** label. A **New expectation** dialog box will come up.



3. You can provide one of three expectations given the set of data that was created in the Given section. You can:

   - Either type in the name of a rule that is expected to be fired or select it from the list of rules and then click the **OK** button.

   - Expect a particular instance of a model object (and one or more of its properties) to have a certain value by selecting that instance from the drop down in the **Fact Value** field. For example, **product1** shown in the figure, which is an instance of the fictitious **Product** model created in the Given section. You specify the property values by first adding that instance by clicking the **Add** button and then clicking a green arrow 

to bring up the fields to add. Once you have selected the field to add, you can provide a literal value for that field.

   - Expect a fact model to match your Given facts by selecting it from the **Any fact that matches** drop down. Instances of data objects are in the working memory, rather than matching what has been set up in the Given section. Rules may change the contents of working memory for example, some facts may be retracted, others inserted, and some will have their properties changed.

     In the figure shown above, you can mandate that instances of the Product's one or more properties match the Given rule conditions.

Procedure 9.6. Reviewing, Saving, and Running a Scenario

1. Once you are satisfied with your Test Scenario's facts, you can save it by clicking the **Save** button in the upper right corner. Ensure you regularly save and review your scenarios.

2. You can now run your Test Scenario by clicking the **Run scenario** button at the top of the Test Scenario screen. The results are displayed at the bottom of this screen in a new panel called **Reporting**.

3. Once you have a bunch of Test Scenarios for a particular package, you can run all of them together by accessing the **All Test Scenarios** tab and then clicking the **Run all scenarios** button.

9.3. ADDITIONAL TEST SCENARIO FEATURES

In addition to the previous Test Scenario features, which encompassed the **GIVEN** and **EXPECT** input facts, Red Hat JBoss BRMS Test Scenarios include various other features.

Procedure 9.7. Calling Methods

1. **`Call Method`** - allows you to call a method on an existing fact in the beginning of the rule execution. This feature is accessed by clicking on the green plus sign next to the **CALL METHOD** feature. This opens up a new input box like the following:

Figure 9.2. Call Method

2. After selecting an existing fact from the drop-down list, click the Add button. The green arrow button allows you to call a method on the fact.

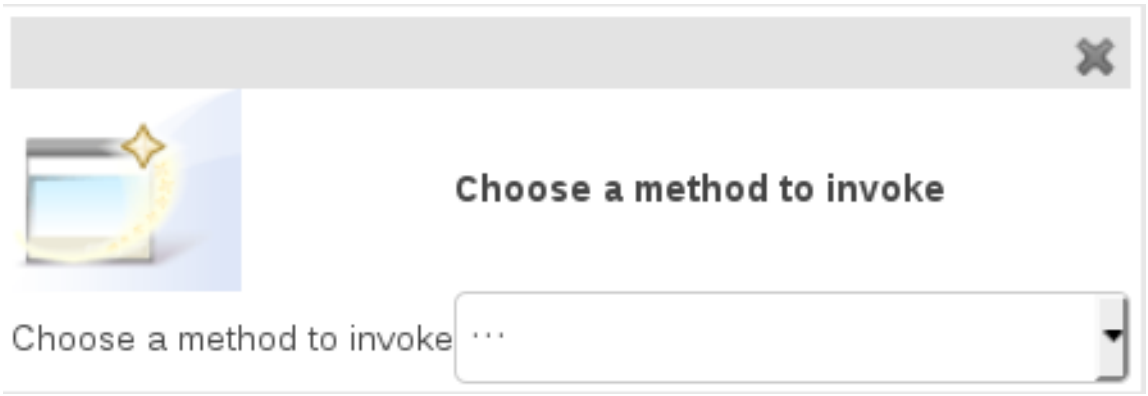


Figure 9.3. Invoke a Method

Procedure 9.8. Adding Globals

1. **`globals`** - validate the global field values. Globals are named objects that are visible to the rule engine but are different from the objects for facts. Accordingly, the changes in the object of a global do not trigger the reevaluation of rules. This feature is accessed by clicking on the green plus sign next to the **(globals)** feature.

Figure 9.4. New Global

2. After selecting an existing global from the drop-down list, click the Add button. Clicking on the newly selected global allows you to assign values to fields of the global variable.

Figure 9.5. Apply Field

3. After selecting an existing field from the drop-down list, click the OK button. Once you have selected the field, you will be able to set its value.  By clicking on this icon, you will be able to edit the field values.

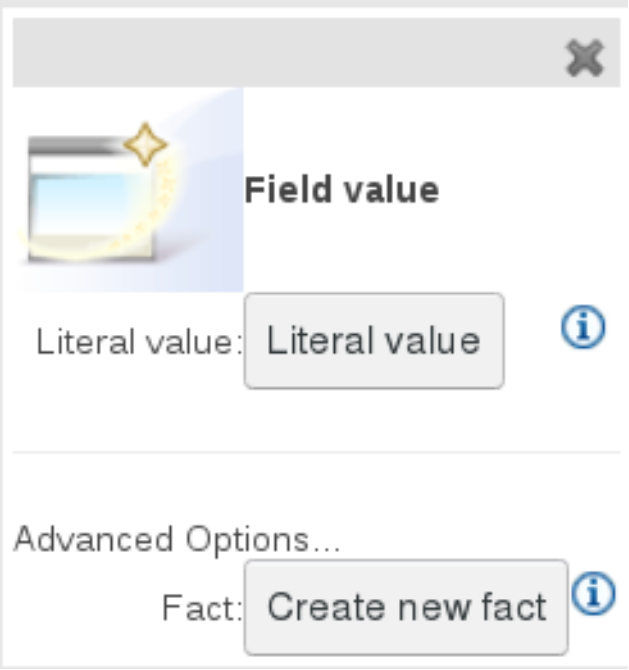


Figure 9.6. Field Values

Procedure 9.9. Configuring Rules

1. **`Configuring rules`** - allows the user to determine additional constraints on the firing of rules by providing the following options:
   - **`Allow these rules to fire:`** - allows you to select which rules are allowed to fire.
   - **`Prevent these rules from firing:`** - allows you to prevent certain rules from firing for the test scenario.
   - **`All rules may fire`** - will allow all the rules to fire for the given test.

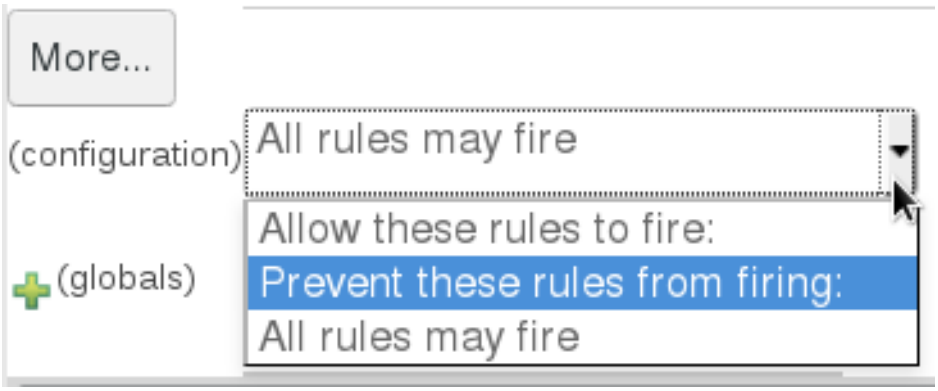


Figure 9.7. Configuration

2. If you select **`Allow these rules to fire:`** or **`Prevent these rules from firing:`**, you will be supplied with an empty field where you can select the rules which pertain to the configuration by clicking the green plus sign next to the empty field. This will open a dialog to select which rules are affected by the condition.

Figure 9.8. Selecting rules

3. Choose a rule from the drop-down list and click the OK button. The selected rules will appear in the field next to the rules configuration option. You can add as many rules as you like to this field.

Procedure 9.10. Date and Time Configuration

1. **`Use real date and time`** - the real time is used when running the test scenario.

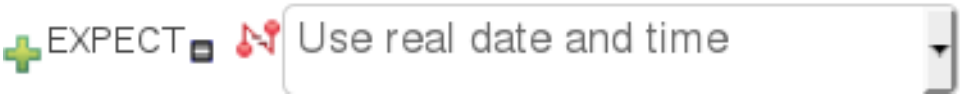


Figure 9.9. Real Date and Time

2. **`Use a simulated date and time`** - this option allows you to specify the year, month, day, hour, and minute associated with the test scenario run.

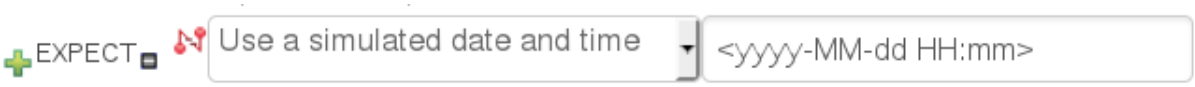


Figure 9.10. Title

Procedure 9.11. Advanced Fact Data

1. After providing values to editable properties as part of your created fact, the Field value dialog box will appear where you can edit literal values or provide advanced fact data.

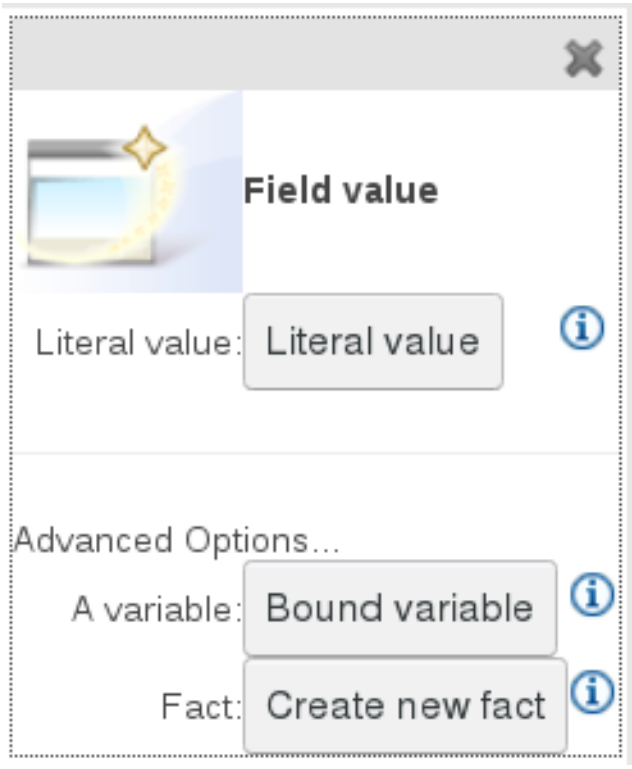


Figure 9.11. Advanced Options

2. Within the Advanced Options section, you will be provided with various choices depending on the type of fact created and the model objects used for the particular Test Scenario.

   - **`Bound variable`** - sets the value of the field to the fact bout to the selected variable. The field must be of the right type.

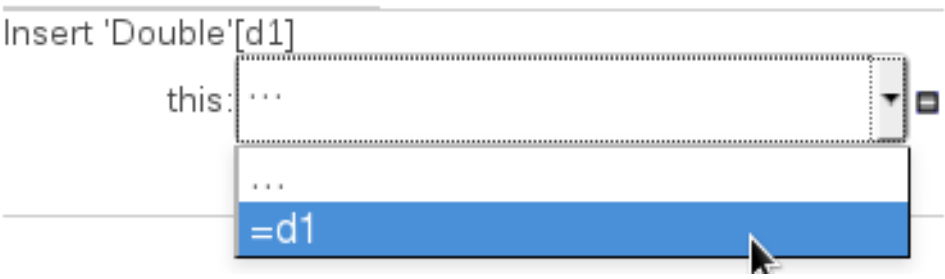


Figure 9.12. Bound Variable

   - **`Create new fact`** - allows you to create a new fact. By clicking on the **Create new fact** button, a new fact will appear as the value of the field. By clicking on that fact, you will be supplied with a drop down of various field values, and these values may be given further field values.

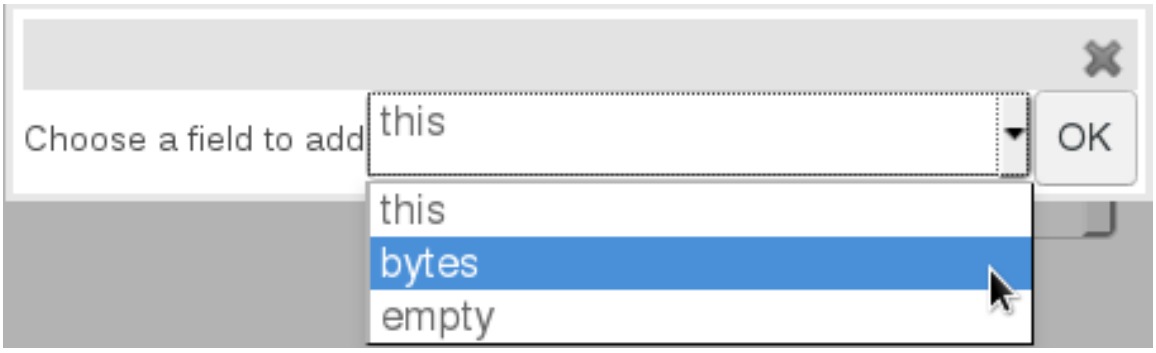


Figure 9.13. Create New Fact

Procedure 9.12. Adding More Sections

- The Test Scenario edit screen allows you to add Given, Call Method, and Expect sections to the scenario by clicking the **`More`**  button below the **`EXPECT`** section. This will open a block encompassing all three sections that can be removed by clicking the delete item  button.

Procedure 9.13. Modifying or Deleting an Existing Fact

- - **`Modifying an existing fact`** - allows the editing of a fact between knowledge base executions.
  - **`Delete an existing fact`** - allows the removing of facts between executions.

Figure 9.14. Modifying and Deleting Existing Facts

CHAPTER 10. THE REALTIME DECISION SERVER

The Realtime Decision Server is a standalone, out-of-the-box component that can be used to instantiate and execute rules via interfaces available for REST, JMS or a Java client side application. Created as a web deployable WAR file, this server can be deployed on any web container.

This server has a low footprint, with minimal memory consumption, and therefore, can be deployed easily on a cloud instance. Each instance of this server can open and instantiate multiple KIE Containers which allows you to execute multiple rule services in parallel.

You can provision the decision server instances via Business Central. In this chapter, you will go through the steps required to setup a decision server, provision and connect to this server via Business Central, control what rule artifacts go in each instance and go through its lifecycle.

10.1. DEPLOYING THE REALTIME DECISION SERVER

The Realtime Decision Server is distributed as a web application archive (WAR) file with the name of **`kie-server.war`**. By default, when you install JBoss BRMS, this WAR file is installed and deployed in your web container. Consequently this gives you automatic access to this web application with the URL of **`http://SERVER:PORT/kie-server/`**.

You can of course, copy this WAR file and deploy it in any other web container (Red Hat JBoss Web Server or another Red Hat JBoss EAP install, for example) and have this server available from other web containers. Note that you must deploy the WAR file that is compatible for your web container. The Realtime Decision Server bundled with the EAP install is only suitable for installation on another JBoss EAP install, while the one bundled with the generic deployment is suitable for installation on Red Hat JBoss Web Server only.

1. Once you have deployed this WAR file (or if you have it deployed on the same web container where JBoss BRMS is deployed), create a user with the role of kie-server in this web container.

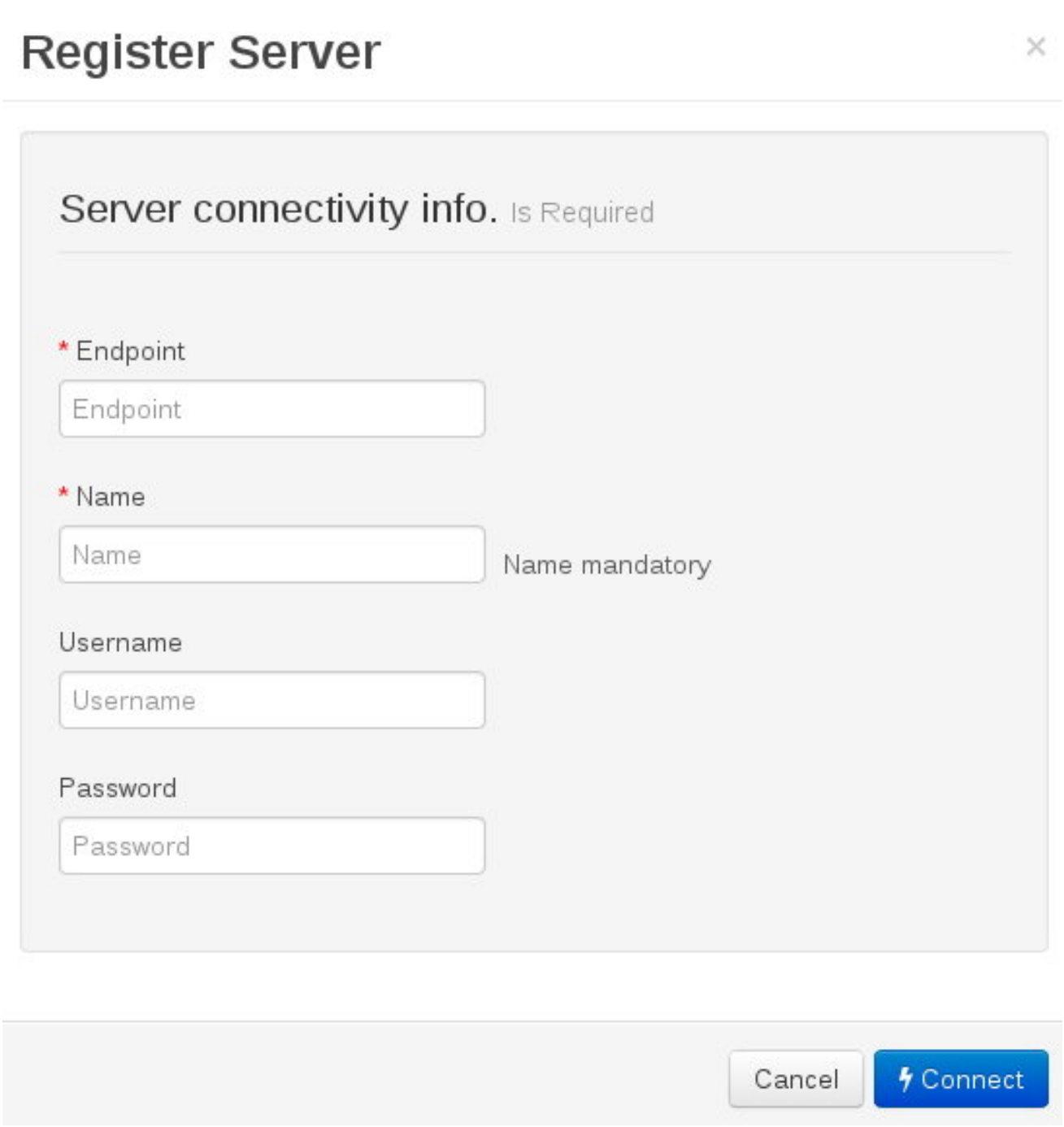
2. Test that you can access the decision engine by navigating to the endpoint in a browser window: **`http://SERVER:PORT/kie-server/services/rest/server/`** with the web container running. You will be prompted for your username/password that you created in the previous step.

3. Once authenticated, you will see an XML response in the form of engine status, similar to this:

```
<response type="SUCCESS" msg="Kie Server info"><kie-server-info>
<version>6.2.0.redhat-1</version></kie-server-info></response>
```

10.2. REGISTERING A DECISION SERVER

To register a new Realtime Decision Server instance, click on **Deploy → Rule Deployments** in Business Central. This will open up the screen to show you any existing servers that you have registered. Click on **Register** to bring up the screen for entering details for a new server.



Enter details of your new server by specifying the endpoint and giving it a unique and identifiable name. The endpoint should look something like this: **`http://localhost:8080/kie-server/`**. Also provide a unique identifiable name for this server.

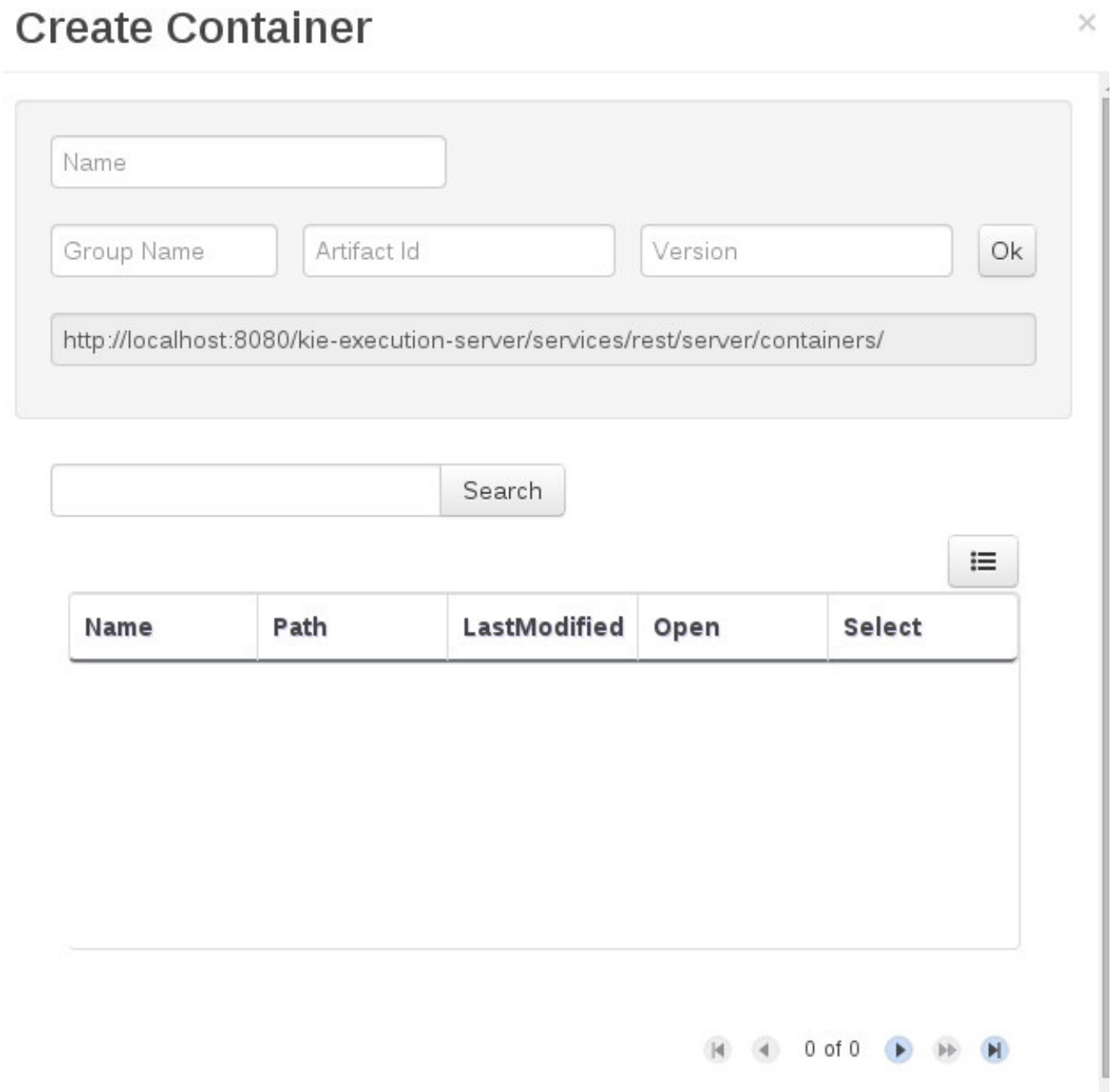
Although the username/password sections are not mandatory, unless you specifically take away the authentication required for your kie-server webapp, then you will need to enter these. These values should match the ones created in the previous section when you were installing this webapp with the role of kie-server.

Click **`Connect`** to connect and register this server. If you have successfully entered all the details, your server will be listed in the Rule Deployments Screen. Common errors at this stage include issues like invalid username/password or the username not having the required kie-server role or the endpoint not being correct. All of these issues will give the same error: **`Can't connect to endpoint`**

10.3. CREATING A CONTAINER

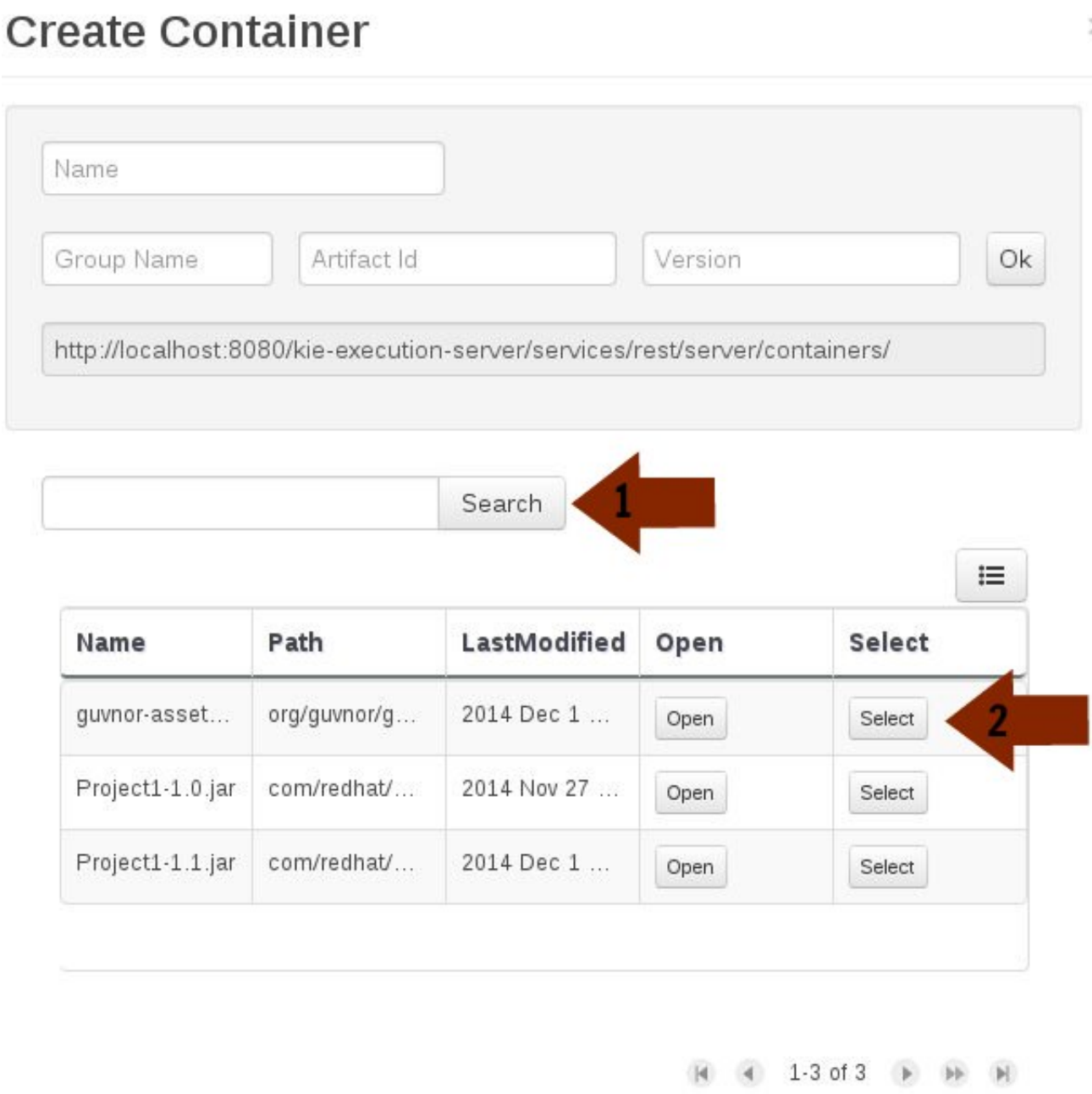
Once your Decision Server is registered, you can start adding Containers to it. Containers are self contained environments that have been provisioned to hold instances of your packaged and deployed rule instances.

Start by clicking the + icon next to the Decision Server where you want to deploy your Container. This will bring up the New Container screen.

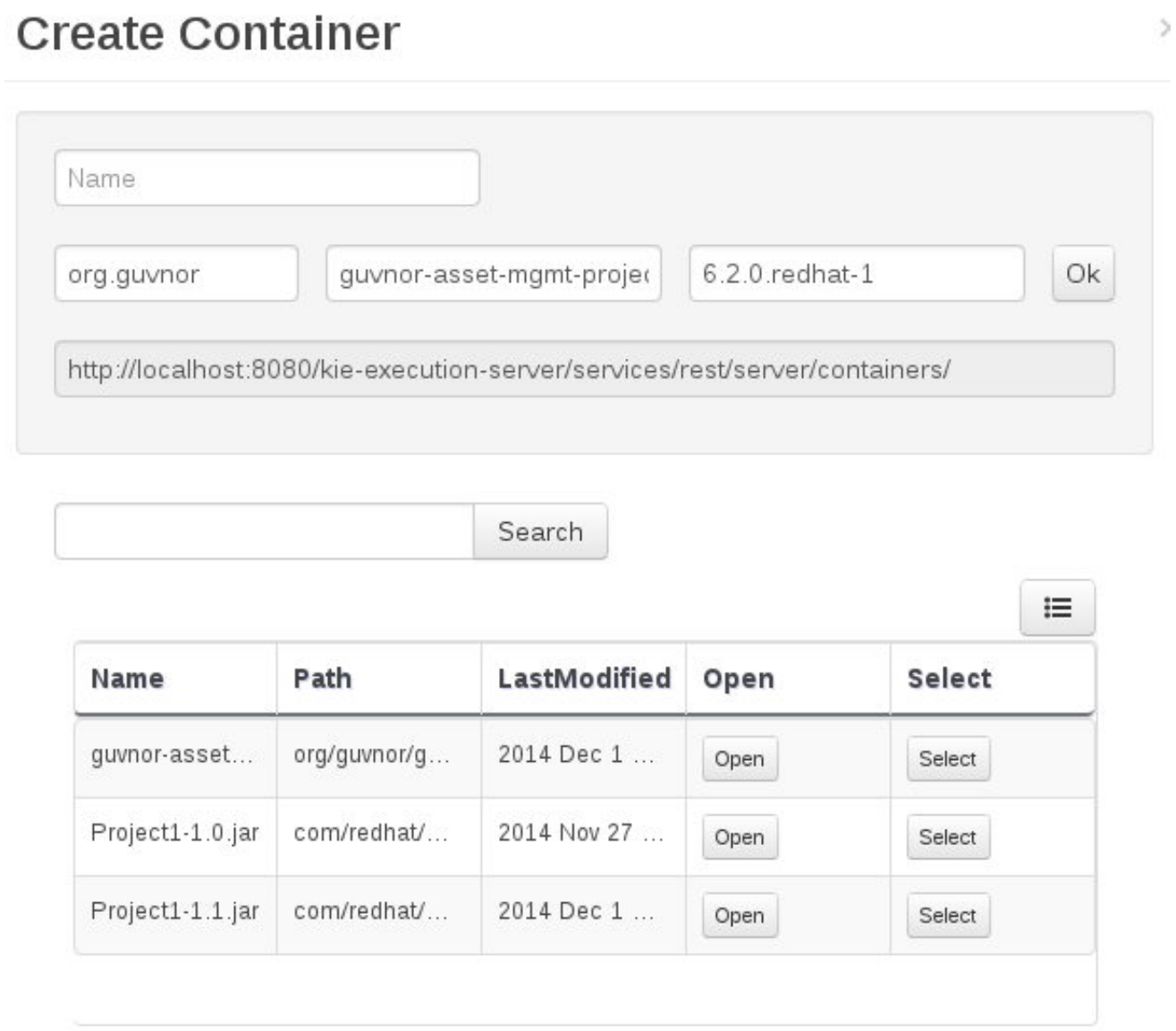


If you know the Group Name, Artifact Id and Version (GAV) of your deployed package, then you can enter those details and click the **Ok** button to select that instance (and provide a name for the Container).

If you don't know these values, you can search Business Central for all packages that can be deployed. Click the **Search** button without entering any value in the search field (you can narrow your search by entering any term that you know exists in the package that you want to deploy).



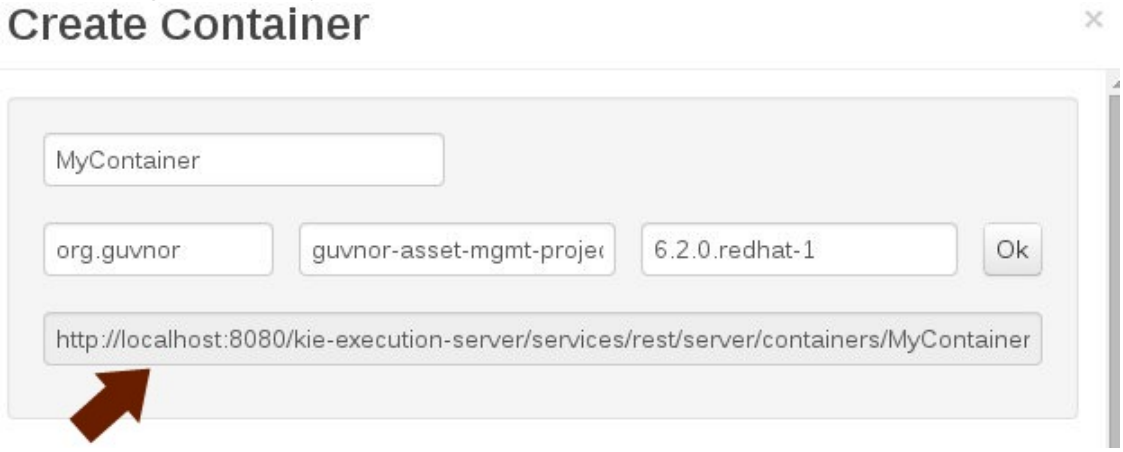
The figure above shows that there are three deployable packages available to be used as containers on the Decision Server. Select the one that you want by clicking the **`Select`** button. This will auto-populate the GAV and you can then click the **`Ok`** button to use this deployable as the new Container.



You are not done! This Container needs a name. Enter a name for this Container at the top and then press the **0k** button.

NOTE

Just below the GAV row, you will see an uneditable row that shows you the URL for your Container against which you will be able to execute REST commands.

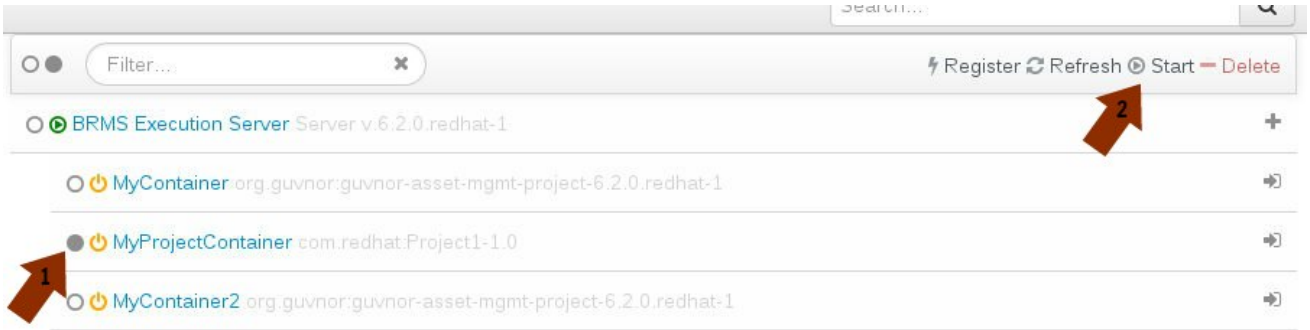


10.4. MANAGING CONTAINERS

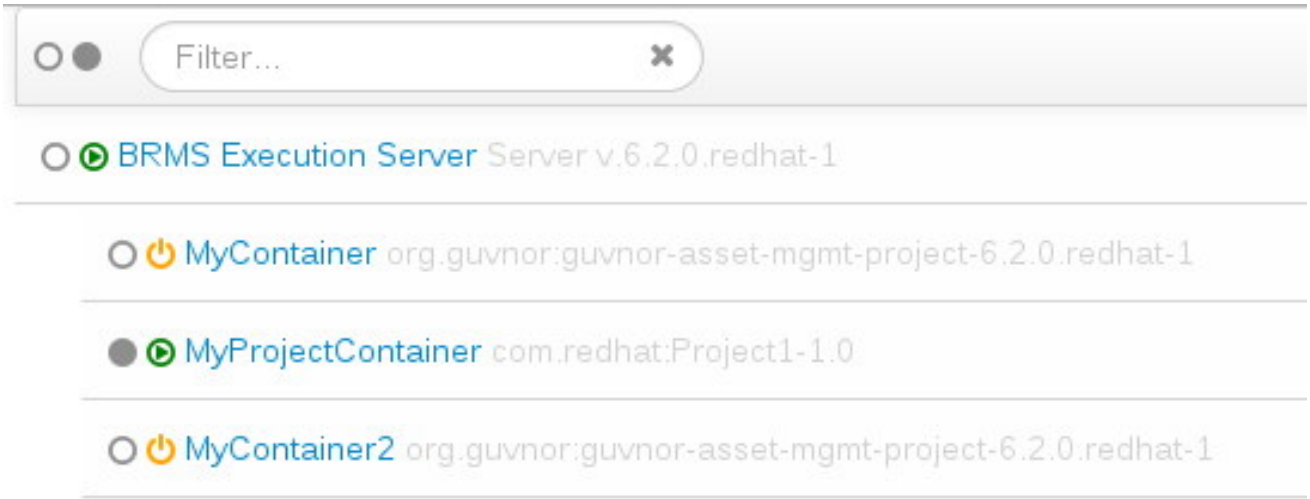
Containers within the Decision Server can be started, stopped, updated from within Business Central.

Starting a Container

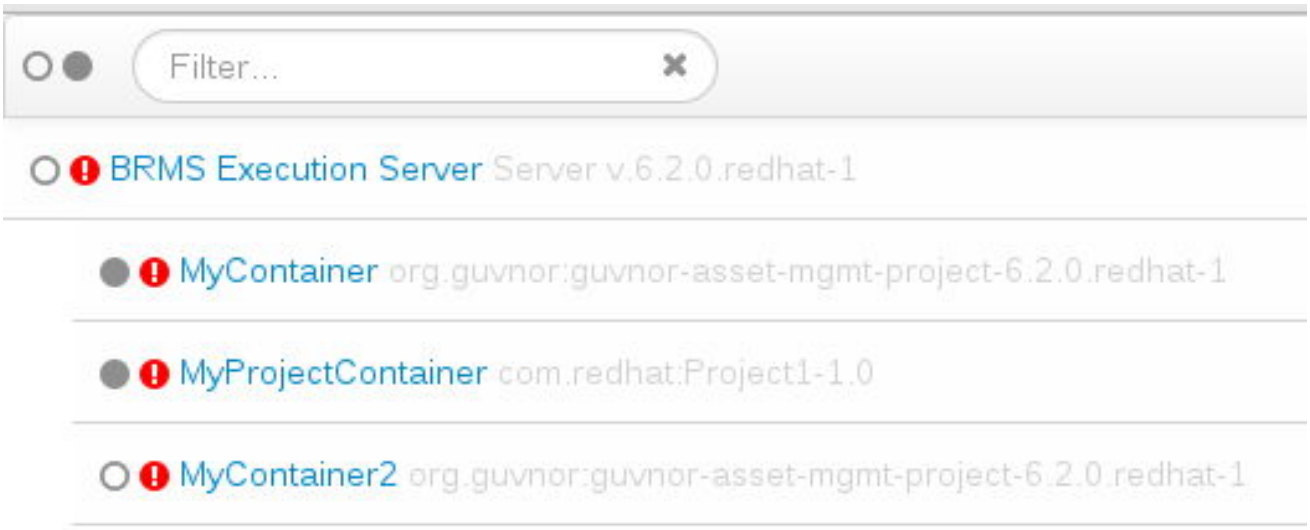
Once registered, a Container is in the 'Stopped' mode. It can be started by first selecting it and then clicking the **`Start`** button. You can of course, select multiple Containers and start them all at the same time.



Once the Container is in the 'Running' mode, a green arrow appears next to it.



If there are any errors starting the Container(s), red icons appear next to Containers and the Decision Server that they are deployed on. You should check the logs of both the Decision Server and the current Business Central to see what the errors are before redeploying the Containers (and possibly the Decision Server).



Stopping and Deleting a Container

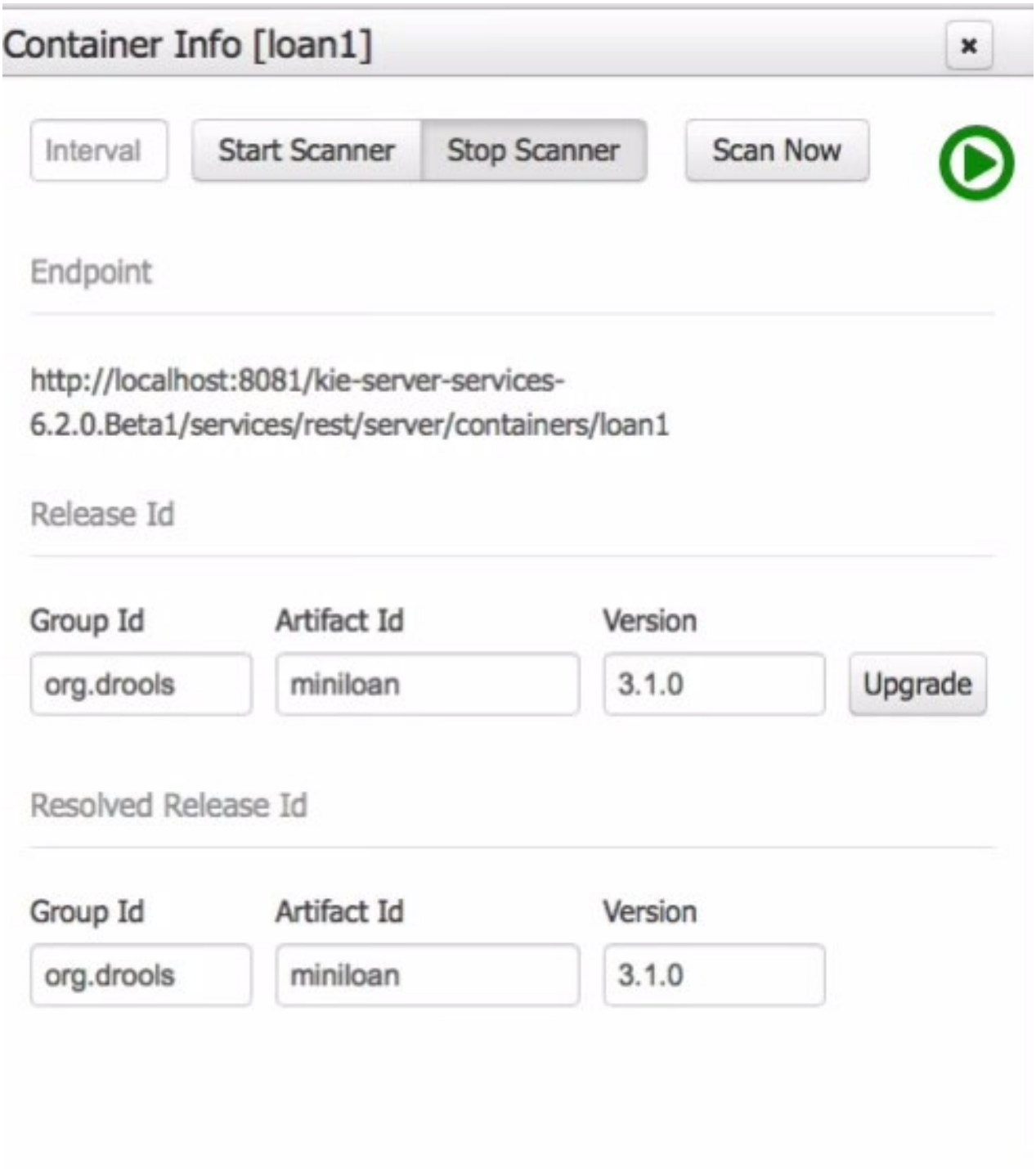
Similar to starting a Container, select the Container(s) that you want to stop (or delete) and click the **`Stop`** button (which replaces the **`Start`** button for that Container once it has entered the 'Running' mode) or the **`Delete`** button.

Updating Container Deployments

You can update deployed Containers without needing to restart the Decision Server. This is useful in cases where the Business rule changes cause new versions of packages to be provisioned.

You can of course, have multiple versions of the same package provisioned and deployed.

To update deployments in a Container dynamically, click on the  next to the Container. This will open up the Container Info screen. An example of this screen is shown here:

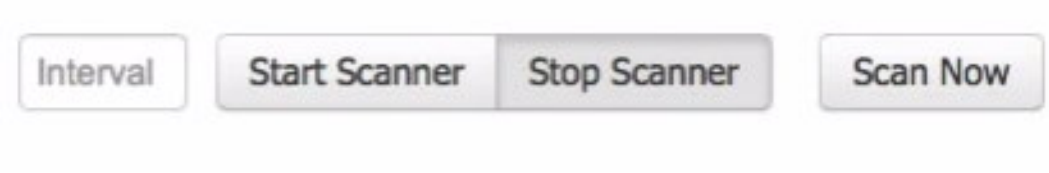


The Container Info screen is a useful tool because it not only allows you to see the endpoint for this provision, but it also allows you to either manually or automatically refresh the provision if an update is available.

1. Manual Update: To manually update a provision, enter the new Version number in the Version box and click on **`Update`** button. You can of course, update the Group Id or the Artifact Id, if these have changed as well. Once updated, the Decision server updates the provision and shows you the resolved GAV attributes at the bottom of the screen in the **`Resolved Release Id`** section.

2. Automatic Update: If you want a deployed Container to always have the latest version of your deployment without manually editing it, you will need to set the Version property to the value of **`LATEST`** and starting a Scanner. This will ensure that the deployed provision always contains the

latest version. The Scanner can be started just once on demand by clicking the **Scan Now** button or you can start it in the background with scans happening at a specified interval (in seconds).



You can also set this value to **LATEST** when you are first creating this deployment.

The **Resolved Release Id** in this case will show you the actual, latest version number.

10.5. THE REST API FOR MANAGING THE REALTIME DECISION SERVER

The Decision Server supports the following commands via the REST API. Note the following before using these commands:

1. The base URL for these will remain as the endpoint defined earlier (for example: *http://SERVER:PORT/kie-server/services/rest/server/*)

2. All requests require basic HTTP Authentication for the role kie-server as indicated earlier.

[GET] /

returns the Decision Server information - [GET]

Example 10.1. Example Server Response

```
<response type="SUCCESS" msg="Kie Server info">
  <kie-server-info>
    <version>6.2.0.redhat-1</version>
  </kie-server-info>
</response>
```

[POST] /

Using POST HTTP method, you can execute various commands on the Decision Server: **create-container, list-containers, dispose-container and call-container**.

[GET] **/containers**

Returns a list of containers that have been created on this Decision Server

Example 10.2. Example list of created Containers

```
<response type="SUCCESS" msg="List of created containers">
  <kie-containers>
    <kie-container container-id="MyProjectContainer"
status="STARTED">
      <release-id>
        <artifact-id>Project1</artifact-id>
```

```
        <group-id>com.redhat</group-id>
        <version>1.0</version>
      </release-id>
      <resolved-release-id>
        <artifact-id>Project1</artifact-id>
        <group-id>com.redhat</group-id>
        <version>1.0</version>
      </resolved-release-id>
    </kie-container>
  </kie-containers>
</response>
```

[GET] /containers/{id}

Returns the status and information about a particular Container. For example, executing http://SERVER:PORT/kie-server/services/rest/server/containers/MyProjectContainer will return the following example Container info.

Example 10.3. Example Container Info

```
<response type="SUCCESS" msg="Info for container MyProjectContainer">
  <kie-container container-id="MyProjectContainer" status="STARTED">
    <release-id>
      <artifact-id>Project1</artifact-id>
      <group-id>com.redhat</group-id>
      <version>1.0</version>
    </release-id>
    <resolved-release-id>
      <artifact-id>Project1</artifact-id>
      <group-id>com.redhat</group-id>
      <version>1.0</version>
    </resolved-release-id>
  </kie-container>
</response>
```

[PUT] /containers/{id}

Allows you to create a new Container in the Decision Server. For example, to create a Container with the id of MyRESTContainer the complete endpoint will be: http://SERVER:PORT/kie-server/services/rest/server/containers/MyRESTContainer with the following data sent:

Example 10.4. Example data for creating a Container

```
<kie-container container-id="MyRESTContainer" status="RUNNING">
<release-id>
  <group-id>com.redhat</group-id>
  <artifact-id>Project1</artifact-id>
  <version>1.1</version>
</release-id>
<resolved-release-id>
  <group-id>com.redhat</group-id>
  <artifact-id>Project1</artifact-id>
```

```
  <version>1.1</version>
</resolved-release-id>
</kie-container>
```

And the response from the server, if successful, will be:

Example 10.5. Example response after creating a Container

```
<response type="SUCCESS" msg="Container MyRESTContainer successfully
deployed with module com.redhat:Project1:1.1.">
   <kie-container container-id="MyRESTContainer" status="STARTED">
      <release-id>
         <artifact-id>Project1</artifact-id>
         <group-id>com.redhat</group-id>
         <version>1.1</version>
      </release-id>
      <resolved-release-id>
         <artifact-id>Project1</artifact-id>
         <group-id>com.redhat</group-id>
         <version>1.1</version>
      </resolved-release-id>
   </kie-container>
</response>
```

[DELETE] /containers/{id}

Stops (but does not delete) the Container specified by the id. For example, executing http://SERVER:PORT/kie-server/services/rest/server/containers/MyProjectContainer using the DELETE HTTP method will give the following server response:

Example 10.6. Example response of deleting a Container

```
<response type="SUCCESS" msg="Container MyProjectContainer
successfully disposed."/>
```

[POST] /containers/{id}

Executes operations and commands against the specified Container. You can send commands to this Container in the body of the POST request. For example, to fire all rules for Container with id MyRESTContainer (http://SERVER:PORT/kie-server/services/rest/server/containers/MyRESTContainer), you would send the fire-all-rules command to it as shown below (in the body of the POST request):

Example 10.7. Example POST XML Data for firing all rules against a Container

```
<fire-all-rules/>
```

Any valid command (of the type org.kie.api.Command) with correct attributes can be sent.

NOTE

For the list supported JBoss BRMS Commands, refer the Red Hat JBoss BPM Suite Development Guide. Except for the following commands, all the listed commands can be send to endpoint ***[POST] /containers/{id}***:

- **StartProcessCommand**
- **SignalEventCommand**
- **CompleteWorkItemCommand**
- **AbortWorkItemCommand**

[GET] /containers/{id}/release-id

Returns the full release id for the Container specified by the id.

Example 10.8. Example release id response

```
<response type="SUCCESS" msg="ReleaseId for container
MyProjectContainer">
  <release-id>
    <artifact-id>Project1</artifact-id>
    <group-id>com.redhat</group-id>
    <version>1.0</version>
  </release-id>
</response>
```

[POST] /containers/{id}/release-id

Allows you to update the release id of the Container deployment. Send the new complete release id to the Server.

Example 10.9. Example release id POST

```
<release-id>
  <group-id>com.redhat</group-id>
  <artifact-id>Project1</artifact-id>
  <version>1.1</version>
</release-id>
```

The Server will respond with a success or error message, similar to the one below:

Example 10.10. Example release id update response

```
<response type="SUCCESS" msg="Release id successfully updated.">
   <release-id>
      <artifact-id>Project1</artifact-id>
      <group-id>com.redhat</group-id>
```

```
      <version>1.1</version>
   </release-id>
</response>
```

[GET] /containers/{id}/scanner

Returns information about the scanner for this Container's automatic updates.

Example 10.11. Example scanner info response

```
<response type="SUCCESS" msg="Scanner info successfully retrieved">
<kie-scanner status="DISPOSED"/>
</response>
```

[POST] /containers/{id}/scanner

Allows you to start or stop a scanner that controls polling for updated Container deployments. To start the scanner, send a request similar to: http://SERVER:PORT/kie-server/services/rest/server/containers/{container-id}/scanner with the following POST data.

Example 10.12. Example scanner POST data to start

```
<kie-scanner status="STARTED" poll-interval="20">
</kie-scanner>
```

The poll-interval attribute is in seconds. The response from the server will be similar to:

Example 10.13. Example scanner POST response

```
<response type="SUCCESS" msg="Kie scanner successfully created.">
   <kie-scanner status="STARTED"/>
</response>
```

To stop the Scanner, replace the status with **`DISPOSED`** and remove the poll-interval attribute.

CHAPTER 11. REST API

Representational State Transfer (REST) is a style of software architecture of distributed systems (applications). It allows for a highly abstract client-server communication: clients initiate requests to servers to a particular URL with parameters if needed and servers process the requests and return appropriate responses based on the requested URL. The requests and responses are built around the transfer of representations of resources. A resource can be any coherent and meaningful concept that may be addressed (such as a repository, a Process, a Rule, etc.).

Red Hat JBoss BPM Suite and Red Hat JBoss BRMS provide REST API for individual application components. The REST API implementations differ slightly:

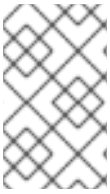
- Knowledge Store (Artifact Repository) REST API calls are calls to the static data (definitions) and are asynchronous, that is, they continue running after the call as a job. These calls return a job ID, which can be used after the REST API call was performed to request the job status and verify whether the job finished successfully. Parameters of these calls are provided in the form of JSON entities.

  The following two API's are only available in Red Hat JBoss BPM Suite.

- Deployment REST API calls are asynchronous or synchronous, depending on the operation performed. These calls perform actions on the deployments or retrieve information about one ore more deployments.

- Runtime REST API calls are calls to the Execution Server and to the Process Execution Engine, Task Execution Engine, and Business Rule Engine. They are synchronous and return the requested data as JAXB objects.

All REST API calls use the following URL with the request body:
`http://SERVER_ADDRESS:PORT/business-central/rest/REQUEST_BODY`



NOTE

Note that it is not possible to issue REST API calls over project resources, such as, rules files, work item definitions, process definition files, etc. are not supported. Perform operation over such files with Git and its REST API directly.

11.1. KNOWLEDGE STORE REST API

REST API calls to Knowledge Store allow you to manage the Knowledge Store content and manipulate the static data in the repositories of the Knowledge Store.

The calls are asynchronous; that is, they continue their execution after the call was performed as a job. All **`POST`** and **`DELETE`** return details of the request as a well as a job id that can be used to request the job status and verify whether the job finished successfully. The **`GET`** operations return information about repositories, projects and organizational units.

Parameters and results of these calls are provided in the form of JSON entities.

11.1.1. Job calls

Most Knowledge Store REST calls return a job ID after it is sent. This is necessary as the calls are asynchronous and you need to be able to reference the job to check its status as it goes through its lifecycle. During its lifecycle, a job can have the following statuses:

- ACCEPTED: the job was accepted and is being processed.
- BAD_REQUEST: the request was not accepted as it contained incorrect content.
- RESOURCE_NOT_EXIST: the requested resource (path) does not exist.
- DUPLICATE_RESOURCE: the resource already exists.
- SERVER_ERROR: an error on the server occurred.
- SUCCESS: the job finished successfully.
- FAIL: the job failed.
- APPROVED: the job was approved.
- DENIED: the job was denied.
- GONE: the job ID could not be found.

  A job can be GONE in the following cases:

  - The job was explicitly removed.
  - The job finished and has been deleted from the status cache (the job is removed from status cache after the cache has reached its maximum capacity).
  - The job never existed.

The following **`job`** calls are provided:

[GET] /jobs/{jobID}

returns the job status - [GET]

> **Example 11.1. Response of the job call on a repository clone request**
>
> ```
> "{"status":"SUCCESS","jodId":"1377770574783-27","result":"Alias:
> testInstallAndDeployProject, Scheme: git, Uri:
> git://testInstallAndDeployProject","lastModified":1377770578194,"deta
> iledResult":null}"
> ```

[DELETE] /jobs/{jobID}

removes the job - [DELETE]

11.1.2. Repository calls

Repository calls are calls to the Knowledge Store that allow you to manage its Git repositories and their projects.

The following **`repositories`** calls are provided:

[GET] /repositories

This returns a list of the repositories in the Knowledge Store as a JSON entity - [GET]

Example 11.2. Response of the repositories call

```
[{"name":"brms-assets","description":"generic
assets","userName":null,"password":null,"requestType":null,"gitURL":"
git://brms-assets"},{"name":"loanProject","description":"Loan
processes and
rules","userName":null,"password":null,"requestType":null,"gitURL":"g
it://loansProject"}]
```

[GET] /repositories/{repositoryName}

This returns information on a specific repository - [GET]

[DELETE] /repositories/{repositoryName}

This deletes the repository - [DELETE]

[POST] /repositories/

This creates or clones the repository defined by the JSON entity - [POST]

Example 11.3. JSON entity with repository details of a repository to be cloned

```
{"name":"myClonedRepository", "description":"", "userName":"",
"password":"", "requestType":"clone",
"gitURL":"git://localhost/example-repository"}
```

[GET] /repositories/{repositoryName}/projects/

This returns a list of the projects in a specific repository as a JSON entity - [POST]

Example 11.4. JSON entity with details of existing projects

```
[ {
  "name" : "my-project-name",
  "description" : "Project to illustrate REST output",
  "groupId" : "com.acme",
  "version" : "1.0"
}, {
  "name" : "yet-another-project-name",
  "description" : "Yet Another Project to illustrate REST output",
  "groupId" : "com.acme",
  "version" : "2.2.1"
} ]
```

[POST] /repositories/{repositoryName}/projects/

This creates a project in the repository - [POST]

Example 11.5. Request body that defines the project to be created

```
"{"name":"myProject","description": "my project"}"
```

[DELETE] /repositories/{repositoryName}/projects/

This deletes the project in the repository - [DELETE]

Example 11.6. Request body that defines the project to be deleted

```
"{"name":"myProject","description": "my project"}"
```

11.1.3. Organizational unit calls

Organizational unit calls are calls to the Knowledge Store that allow you to manage its organizational units.

The following **`organizationalUnits`** calls are provided:

[GET] /organizationalunits/

This returns a list of all the organizational units - [GET].

Example 11.7. Organizational unit list in JSON

```
[ {
  "name" : "EmployeeWage",
  "description" : null,
  "owner" : "Employee",
  "defaultGroupId" : "org.bpms",
  "repositories" : [ "EmployeeRepo", "OtherRepo" ]
}, {
  "name" : "OrgUnitName",
  "description" : null,
  "owner" : "OrgUnitOwner",
  "defaultGroupId" : "org.group.id",
  "repositories" : [ "repository-name-1", "repository-name-2" ]
} ]
```

[GET] /organizationalunits/{organizationalUnitName}

This returns a JSON entity with info about a specific organizational unit - [GET].

[POST] /organizationalunits/

This creates an organizational unit in the Knowledge Store - [POST]. The organizational unit is defined as a JSON entity. This consumes an **`OrganizationalUnit`** instance and returns a **`CreateOrganizationalUnitRequest`** instance.

Example 11.8. Organizational unit in JSON

```
{
```

```
  "name":"testgroup",
  "description":"",
  "owner":"tester",
  "repositories":["testGroupRepository"]
}
```

[POST] /organizationalunits/{organizationalUnitName}

This *updates* the details of an existing organizational unit - [POST].

Both the **name** and **owner** fields in the consumed **UpdateOrganizationalUnit** instance can be left empty. Both the **description** field and the repository association can *not* be updated via this operation.

Example 11.9. Update organizational unit input in JSON

```
{
  "owner" : "NewOwner",
  "defaultGroupId" : "org.new.default.group.id"
}
```

[DELETE] /organizationalunits/{organizationalUnitName}

This deletes an organizational unit - [GET].

[POST] /organizationalunits/{organizationalUnitName}/repositories/{repositoryName}

This adds the repository to the organizational unit - [POST].

[DELETE] /organizationalunits/{organizationalUnitName}/repositories/{repositoryName}

This removes a repository from the organizational unit - [POST].

11.1.4. Maven calls

Maven calls are calls to a Project in the Knowledge Store that allow you to compile and deploy the Project resources.

The following **maven** calls are provided below:

[POST] /repositories/{repositoryName}/projects/{projectName}/maven/compile/

This compiles the project (equivalent to **mvn compile**) - [POST]. It consumes a **BuildConfig** instance, which must be supplied but is not needed for the operation and may be left blank. It also returns a **CompileProjectRequest** instance.

[POST] /repositories/{repositoryName}/projects/{projectName}/maven/install/

This installs the project (equivalent to **mvn install**) - [POST]. It consumes a **BuildConfig** instance, which must be supplied but is not needed for the operation and may be left blank. It also returns a **InstallProjectRequest** instance.

[POST] /repositories/{repositoryName}/projects/{projectName}/maven/test/

This compiles and runs the tests - [POST]. It consumes a **`BuildConfig`** instance and returns a **`TestProjectRequest`** instance.

[POST] /repositories/{repositoryName}/projects/{projectName}/maven/deploy/

This deploys the project (equivalent to mvn deploy) - [POST]. It consumes a **`BuildConfig`** instance, which must be supplied but is not needed for the operation and may be left blank. It also returns a **`DeployProjectRequest`** instance.

11.2. REST SUMMARY

The URL templates in the table below are relative to the following URL:

- **`http://server:port/business-central/rest`**

Table 11.1. Knowledge Store REST calls

URL Template	Type	Description
/jobs/{jobID}	GET	return the job status
/jobs/{jobID}	DELETE	remove the job
/organizationalunits	GET	return a list of organizational units
/organizationalunits	POST	create an organizational unit in the Knowledge Store described by the JSON **`OrganizationalUnit`** entity
/organizationalunits/{organizationalUnitName}/repositories/{repositoryName}	POST	add a repository to an organizational unit
/repositories/	POST	add the repository to the organizational unit described by the JSON **`RepositoryReqest`** entity
/repositories	GET	return the repositories in the Knowledge Store
/repositories/{repositoryName}	DELETE	remove the repository from the Knowledge Store
/repositories/	POST	create or clone the repository defined by the JSON **`RepositoryRequest`** entity

URL Template	Type	Description
/repositories/{repositoryName}/projects/	POST	create the project defined by the JSON entity in the repository
/repositories/{repositoryName}/projects/{projectName}/maven/compile/	POST	compile the project
/repositories/{repositoryName}/projects/{projectName}/maven/install	POST	install the project
/repositories/{repositoryName}/projects/{projectName}/maven/test/	POST	compile the project and run tests as part of compilation
/repositories/{repositoryName}/projects/{projectName}/maven/deploy/	POST	deploy the project

APPENDIX A. REVISION HISTORY

Revision 1.0.0-22 **Thu Dec 17 2015** **Vidya Iyengar**

Build includes various enhancements and fixes