



Red Hat JBoss BRMS 6.2

User Guide

The User Guide for Red Hat JBoss BRMS

Red Hat JBoss BRMS 6.2 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

Copyright © 2015 Red Hat, Inc.

This document is licensed by Red Hat under the [Creative Commons Attribution-ShareAlike 3.0 Unported License](#). If you distribute this document, or a modified version of it, you must provide attribution to Red Hat, Inc. and provide a link to the original. If the document is modified, all Red Hat trademarks must be removed.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat Software Collections is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

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

Table of Contents

| | |
|-----------------------------------------------------------------------------------------------|-----------|
| CHAPTER 1. INTRODUCTION | 4 |
| 1.1. ABOUT RED HAT JBOSS BRMS | 4 |
| 1.2. USE CASE: BUSINESS DECISION MANAGEMENT IN THE INSURANCE INDUSTRY WITH RED HAT JBOSS BRMS | 4 |
| 1.3. ASSETS | 5 |
| CHAPTER 2. BUSINESS CENTRAL | 7 |
| 2.1. PERSPECTIVES | 7 |
| 2.2. LOGGING ON TO BUSINESS CENTRAL | 8 |
| 2.3. THE HOME SCREEN | 8 |
| 2.4. EMBEDDING BUSINESS CENTRAL | 9 |
| 2.5. PROJECT AUTHORIZING | 10 |
| 2.6. ASSET LOCKING SUPPORT | 14 |
| 2.7. PROJECT EDITOR | 15 |
| 2.8. ADMINISTRATION MENU | 20 |
| 2.9. RENAME, COPY, DELETE ASSETS | 20 |
| 2.10. DEPLOYMENT MENU: THE ARTIFACT REPOSITORY | 21 |
| CHAPTER 3. SETTING UP A NEW PROJECT | 22 |
| 3.1. CREATING AN ORGANIZATIONAL UNIT | 22 |
| 3.2. CREATING A REPOSITORY | 23 |
| 3.3. CLONING A REPOSITORY | 24 |
| 3.4. CREATING A PROJECT | 26 |
| 3.5. CREATING A NEW PACKAGE | 27 |
| 3.6. ADDING DEPENDENCIES | 28 |
| 3.7. DEFINING KIE BASES AND SESSIONS | 28 |
| 3.8. CREATING A RESOURCE | 30 |
| 3.9. SYSTEM PROPERTIES | 30 |
| CHAPTER 4. SOCIAL EVENTS | 33 |
| FOLLOW USER | 33 |
| ACTIVITY TIMELINE | 33 |
| CHAPTER 5. DATA MODELS | 34 |
| 5.1. DATA MODELER | 34 |
| 5.2. ANNOTATIONS IN DATA MODELER | 35 |
| 5.3. CREATING A DATA OBJECT (NOT PERSISTABLE) | 35 |
| 5.4. PERSISTABLE DATA OBJECTS | 36 |
| 5.5. DATA OBJECT DOMAIN SCREENS | 37 |
| 5.6. CONFIGURING RELATIONSHIPS BETWEEN DATA OBJECTS | 43 |
| 5.7. PERSISTENCE DESCRIPTOR | 44 |
| CHAPTER 6. WRITING RULES | 46 |
| 6.1. CREATING A RULE | 46 |
| 6.2. THE ASSET EDITOR | 46 |
| 6.3. DECISION TABLES | 51 |
| 6.4. WEB BASED GUIDED DECISION TABLES | 54 |
| 6.5. RULE TEMPLATES | 68 |
| 6.6. THE DOMAIN SPECIFIC LANGUAGE EDITOR | 77 |
| 6.7. DATA ENUMERATIONS | 78 |
| 6.8. SCORECARDS | 79 |
| 6.9. VERIFICATION AND VALIDATION OF GUIDED DECISION TABLES | 80 |

| | |
|-------------------------------------------------------------------------|------------|
| CHAPTER 7. BUILDING AND DEPLOYING ASSETS | 83 |
| CHAPTER 8. MANAGING ASSETS | 84 |
| 8.1. VERSIONS AND STORAGE | 84 |
| CHAPTER 9. TESTING | 85 |
| 9.1. TEST SCENARIOS | 85 |
| 9.2. CREATING A TEST SCENARIO | 85 |
| 9.3. ADDITIONAL TEST SCENARIO FEATURES | 88 |
| CHAPTER 10. THE REALTIME DECISION SERVER | 94 |
| 10.1. DEPLOYING THE REALTIME DECISION SERVER | 94 |
| 10.2. INSTALLING THE REALTIME DECISION SERVER IN OTHER CONTAINERS | 94 |
| 10.3. REALTIME DECISION SERVER SETUP | 95 |
| 10.4. CREATING A CONTAINER | 101 |
| 10.5. MANAGING CONTAINERS | 102 |
| 10.6. THE REST API FOR REALTIME DECISION SERVER EXECUTION | 104 |
| 10.7. THE REST API FOR REALTIME DECISION SERVER ADMINISTRATION | 121 |
| 10.8. REALTIME DECISION SERVER JAVA CLIENT API OVERVIEW | 128 |
| CHAPTER 11. REST API | 135 |
| 11.1. KNOWLEDGE STORE REST API | 135 |
| 11.2. REST SUMMARY | 140 |
| APPENDIX A. REVISION HISTORY | 142 |

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.

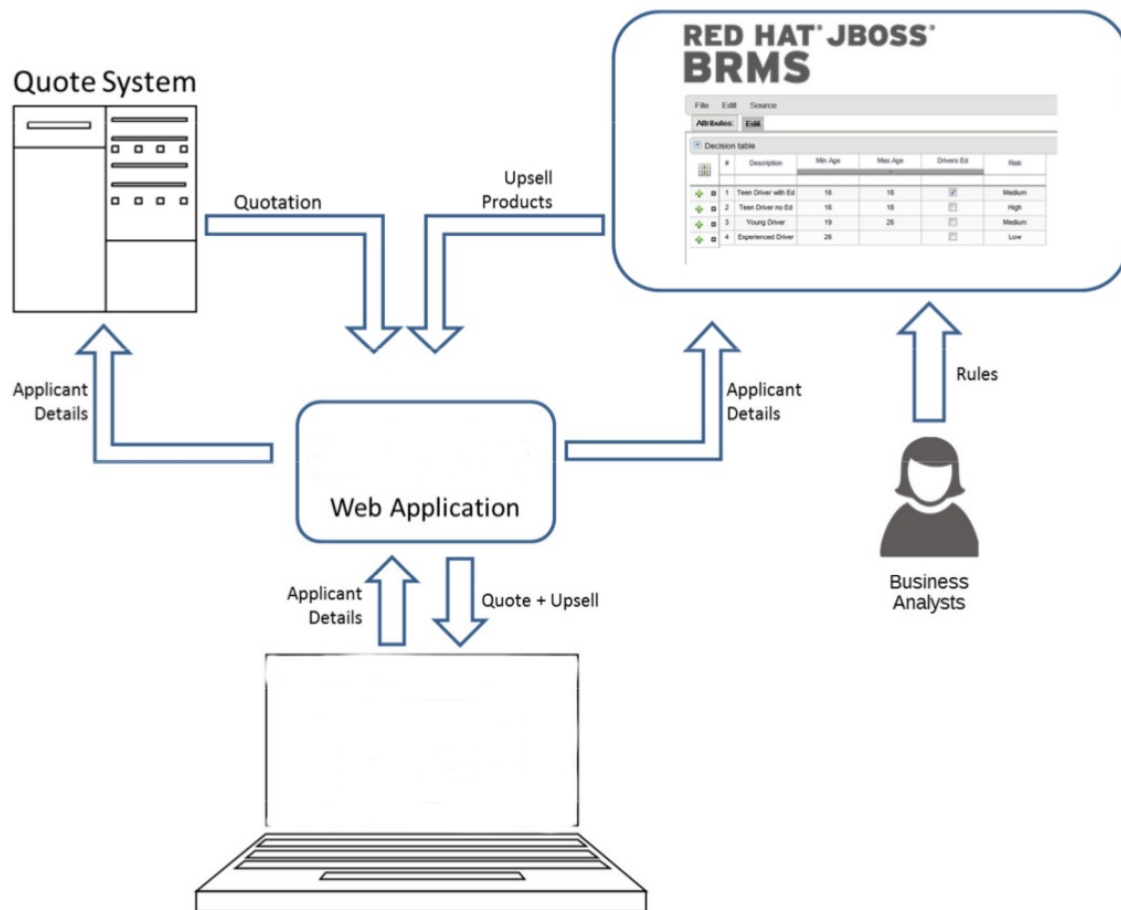


Figure 1.1. JBoss BRMS Use Case: Insurance Industry Decision Making

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 workflow 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:

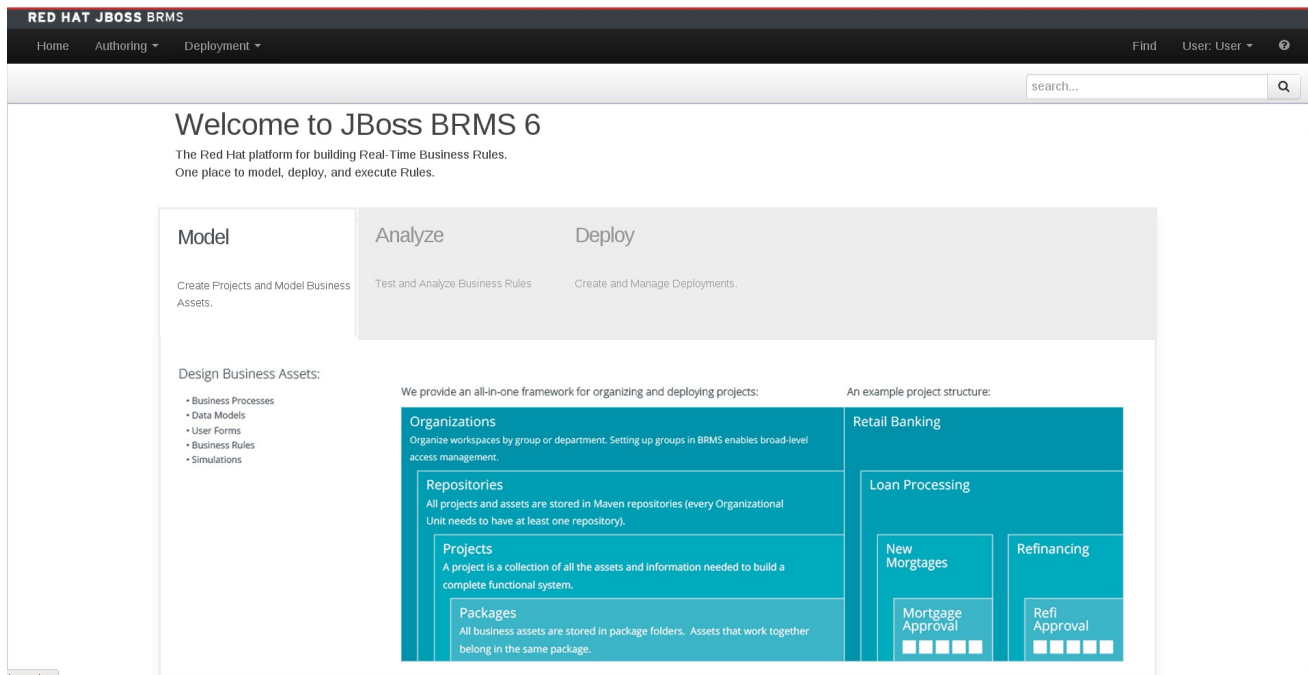


Figure 2.1. Business central home screen

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 exist. | 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 AUTHORIZING

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**.

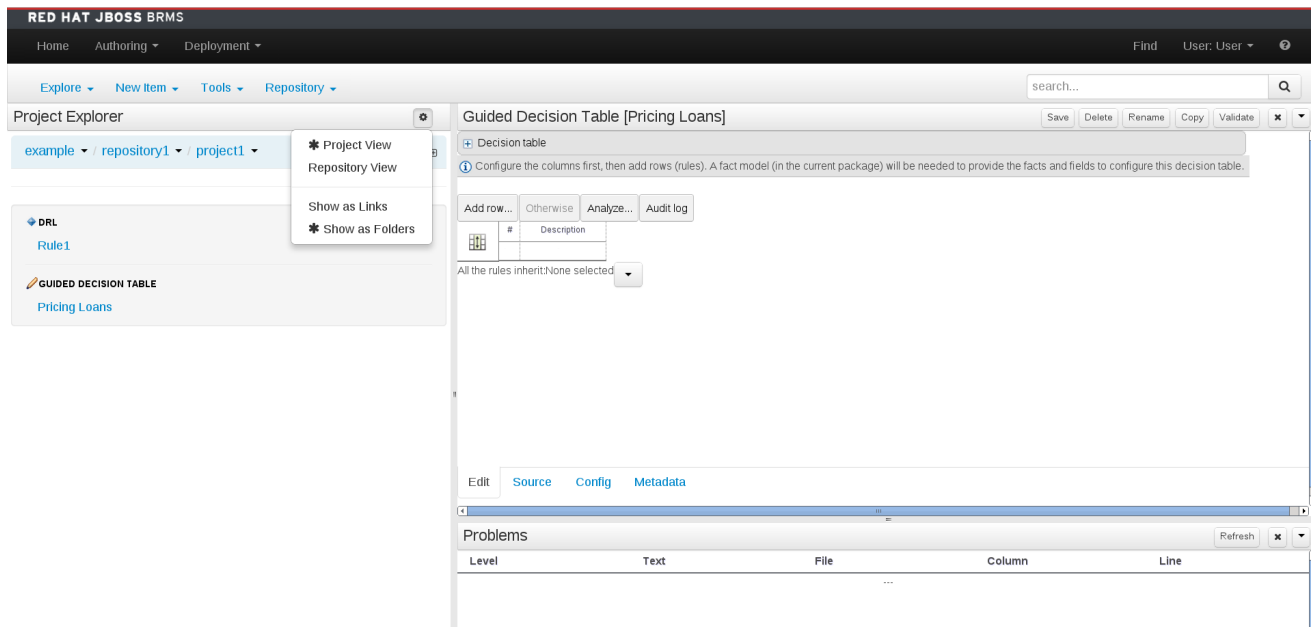



Figure 2.2. The Project Explorer screen

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:

- To resize the width of the screen:
 - Move the mouse pointer over the vertical panel splitter. The pointer changes to .
 - Adjust the width of the screen by dragging the splitter and setting it at the required position.
- To resize the height of the screen:
 - 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.

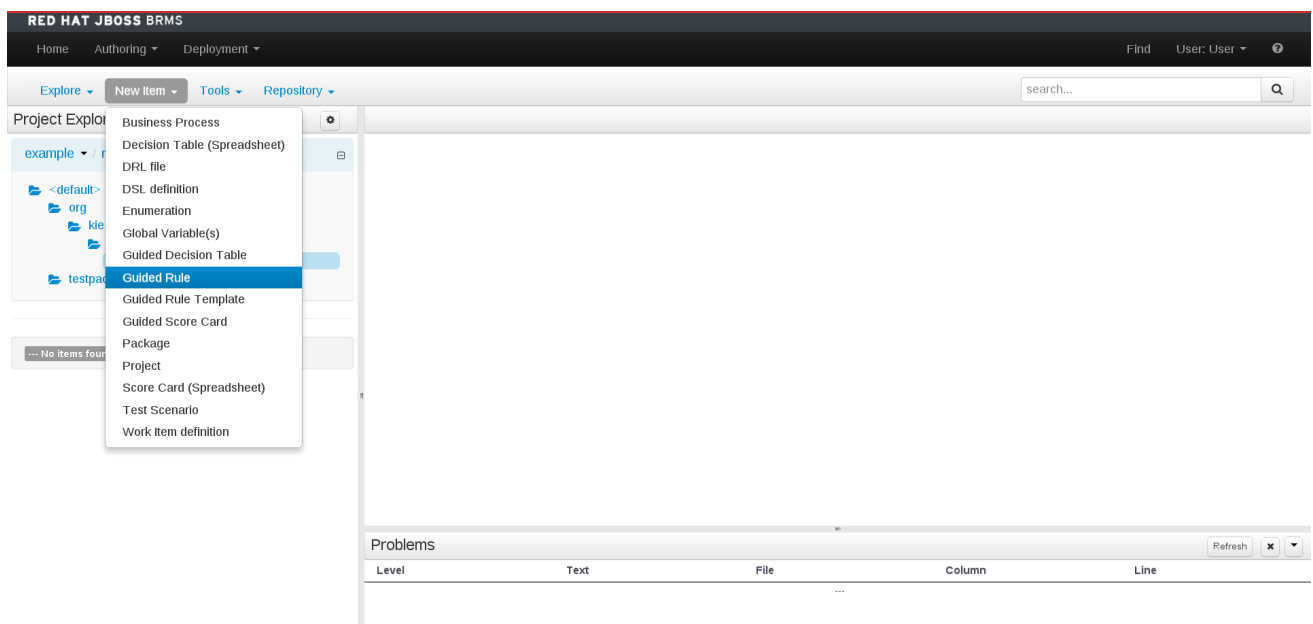
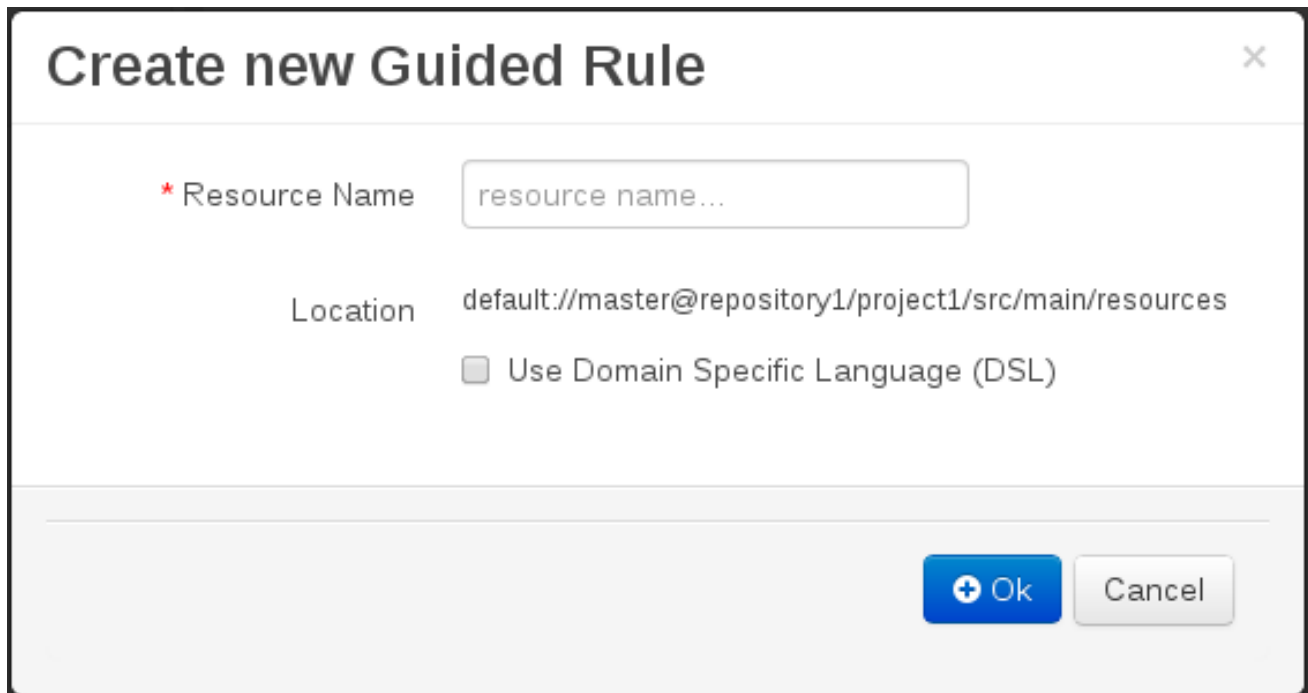


Figure 2.3. Creating new Asset screen

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.



Create new Guided Rule ✕

* Resource Name

Location

☐ Use Domain Specific Language (DSL)

+ Ok Cancel

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

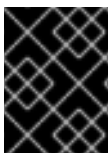
To open up the metadata screen for a DRL asset, click on the **Overview** tab. If an asset doesn't have an **Overview** tab, it means that there is no metadata associated with that asset.



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.



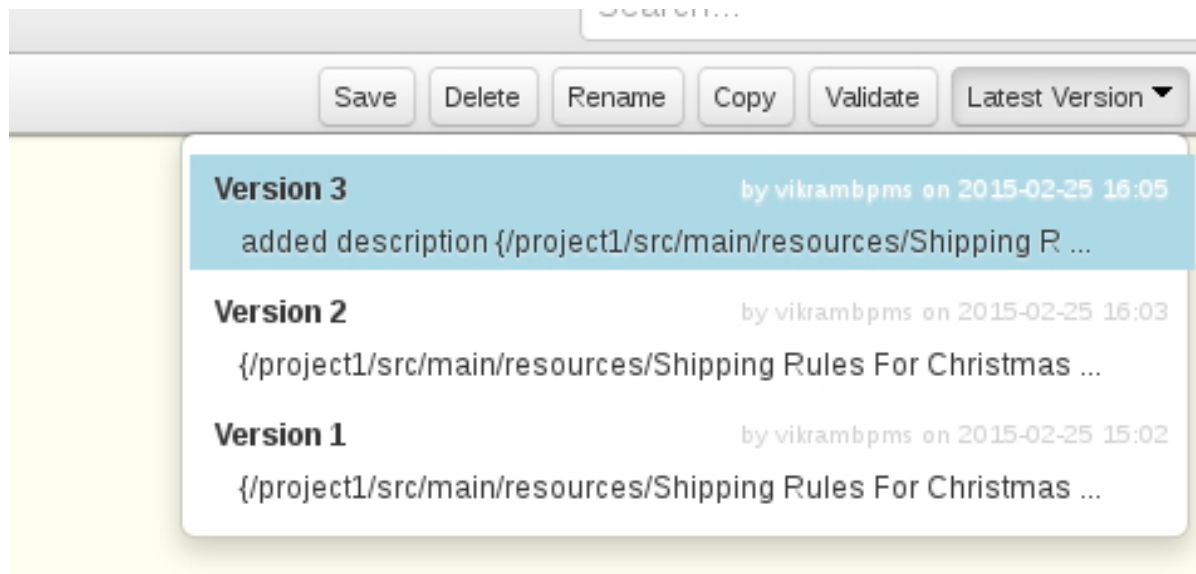
IMPORTANT

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. ASSET LOCKING SUPPORT

The default locking mechanism for locking a BPM and BRMS asset while updating it in Business Central is pessimistic. Whenever you open and modify an asset in Business Central, it automatically locks the asset for your exclusive use, in order to avoid conflicts in a multi-user setup. The pessimistic lock is automatically released when your session ends or when you save or close the asset.

The pessimistic lock feature is provided in order to help prevent users from overwriting each other's changes. However, there may be cases when you may want to edit a file locked by another user. Business Central allows you to force unlock a locked asset. To do this:

Procedure 2.1. Unlocking assets

1. Open the asset.
2. Click on the **Overview** tab and open up the **Metadata** screen.

If the asset is already being edited by another user, the following will be displayed in the **Lock status** field:

Locked by <user_name>

3. To edit the asset locked by another user, click **Force unlock** asset button.

The following confirmation popup message is displayed:

Are you sure you want to release the lock of this asset? This might cause <user_name> to lose unsaved changes!

4. Click Yes to confirm.

The asset goes back to unlocked state.

2.7. PROJECT EDITOR

2.7.1. 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.

2.7.2. 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.

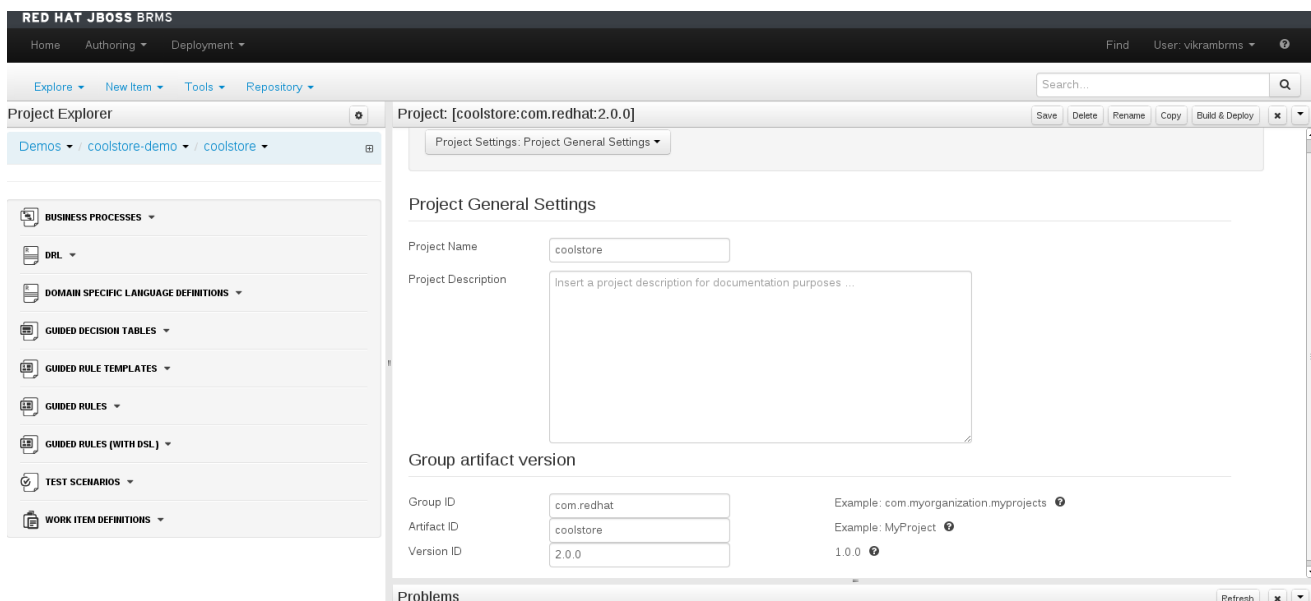


Figure 2.5. Project Editor - Project Settings

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.

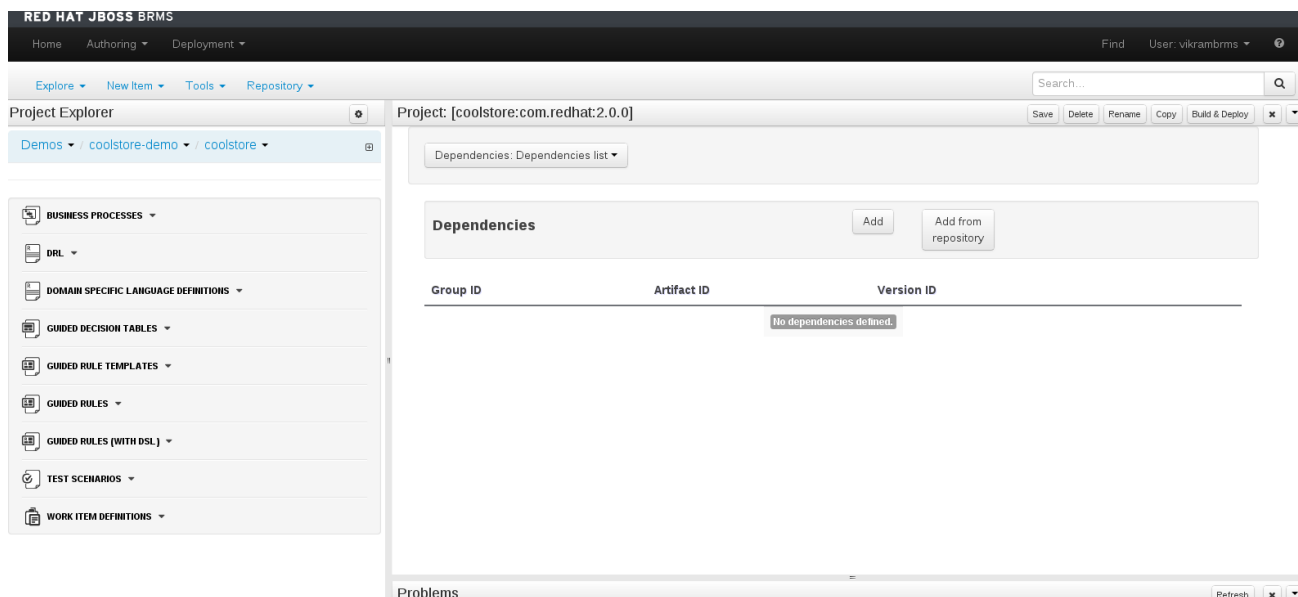



Figure 2.6. Project Editor - Project Dependencies

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.

2.7.3. 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.

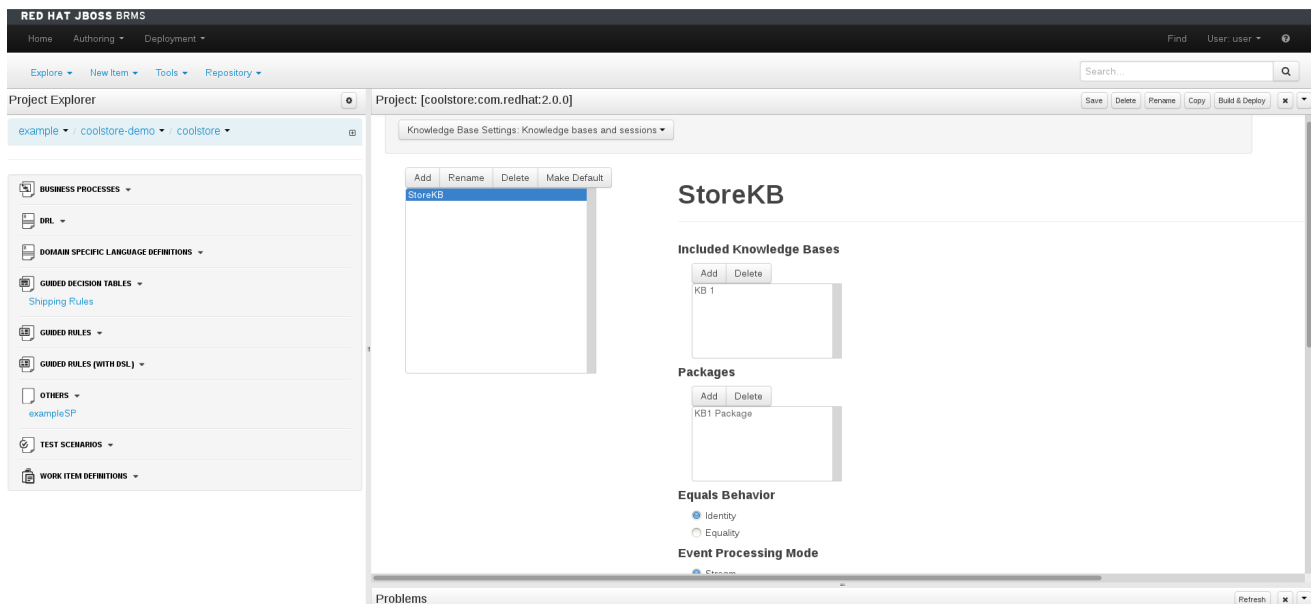


Figure 2.7. Project Editor - Knowledge Base Settings

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 2.7.2, “Project Settings”](#) for more information about Metadata.

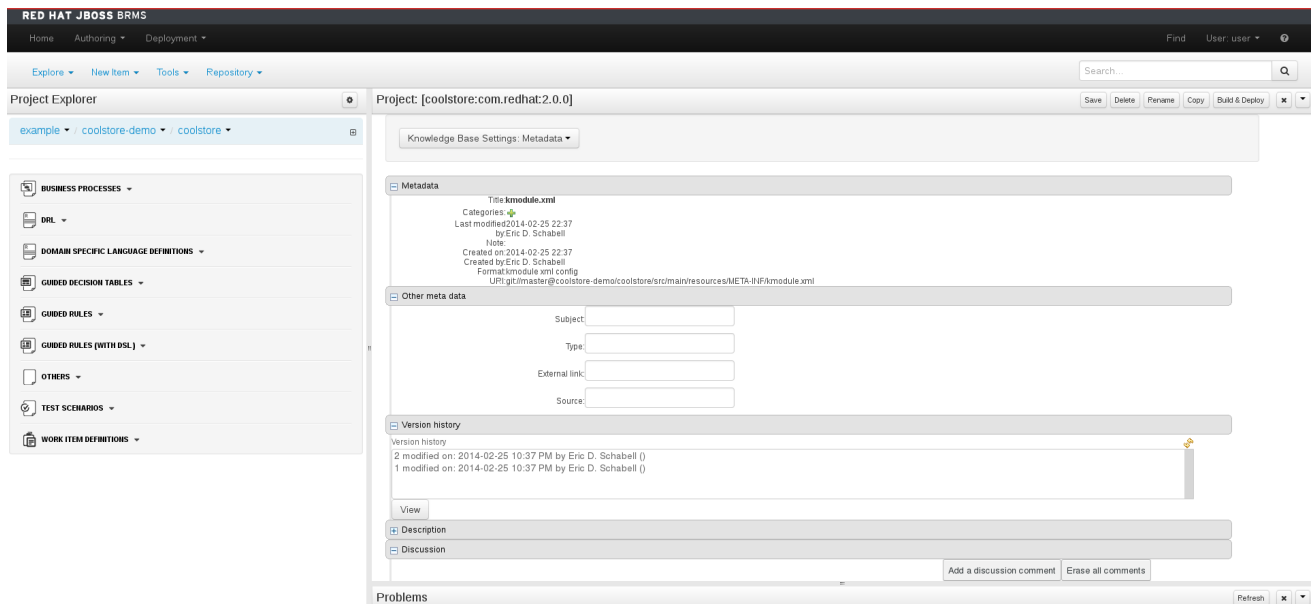


Figure 2.8. Knowledge Base Settings - Metadata

2.7.4. 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.

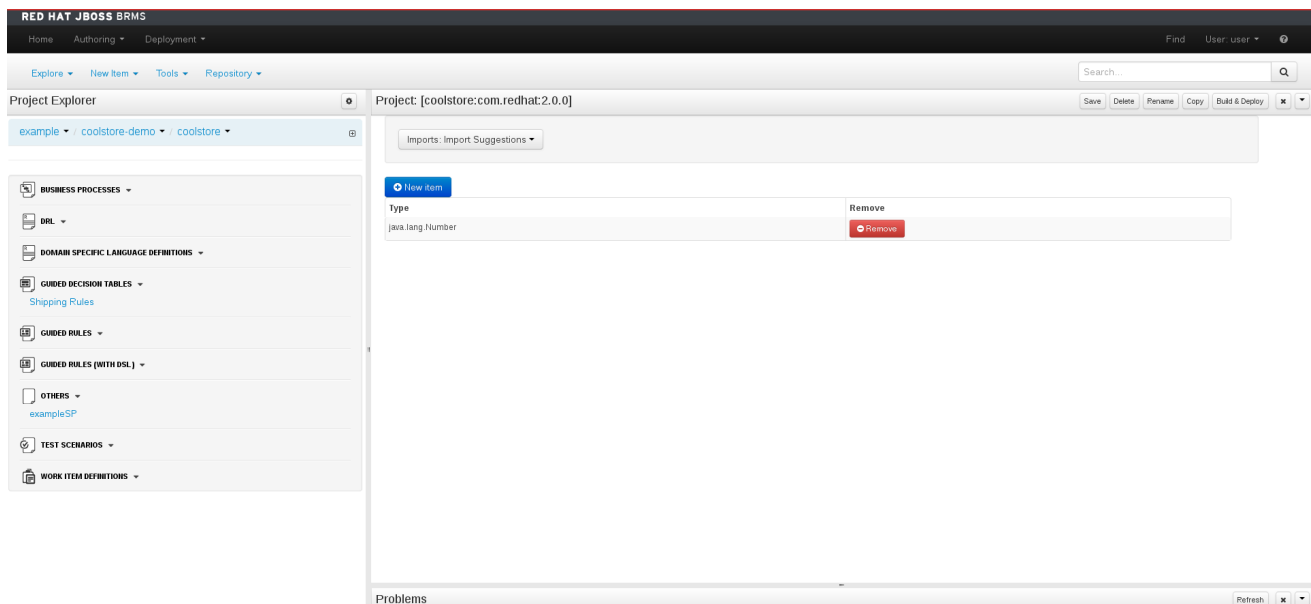



Figure 2.9. Project Editor - Imports

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 2.7.2, “Project Settings”](#) for more information about Metadata.

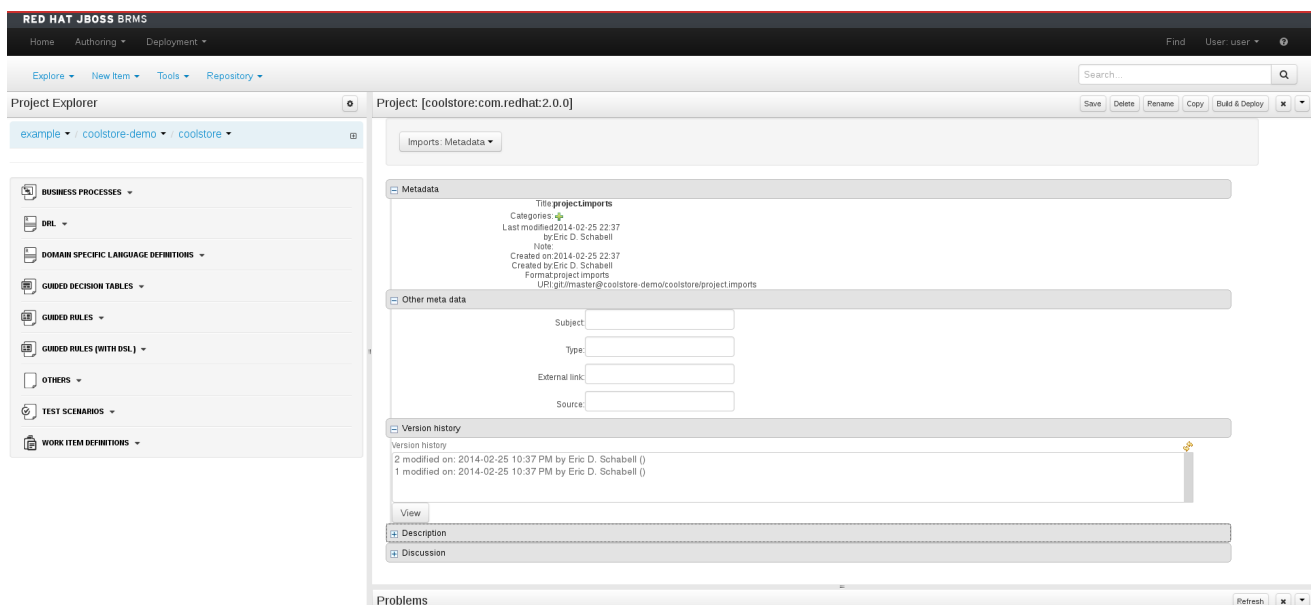


Figure 2.10. Imports - Metadata

2.8. 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](#).

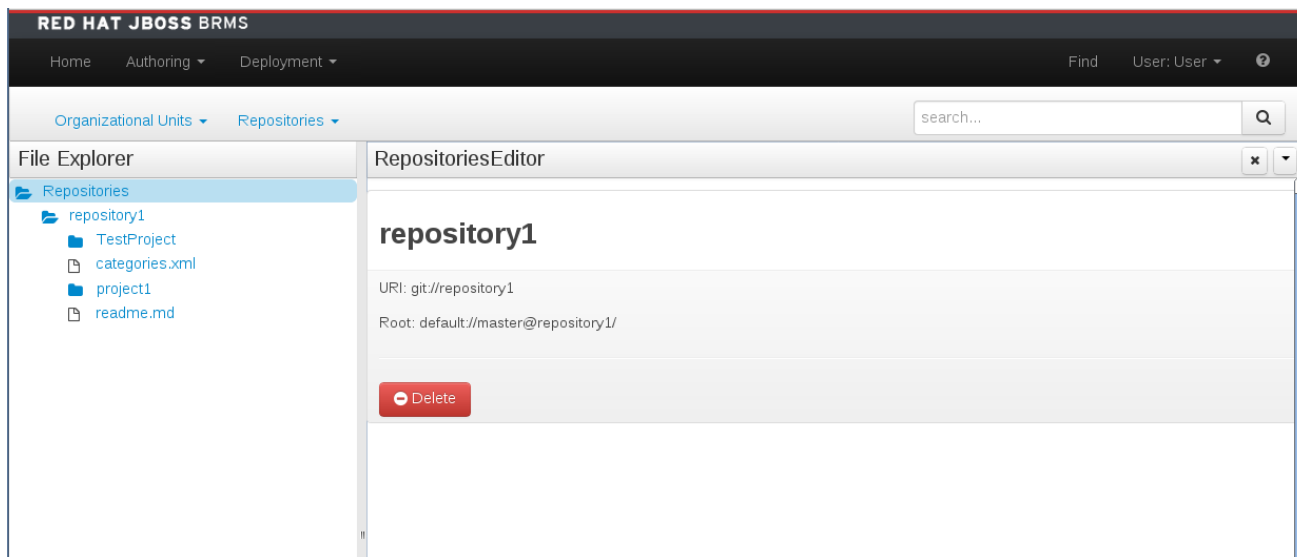



Figure 2.11. The Administration Screen


2.9. RENAME, COPY, DELETE ASSETS

2.9.1. 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.

2.9.2. 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.

2.9.3. 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.

2.10. 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.

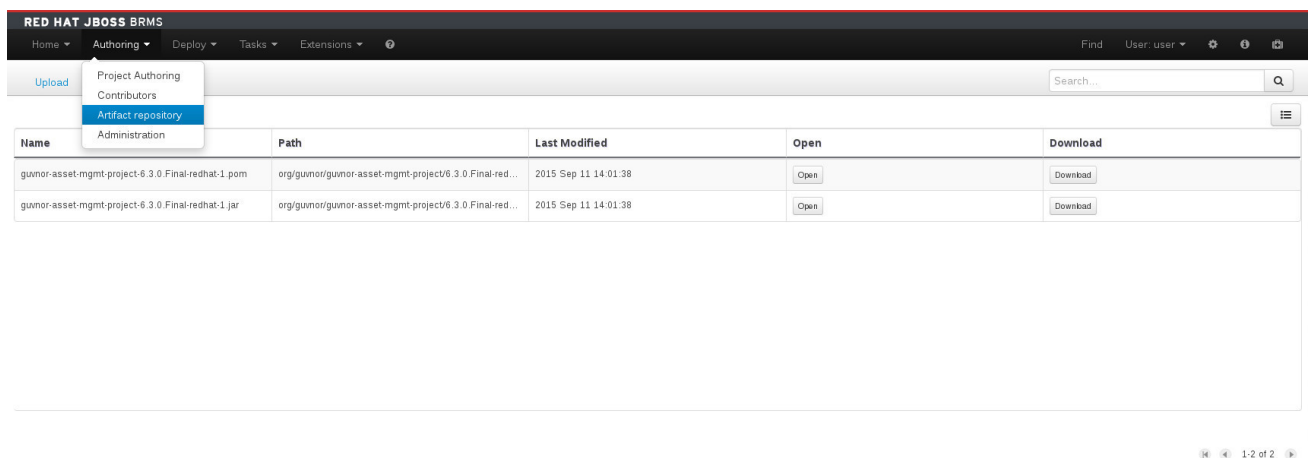


Figure 2.12. The Artifact Repository Screen

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.

Add New Organizational Unit [X]

Organizational Unit Information * is required

* Name
Organizational Unit name...

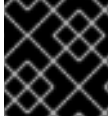
* Owner
Organizational Unit owner...

[+ OK] [Cancel]

Figure 3.1. Add New Organizational Unit Pop-up

4. Enter the mandatory details:
 - o Organizational unit name.
 - o 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.

New Repository [X]

Basic Settings

* Repository Name

* In Organizational Unit

< Previous Next > Cancel ☒ Finish

Figure 3.2. Create Repository Pop-up

4. Enter the mandatory details:
 - o Repository name.

**NOTE**

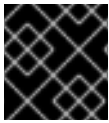
Note that the repository name should be a valid filename. Avoid using a space or any special character that might lead to an invalid folder name.

- 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.

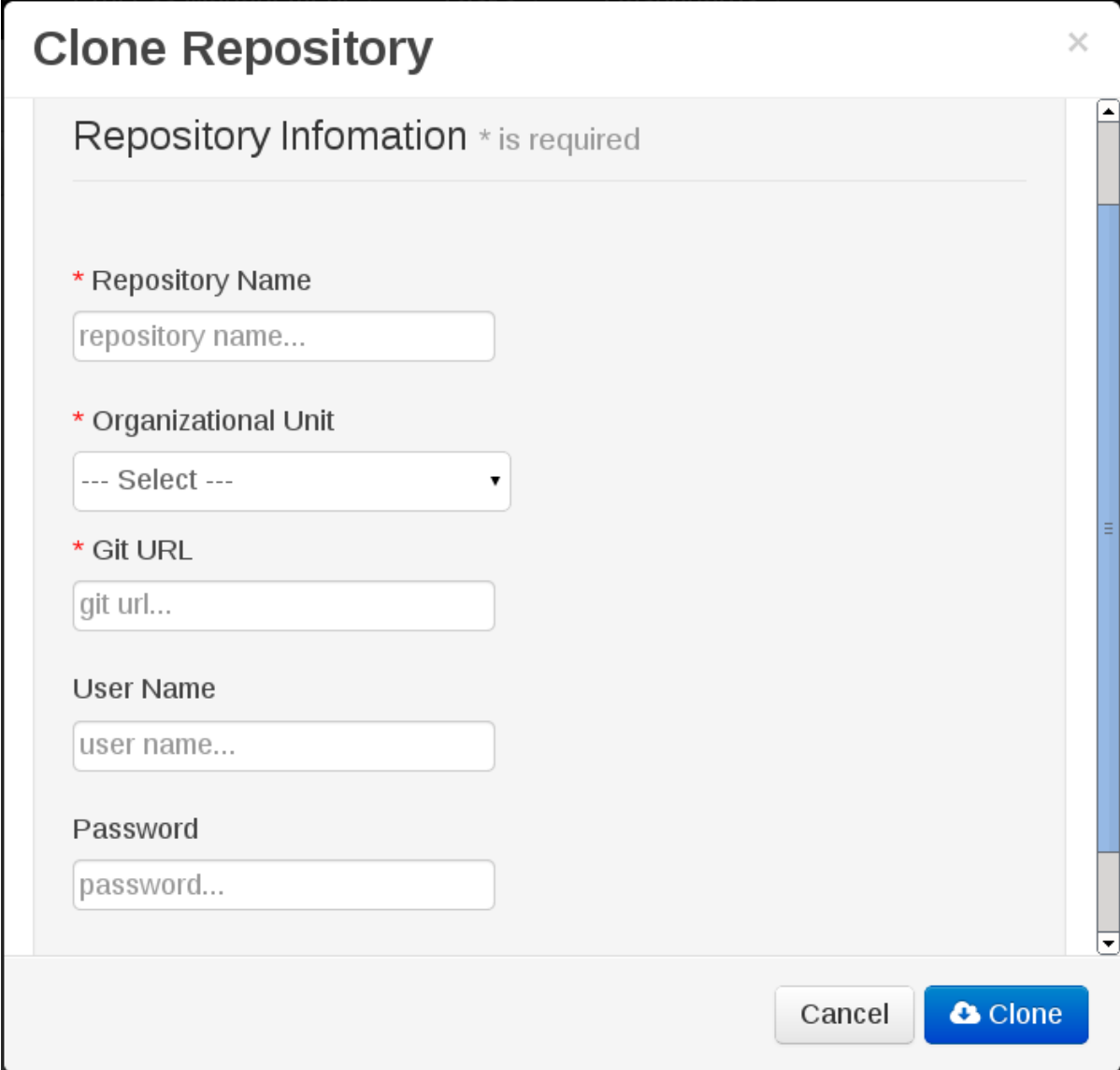
3.3. CLONING A REPOSITORY

**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.



The image shows a 'Clone Repository' dialog window. It has a title bar with a close button (X). The main area is titled 'Repository Information * is required'. It contains several input fields: a text field for 'Repository Name' with placeholder 'repository name...', a dropdown menu for 'Organizational Unit' with '--- Select ---', a text field for 'Git URL' with placeholder 'git url...', a text field for 'User Name' with placeholder 'user name...', and a text field for 'Password' with placeholder 'password...'. At the bottom right, there are two buttons: 'Cancel' and 'Clone' (which has a cloud icon).

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**.

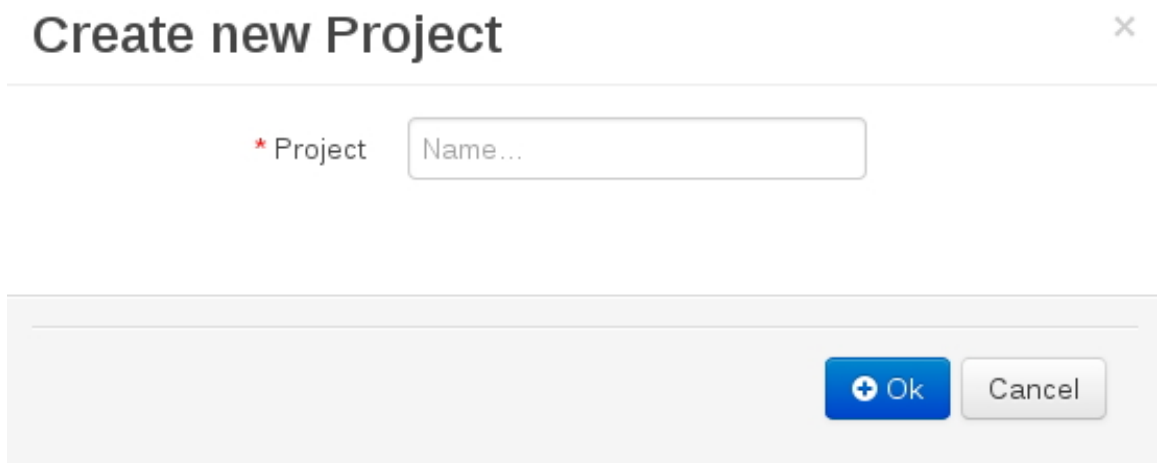
6. A confirmation prompt with an **OK** button is displayed which notifies the user that the repository is created successfully. Click **OK**. The repository will be indexed. Some workbench features may be unavailable until indexing has completed.

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.

- o 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):
 - o 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.
 - o 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 do so through Business Central or by editing `kmodule.xml` in 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 Business Central, do the following:

1. Click **Authoring** → **Project Authoring** and navigate to your project.
2. In the **Project Explorer** window, click **Open Project Editor**.
3. Click **Project Settings: Project General Settings** → **Knowledge bases and sessions**. This view provides a user interface for changing `kmodule.xml`.
4. Click **Add** to define and add your bases.
 - a. After you enter a name for your Knowledge Base, add Packages. For including all packages, click **Add** below **Packages** and enter asterisk `*`.
5. Below **Knowledge Sessions**, click **Add** and enter the name of your session.
6. Mark it **Default** and select appropriate state.

For Red Hat JBoss BRMS, you can choose between **stateful** and **stateless** sessions. Use **stateless** if you do not need iterative invocations of the facts. Otherwise, use **stateful** session.

7. Click **Save** in the top right corner once you are done.

Defining Kie Bases and Sessions in `kmodule.xml`

To define a Kie base or session by editing `kmodule.xml`, do the following:

1. Open the repository view for your project.

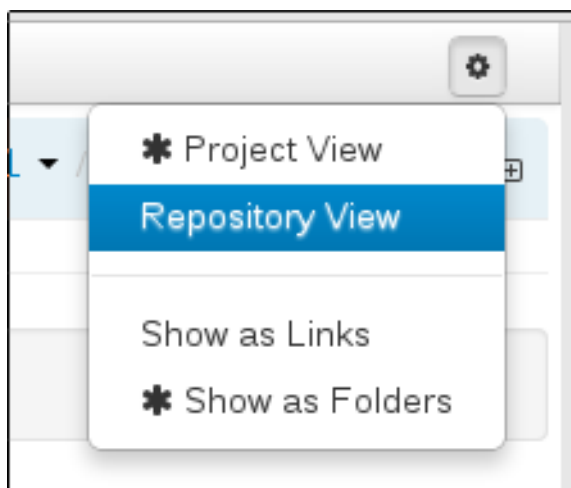


Figure 3.4. Changing to Repository View

2. Navigate to `/src/main/resources/META-INF`. Click on `kmodule.xml` to edit the file directly.
3. Define your **kbases** and **ksessions**. For example:

```
<kmodule xmlns="http://jboss.org/kie/6.0.0/kmodule"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <kbase name="myBase" default="true" eventProcessingMode="stream"
equalsBehavior="identity" packages="*">
```

```
<ksession name="mySession" type="stateless" default="true"
clockType="realtime"/>
</kbase>
</kmodule>
```

- Click **save** in the top right corner.

You can switch between the Project Editor view and the Repository view to look at the changes you make in each view. To do so, 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 |

| Property | Description |
|-----------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>org.uberfire.nio.git.ssh.enabled</code> | Enables/disables ssh daemon. Default: true |
| <code>org.uberfire.nio.git.ssh.host</code> | If ssh daemon enabled, uses this property as local host identifier. Default: localhost |
| <code>org.uberfire.nio.git.ssh.port</code> | If ssh daemon enabled, uses this property as port number. Default: 8001 |
| <code>org.uberfire.nio.git.ssh.cert.dir</code> | Location of the directory .security where local certificates will be stored. Default: working directory |
| <code>org.uberfire.metadata.index.dir</code> | Place where Lucene .index folder will be stored. Default: working directory |
| <code>org.uberfire.cluster.id</code> | Name of the helix cluster, for example: kie-cluster |
| <code>org.uberfire.cluster.zk</code> | Connection string to zookeeper. This is of the form host1:port1,host2:port2,host3:port3 , for example: localhost:2188 |
| <code>org.uberfire.cluster.vfs.lock</code> | Name of the resource defined on helix cluster, for example: kie-vfs |
| <code>org.uberfire.cluster.autostart</code> | Delays VFS clustering until the application is fully initialized to avoid conflicts when all cluster members create local clones. Default: false |
| <code>org.uberfire.sys.repo.monitor.disabled</code> | Disable configuration monitor (do not disable unless you know what you're doing). Default: false |
| <code>org.uberfire.secure.key</code> | Secret password used by password encryption. Default: org.uberfire.admin |
| <code>org.uberfire.secure.alg</code> | Crypto algorithm used by password encryption. Default: PBEWithMD5AndDES |
| <code>org.uberfire.domain</code> | Security-domain name used by uberfire. Default: ApplicationRealm |
| <code>org.guvnor.m2repo.dir</code> | Place where Maven repository folder will be stored. Default: working-directory/repositories/kie |

| Property | Description |
|------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| 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. |
| 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 6.2, 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.

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.

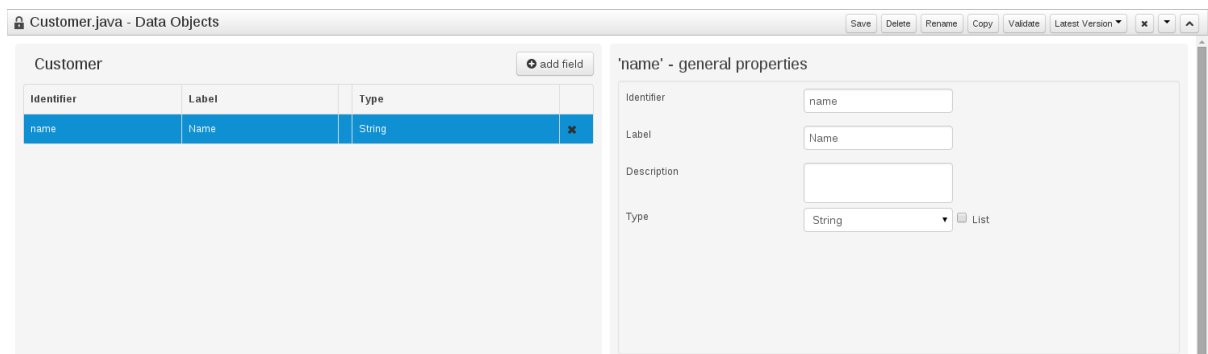
To open the editor, open the Project Authoring perspective, click **New Item** → **Data Object** on the perspective menu. If you want to edit an existing model, these files are located under **Data Objects** in **Project Explorer**.

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.

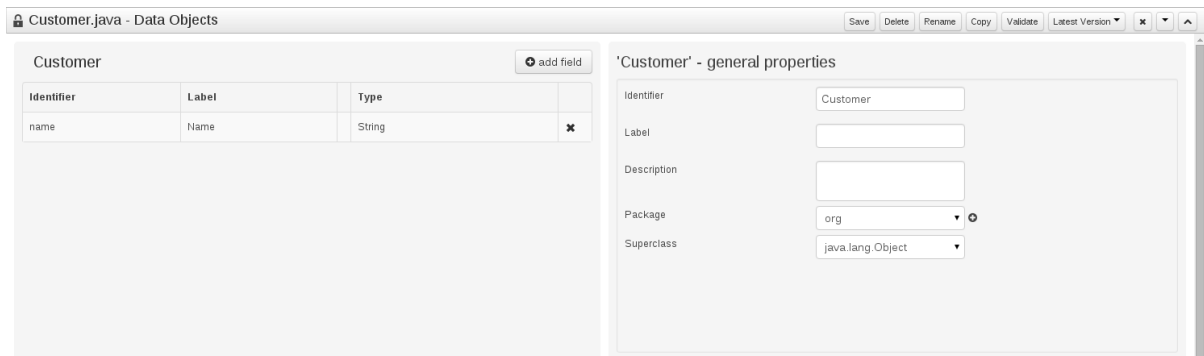
The Data Modeler supports roundtrips between the **Editor** and **Source** tabs, along with source code preservation. This allows you to make changes to your model in external tools, like JBDS, and the Data Modeler updates the necessary code blocks automatically.

In the main editor window the user can

- Add/delete fields
- Select a given field. When a field is selected then the field information will be loaded in all the domain editors.



- Select the data object class. For example, by clicking on the data object name (on the main window) instead of loading the field properties, the domain editors will load the class properties.



5.2. ANNOTATIONS IN DATA MODELER

JBoss BRMS supports all Drools annotations by default, and can be customized using the **Drools & jBPM** interface.

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). You can also use the **Advanced** screen to manage arbitrary annotations.

When creating or adding fields to a persistable data object, the JPA annotations that are added by default will generate a model that can be used by jBPM at runtime. In general, modifying the default configurations where the model will be used by processes is not recommended.

JBoss BRMS 6.2 onwards supports the use of JPA specific annotations, with Hibernate available as the default JPA implementation. Other JPA annotations are also supported where the JPA provider is loaded in the classpath.



NOTE

When adding an annotation in the Data Modeler, the annotation class should be on the workbench classpath, or a project dependency can be added to a `.jar` file that has the annotation. The Data Modeler will run a validation check to confirm that the annotation is on the classpath, and the project will not build if the annotation is not present.

5.3. CREATING A DATA OBJECT (NOT PERSISTABLE)

1. In the Project Authoring perspective, click **New Item** → **Data Object** on the perspective menu. Enter the name and the location (the name has to be unique across the package, but it is possible to have two data object with the same name in two different packages). Uncheck the **Persistable** box to prevent the Data Object from being made persistable. Press the **Ok** button.
2. Create fields of the data object:
 - a. By clicking **add field** button in the main editor window, you can add a field to the object with the attributes **Id**, **Label** and **Type**. Required attributes are marked with *****.
 - **Id**: field ID unique within the data object
 - **Label**: label to be used in the **Fields** panel (optional)
 - **Type**: data type of the field

New field

×

*Id

Insert a valid Java identifier

Label

Insert a label

*Type

☐ List

Create

Create and continue

Cancel

- b. Click the **Create** button (the new field is created and the **New field** window closes) or the **Create and continue** button (the new field is created and the **New field** window remains open, so it is possible to continue adding more fields).

Attributes can be edited selecting the attribute and editing the fields using the general properties screen to the right.



IMPORTANT

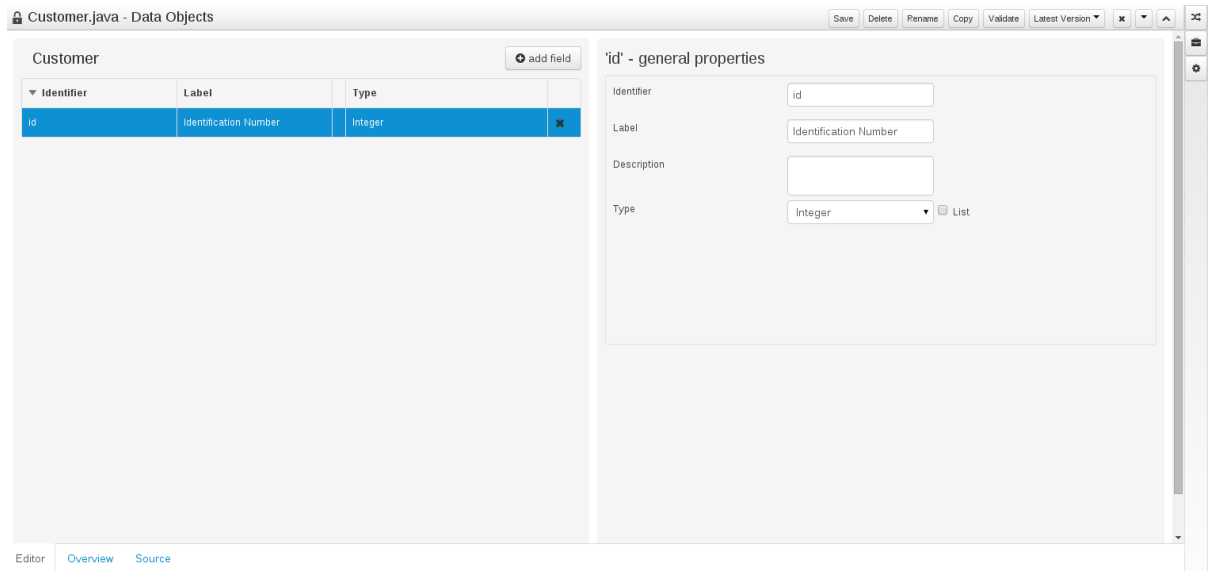
To use a data object, make sure you import the data model into your resource. This is necessary even if the data model lives in the same Project as your resource (Business Process).

5.4. PERSISTABLE DATA OBJECTS

From JBoss BRMS 6.2 onwards, the Data Modeler is extended to support the generation of persistable Data Objects by checking the **Persistable** check box on the perspective menu when you create a new Data Object. Persistable Data Objects are based on the JPA specification. When the **Persistable** check box is selected when the object is created, the platform will set some configurations required for persistence by default. The object can also be made persistable after the object has already been created. This can then be customized as required.

Procedure 5.1. Creating a Persistable Data Object

1. When creating a persistable Data Object, the Identifier field **id** will be generated automatically, along with the **@Entity** annotation.



Selecting the **List** option changes a property to have multiple values. For example, it can enable an attribute **Name**, to have a list of strings with multiple names.

2. Create fields of the data object:

- a. By clicking **add field** button in the main editor window, you can add a field to the object with the attributes **Id**, **Label** and **Type**. Required attributes are marked with *****.

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

New field ✕

***Id**

***Type** ☐ List

Label

Create
Create and continue
Cancel

- b. Click the **Create** button (the new field is created and the **New field** window closes) or the **Create and continue** button (the new field is created and the **New field** window remains open, so it is possible to continue adding more fields).

Attributes can be edited selecting the attribute and editing the fields using the general properties screen to the right.

5.5. DATA OBJECT DOMAIN SCREENS

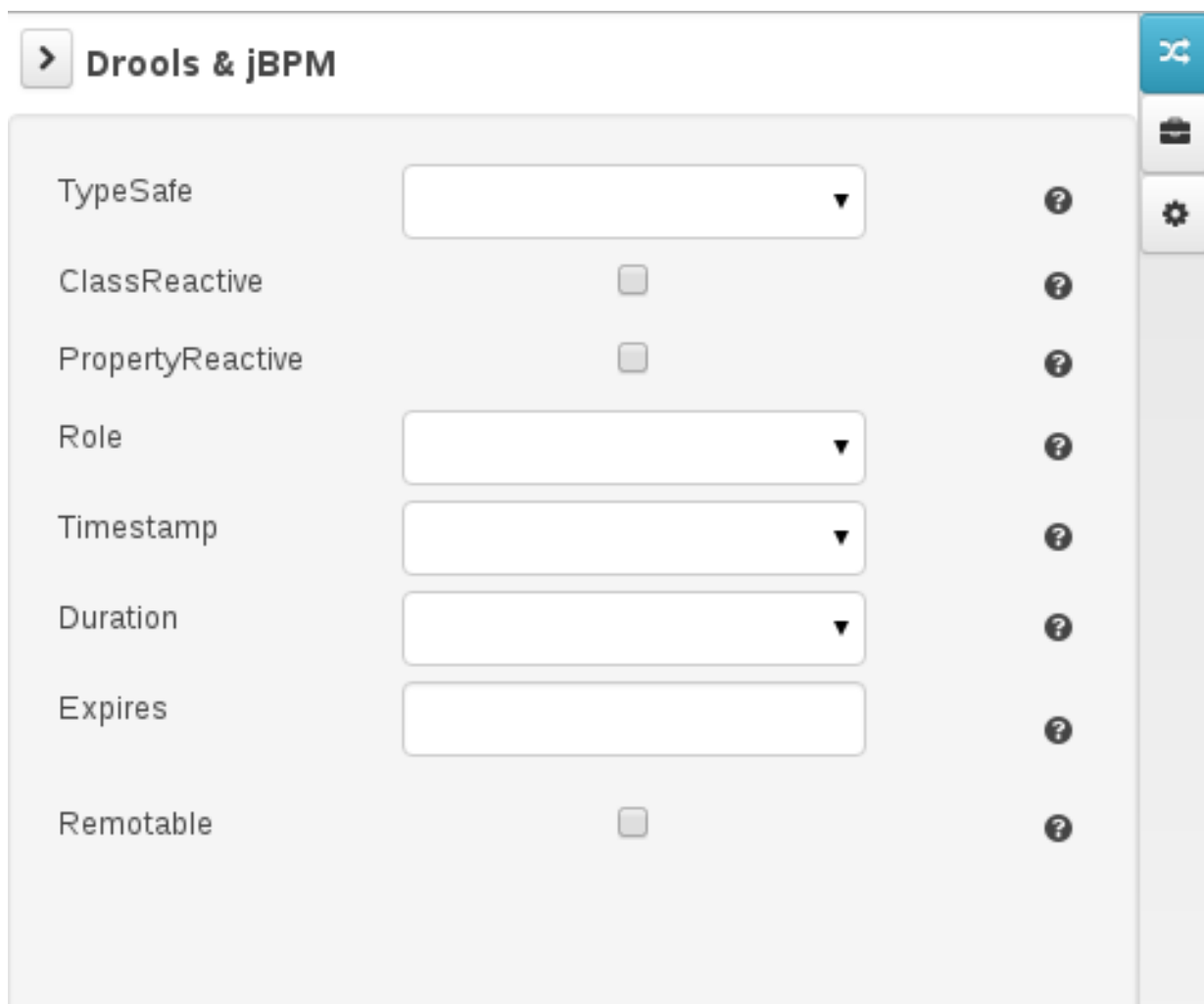
The following domain screen tabs can be selected from the right side of the Data Object editor screen.

Drools & jBPM

The **Drools & jBPM** screen allows configuration of Drools-specific attributes.

The Data Modeler in Business Central supports the editing of pre-defined annotations of fact model classes and attributes. The following Drools annotations are supported, and can be customized using the **Drools & jBPM** interface:

- TypeSafe
- Role
- Timestamp
- Duration
- Expires



| Attribute | Configuration | Help |
|------------------|--------------------------|------|
| TypeSafe | <input type="text"/> | ? |
| ClassReactive | <input type="checkbox"/> | ? |
| PropertyReactive | <input type="checkbox"/> | ? |
| Role | <input type="text"/> | ? |
| Timestamp | <input type="text"/> | ? |
| Duration | <input type="text"/> | ? |
| Expires | <input type="text"/> | ? |
| Remotable | <input type="checkbox"/> | ? |

For the fields within the fact model, the position annotation is supported. The **Drools & jBPM** screen when a specific field is selected looks as follows:

> **Drools & jBPM**

Equals ☐ ?

Position ?

Persistence

The **Persistence** screen can be used to configure attributes on basic JPA annotations for persistence. Fine tuning of annotations, or to add specific annotations, use the **Advanced** screen.

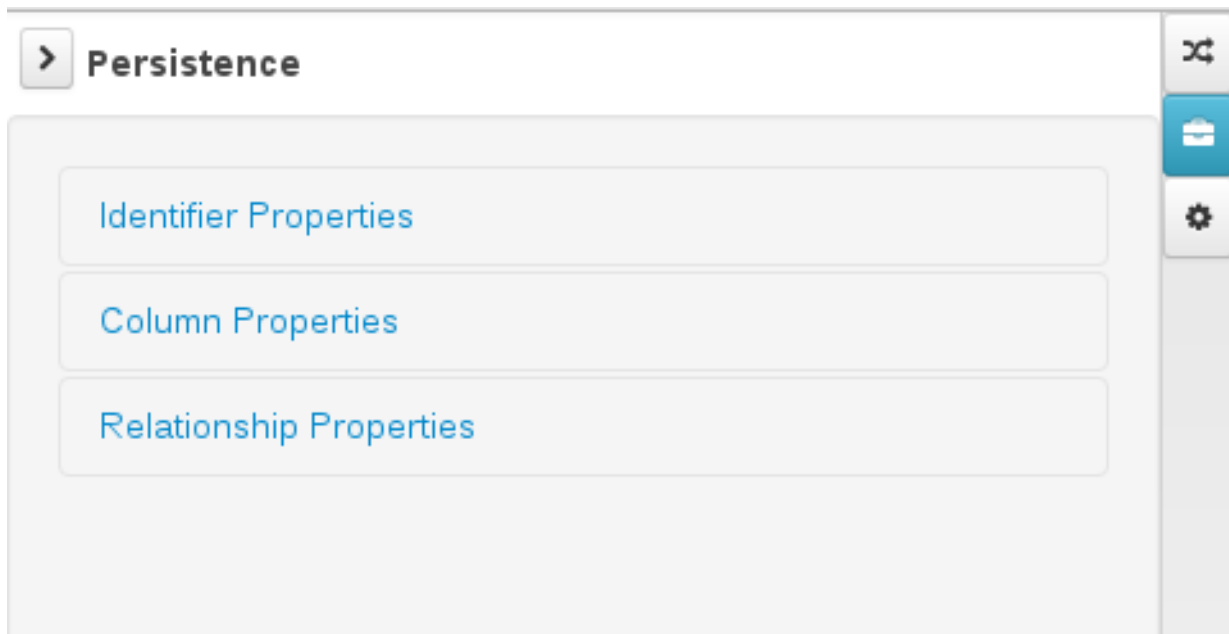
> **Persistence**

Entity Properties

Persistable ☐ ?

Table name ?

The **Persistence** screen when a specific field is selected looks as follows:



The following annotations can be managed via the **Persistence** screen.

Table 5.1. Type Annotations

| Annotation | Automatically Generated when Data Object is Persistable |
|---------------------------------------|---------------------------------------------------------|
| <code>javax.persistence.Entity</code> | Yes |
| <code>javax.persistence.Table</code> | No |

Table 5.2. Field Annotations

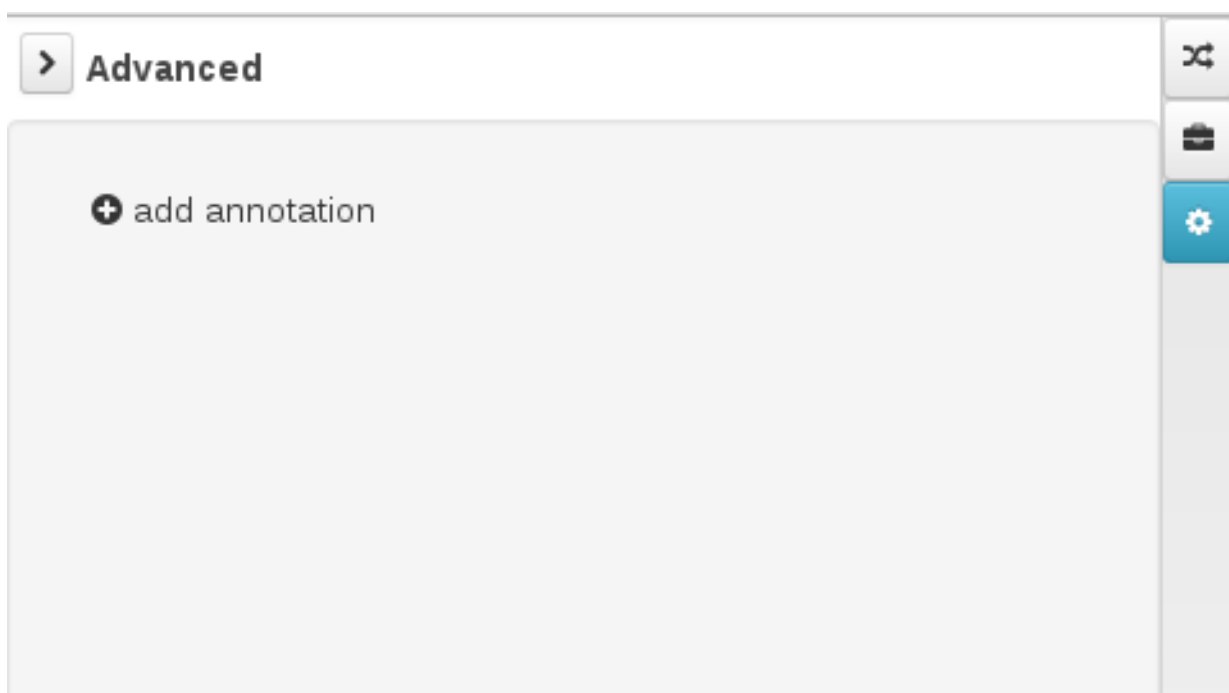
| Annotation | Automatically Generated when Data Object is Persistable |
|--------------------------------------------------|---------------------------------------------------------|
| <code>javax.persistence.Id</code> | Yes |
| <code>javax.persistence.GeneratedValue</code> | Yes |
| <code>javax.persistence.SequenceGenerator</code> | Yes |
| <code>javax.persistence.Column</code> | No |
| <code>javax.persistence.OneToOne</code> | No |
| <code>javax.persistence.OneToMany</code> | Yes - when a field has one or multiple values |
| <code>javax.persistence.ManyToOne</code> | Yes - when a field has multiple values |
| <code>javax.persistence.ManyToMany</code> | No |

| Annotation | Automatically Generated when Data Object is Persistable |
|-------------------------------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| javax.persistence.ElementCollection | Yes - generated by the UI when a new field has one or multiple of a base java type, such as Integer, Boolean, String. This annotation cannot be edited with the Persistence screen tool (use the Advanced screen tool instead). |

All other JPA annotations can be added using the **Advanced** screen.

Advanced

The **Advanced** screen is used for fine-tuning of annotations. Annotations can be configured, added and removed using the Advanced Domain screen. These can be any annotation that is on the classpath.



After you click on the **add annotation** option, the **Add new Annotation** window is displayed. It is required to enter a fully qualified class name of an annotation and by pressing the **search** icon, the annotation definition is loaded into the wizard. Then it is possible to set different annotation parameters (required parameters are marked with *).

Add new Annotation



✓ Search annotation

- ✓ -> cascade
- ✓ -> fetch
- ✓ -> optional
- ✓ -> targetEntity

* Annotation class name



Annotation definition was loaded successfully.

[< Previous](#)[Next >](#)[Cancel](#)[✓ Finish](#)

If possible, the wizard will provide a suitable editor for the given parameters.

Add new Annotation



✓ Search annotation

- ✓ -> **cascade**
- ✓ -> fetch
- ✓ -> optional
- ✓ -> targetEntity

cascade:

- ☐ ALL
- ☐ PERSIST
- ☐ MERGE
- ☐ REMOVE
- ☐ REFRESH
- ☐ DETACH
- ☐ {}

[< Previous](#)[Next >](#)[Cancel](#)[✓ Finish](#)

If it is not possible to provide a customized editor, the wizard will provide a generic parameter editor.

Add new Annotation ✕

- ✓ Search annotation
- ✓ -> cascade
- ✓ -> fetch
- ✓ -> optional
- ✓ -> **targetEntity**

targetEntity:

1

Validate

Enter an optional value for the annotation value pair and press the validate button

< Previous

Next >


Cancel

✓ Finish

After you enter all the required parameters, the **Finish** button is enabled and the annotation can be added to the given field or data object.

5.6. CONFIGURING RELATIONSHIPS BETWEEN DATA OBJECTS

When an attribute type is defined as another Data Object, the relationship is identified and defined by

the  symbol in the object attribute list. You can jump to the Data Object definition to view and edit by clicking on the icon.

Relationship customization is only relevant where the data object is persistable.

Relationships can be configured by selecting an attribute with a relationship and choosing the **Persistence** button on the right. Under **Relationship Properties**, click the **Relationship Type** property editing option.

Relationship configuration



Relationship type

Many to One ▼

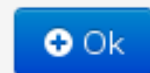
Cascade mode

☒ All ☒ Persist ☒ Merge ☒ Remove ☒ Refresh ☒ Detach

Fetch mode

EAGER ▼

Optional

☐


Cancel

Attempting to delete a Data Object that has a relationship with another Data Object will show the **Usage Detected** screen. It is still possible to delete the object from here, however this will stop your project from building successfully until the resulting errors are resolved.

5.7. PERSISTENCE DESCRIPTOR

When creating persistent data objects, the `persistence.xml` file is created by default when the user opens the project editor and selects **Persistence Descriptor**. It can then be configured via the Persistence Descriptor. The Persistence Descriptor is accessible by opening the Project Editor, and clicking **Project Settings: Project General Settings → Persistence descriptor**.

persistence.xml - Persistence descriptor

Save Delete Rename Copy Validate Latest Version ▼ X ▼ ▲

Persistence Unit
com.redhat.test-project:1.0

Persistence Provider
org.hibernate.ejb.HibernatePersistence

Data Source
java:jboss/datasources/ExampleDS

Transactions Type
☒ JTA

[Advanced properties](#)

[Project persistable Data Objects](#)

In the **Advanced properties** section, it is possible to change or delete all the properties generated by default and add new ones as well.

Advanced properties

| Property Name | Property Value | Action |
|-------------------------------------|-----------------------------------------------------------------------|--------------------------------------------------------------------------------------------|
| hibernate.dialect | org.hibernate.dialect.H2Dialect |  Delete |
| hibernate.max_fetch_depth | 3 |  Delete |
| hibernate.hbm2ddl.auto | update |  Delete |
| hibernate.show_sql | false |  Delete |
| hibernate.id.new_generator_mappings | false |  Delete |
| hibernate.transaction.jta.platform | org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatform |  Delete |

[New property](#)

If you open the **Project persistable Data Objects** section in the Persistence Descriptor, you will see that there are two available buttons:

- **Add class** enables the user to add arbitrary classes to the `persistence.xml` file to be declared as entities.
- **Add project persistable classes** will automatically load all the persistable data objects in the current project.

Project persistable Data Objects

| Class name | Action |
|---------------------|--------|
| No classes selected | |

0 of 0

enter a persistable class name

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:
 - o If in the **Project** view of Project Explorer, select the organizational unit, repository and the project where you want to create the rule.
 - o 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.

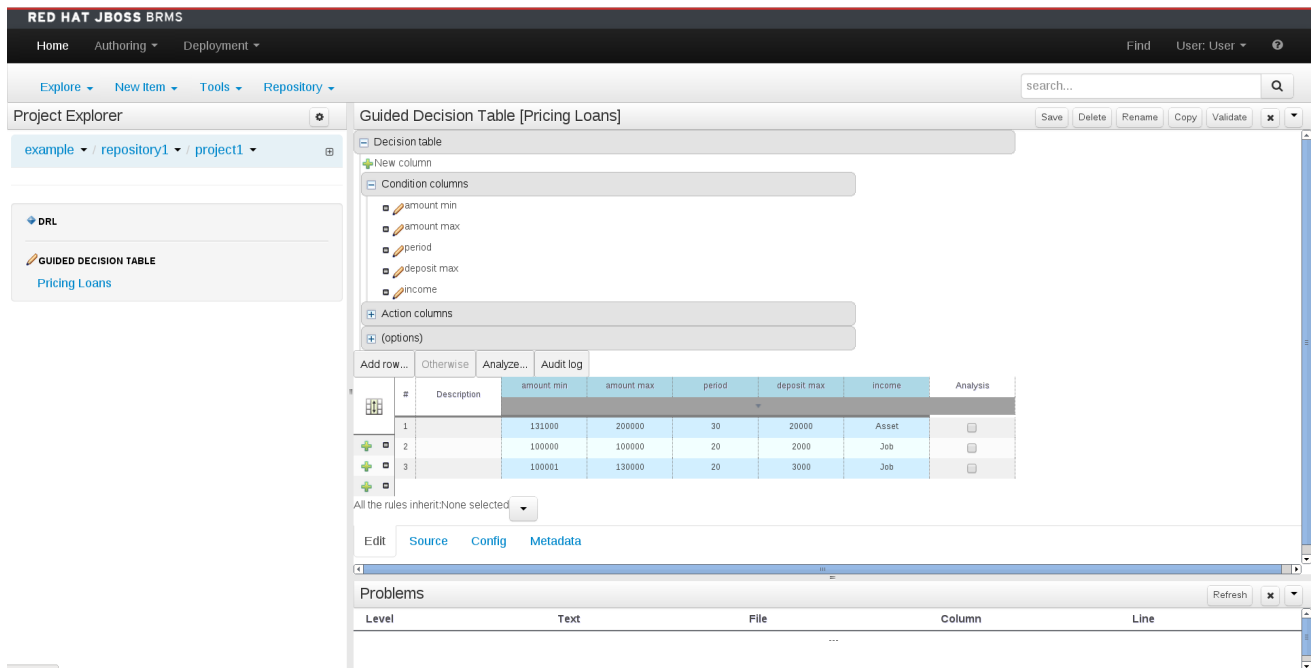


Figure 6.1. The asset editor - Edit tab

Source tab

Source tab shows the DRL source for a selected asset.

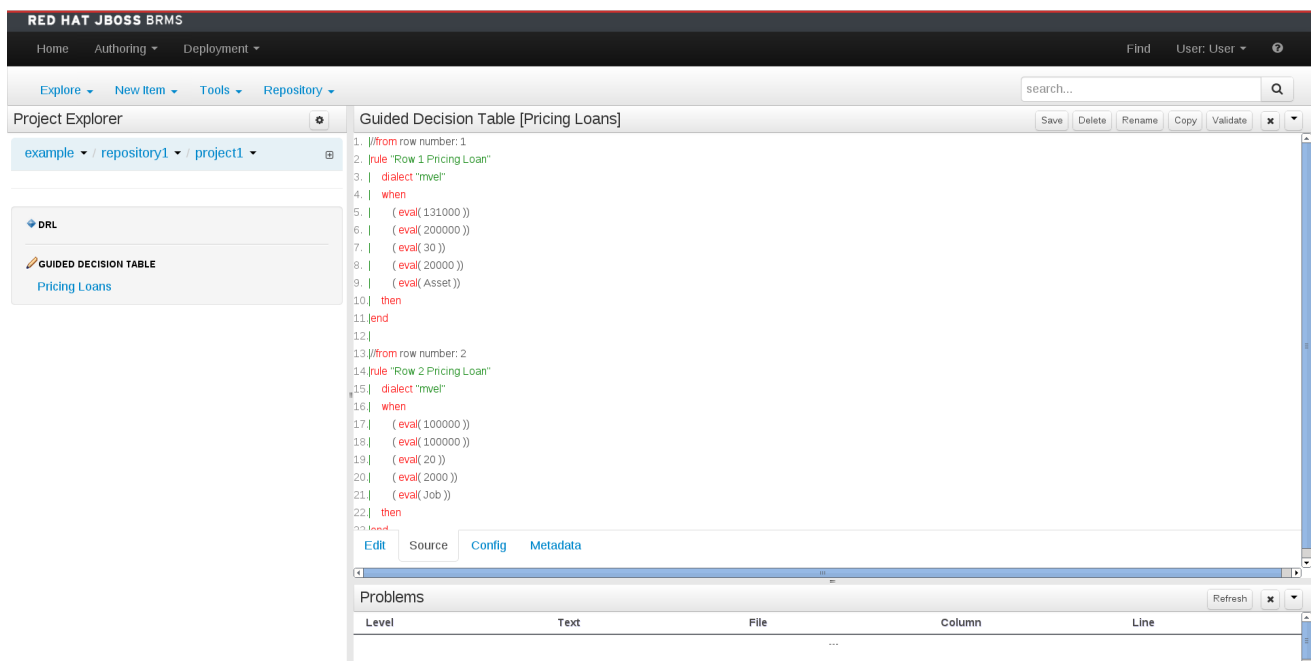


Figure 6.2. The asset editor - Source tab

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.

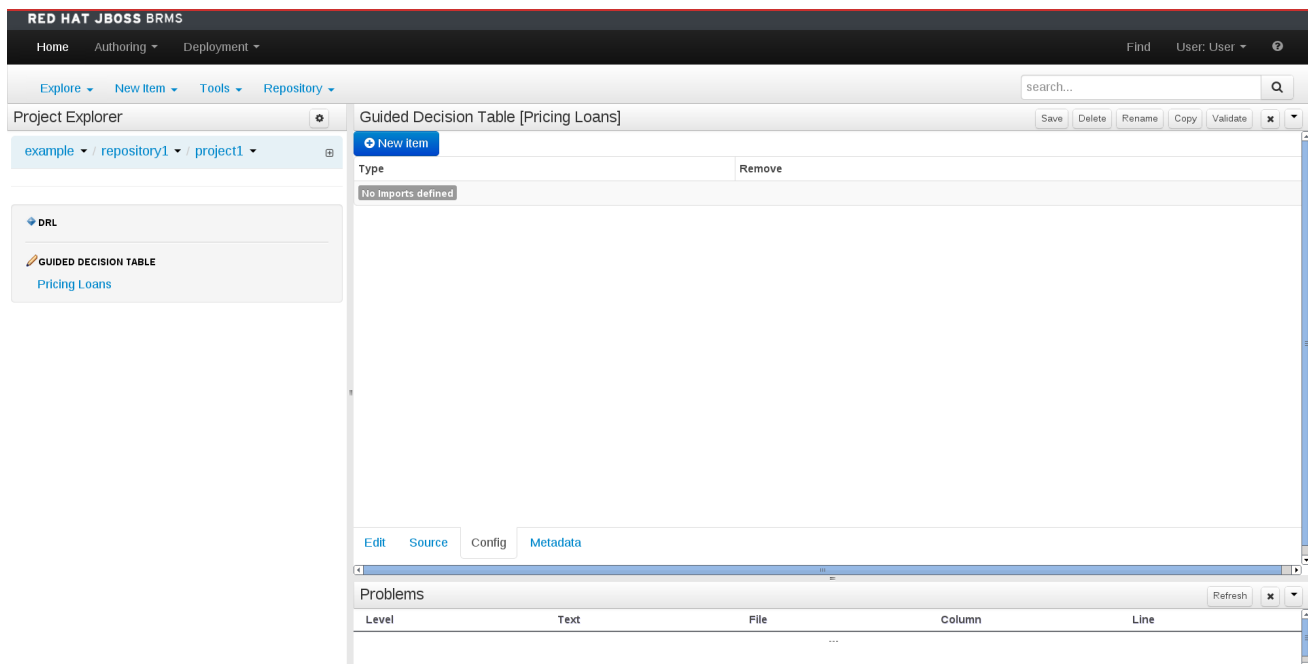


Figure 6.3. The asset editor - Config tab

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.

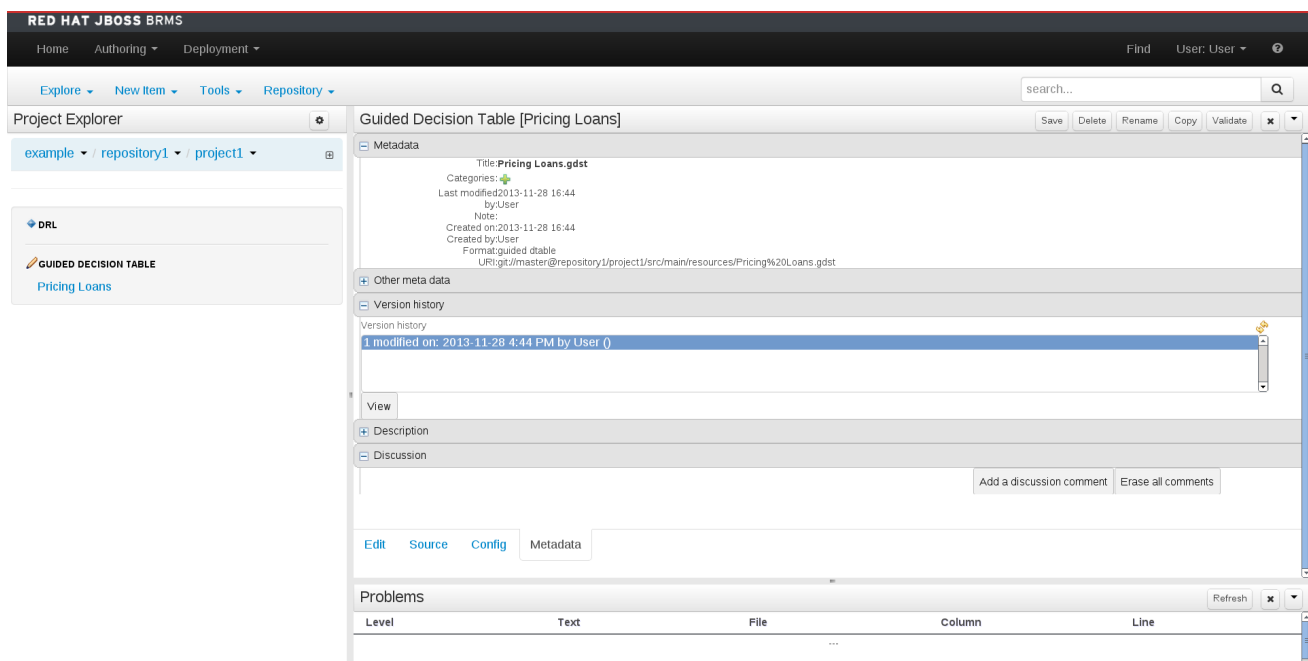


Figure 6.4. The asset editor - Metadata tab

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

WHEN

1. There is a LoanApplication [application]
2. There is an Applicant with:
age 21

THEN

1. Set value of LoanApplication [application] approved
- Set value of LoanApplication [application] explanation
2. Retract LoanApplication [application]

(show options...)

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, an 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 |
| | Standard | N*\$1 |

| 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 represent 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.

Create new Guided Decision Table

* Resource Name

resource name...

Location

default://master@repository1/project1/src/main/resources

☐ Use Wizard

☒ Extended entry, values defined in table body

☐ Limited entry, values defined in columns

+ Ok

Cancel

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.

✓ Summary

✓ Imports

✓ Add Fact Patterns

✓ Add Constraints

✓ Add Actions to update Facts

✓ Add Actions to insert Facts

✓ Columns to expand

Summary of fields for the decision table.

Name: *

Path: default://master@repository1/project1/src/main/resources

Table Format: Extended entry, values defined in table body

<- Previous

Next ->

Cancel

Finish

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).

Guided Decision Table Editor [gdt2]

All the rules inherit: None selected

Decision table

New column

Condition columns

Action columns

(options)

Configure the columns first, then add rows (rules). A fact model (in the current package) will be needed to provide the facts and fields to configure this decision table.

Add row...

Otherwise

Analyze...

Audit log

| # | Description |
|---|-------------|
| | |

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.

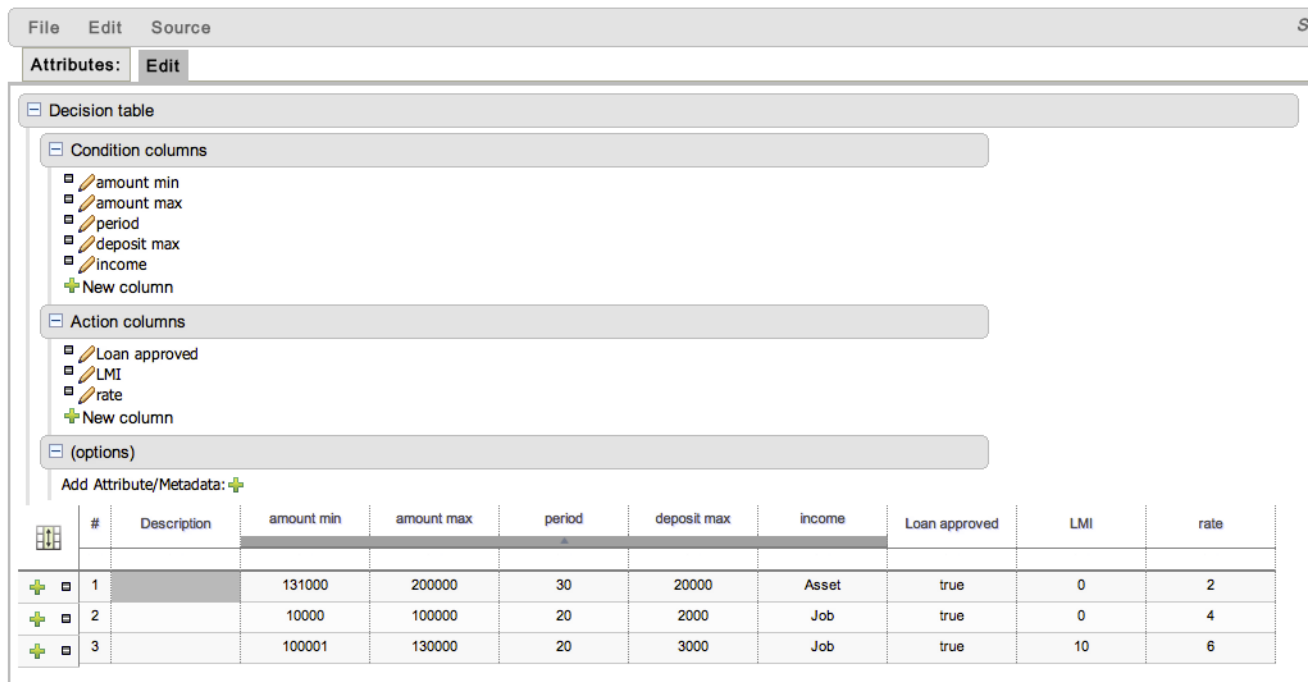


Figure 6.6. Column Configuration

6.4.4. Adding Columns

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

The following column type selection dialog appears:

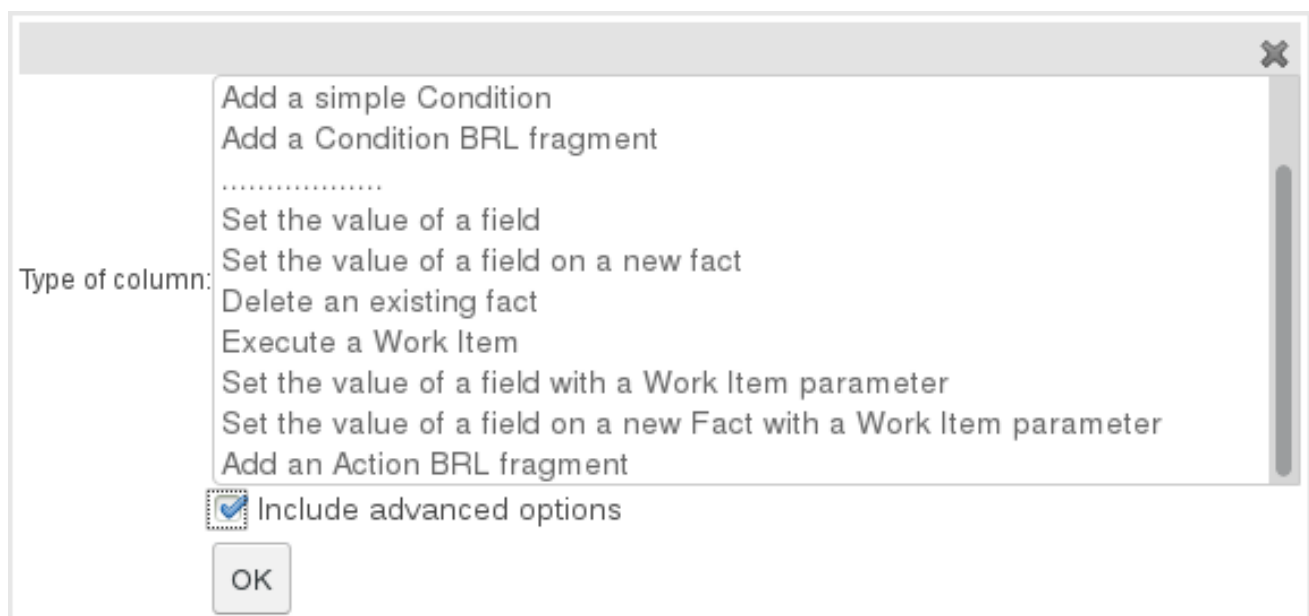


Figure 6.7. Advanced Column Options

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.

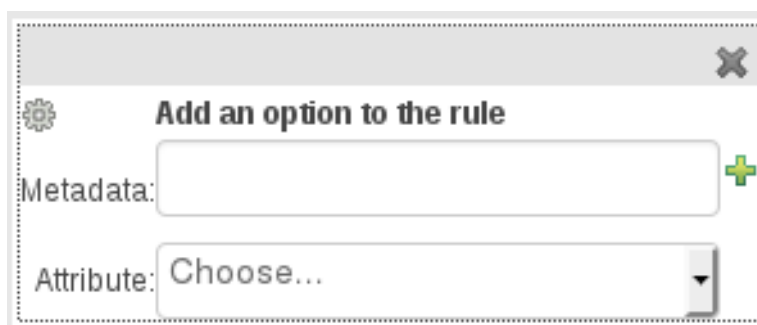


Figure 6.8. Attribute and MetaData Option

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.

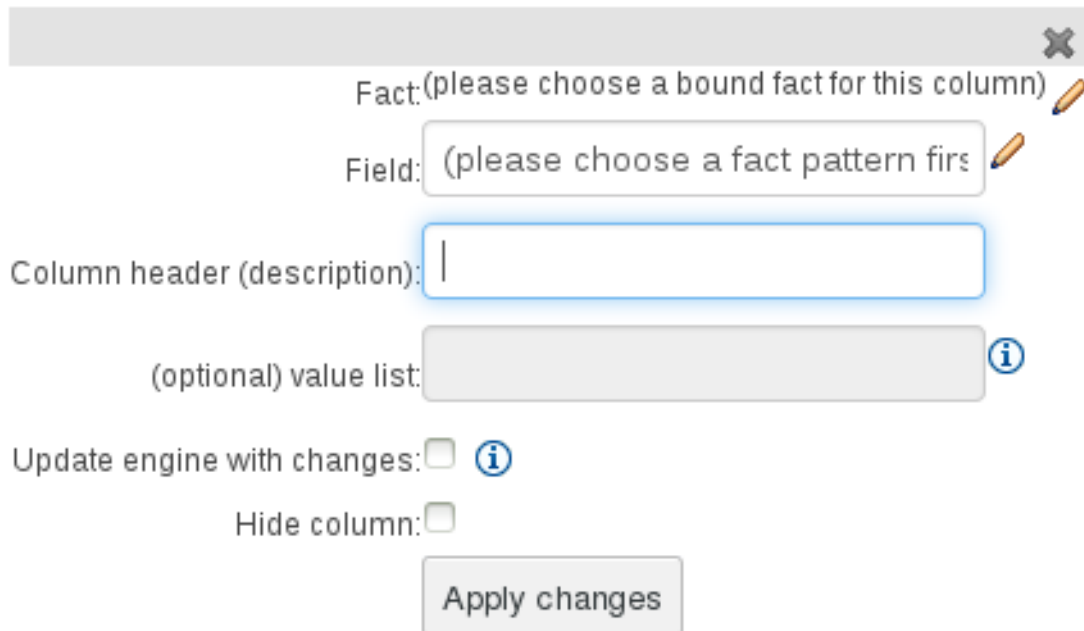
When you edit or create a new column, you will be given a choice of the type of constraint:

- **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.



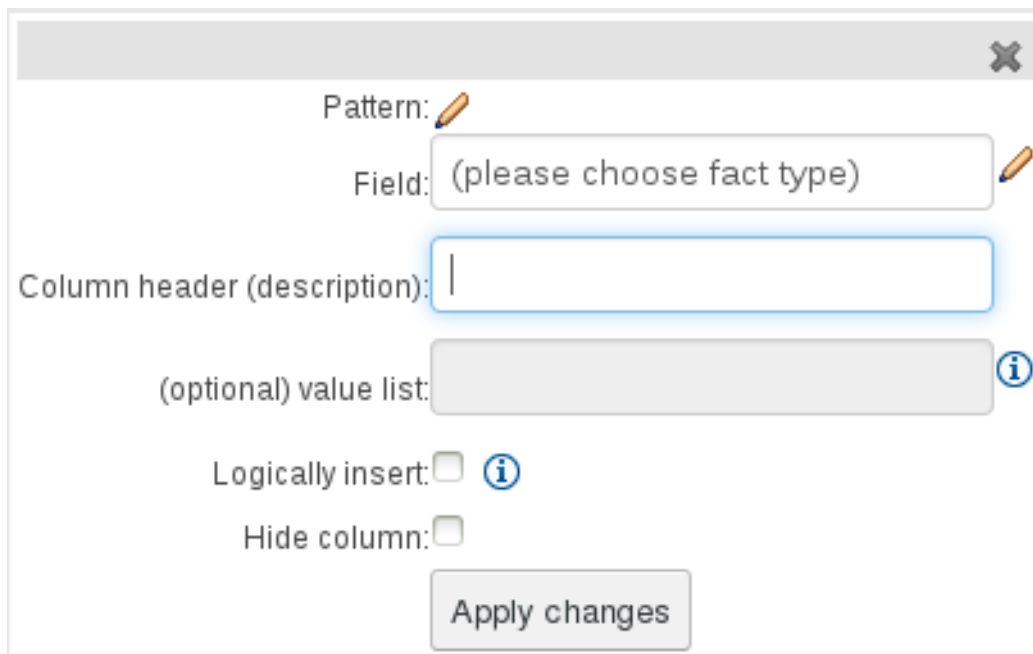
A dialog box titled "Set the value of a field" with a close button (X) in the top right corner. The dialog contains the following fields and controls:

- Fact:** (please choose a bound fact for this column) with a pencil icon.
- Field:** (please choose a fact pattern first) with a pencil icon.
- Column header (description):** A text input field.
- (optional) value list:** A text input field with an information icon (i) to its right.
- Update engine with changes:** A checkbox with an information icon (i) to its right.
- Hide column:** A checkbox.
- Apply changes** button.

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.



A dialog box titled "Set the value of a field on a new fact" with a close button (X) in the top right corner. The dialog contains the following fields and controls:

- Pattern:** A text input field with a pencil icon.
- Field:** (please choose fact type) with a pencil icon.
- Column header (description):** A text input field.
- (optional) value list:** A text input field with an information icon (i) to its right.
- Logically insert:** A checkbox with an information icon (i) to its right.
- Hide column:** A checkbox.
- Apply changes** button.

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

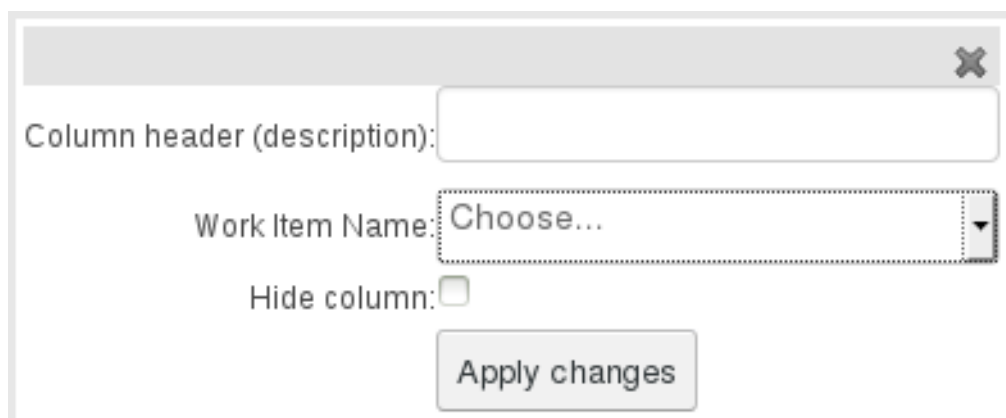
A construct that allows a BRL fragment to be used in the left-hand side of a rule. A BRL fragment is authored using the Guided Rule Editor and hence all features available in that editor can be used to define a decision table column such as the following: "from", "collect", and "accumulate". When using 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

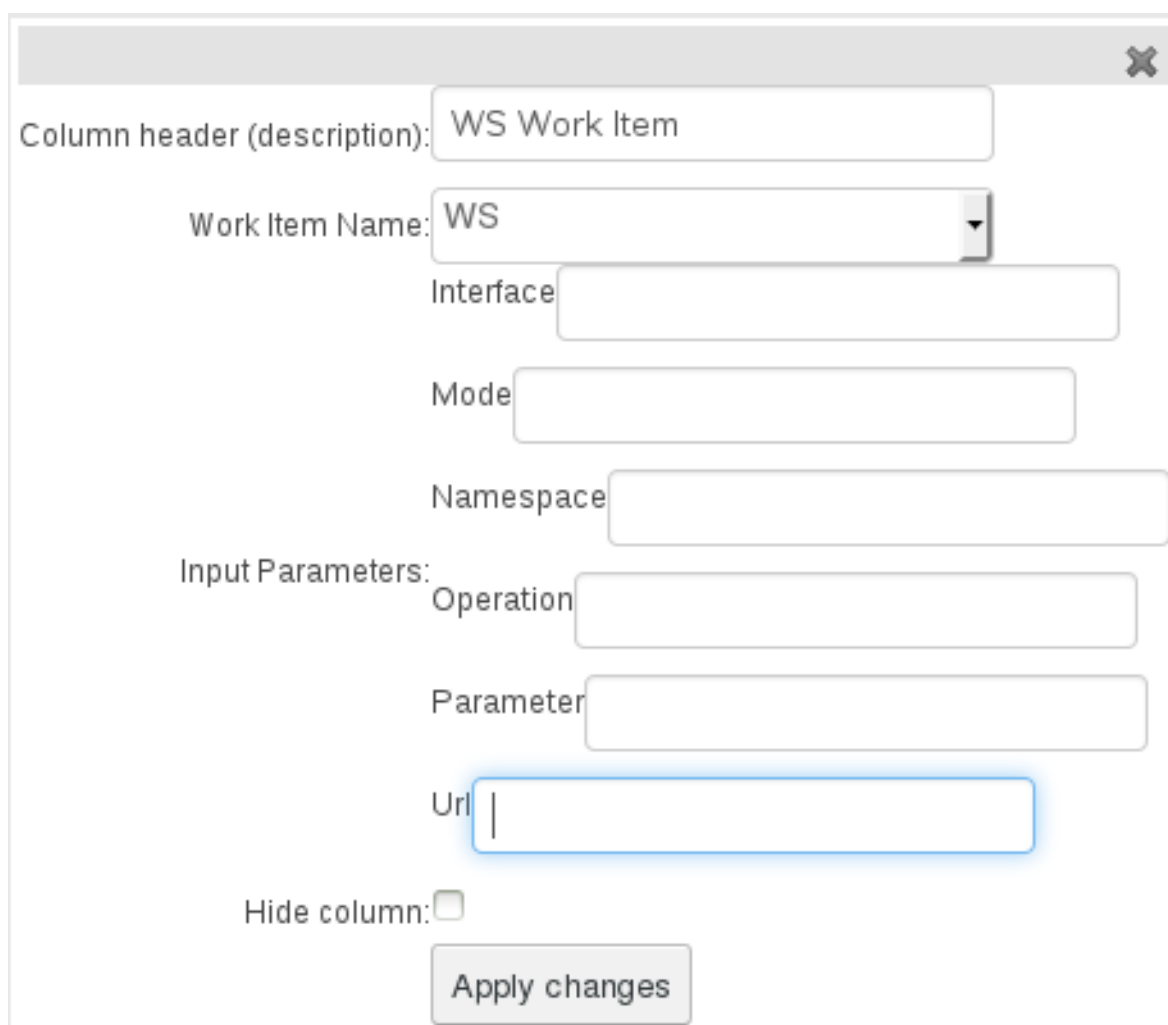
This implements an Action to invoke a Red Hat JBoss Business Process Management Suite Work Item Handler. It sets its input parameters to bound Facts/Facts' fields values. This works for any Work Item Definition.



A dialog box titled "Execute a Work Item" with a close button (X) in the top right corner. It contains the following fields and controls:

- "Column header (description):" followed by an empty text input field.
- "Work Item Name:" followed by a dropdown menu showing "Choose..." with a downward arrow.
- "Hide column:" followed by an unchecked checkbox.
- An "Apply changes" button at the bottom right.

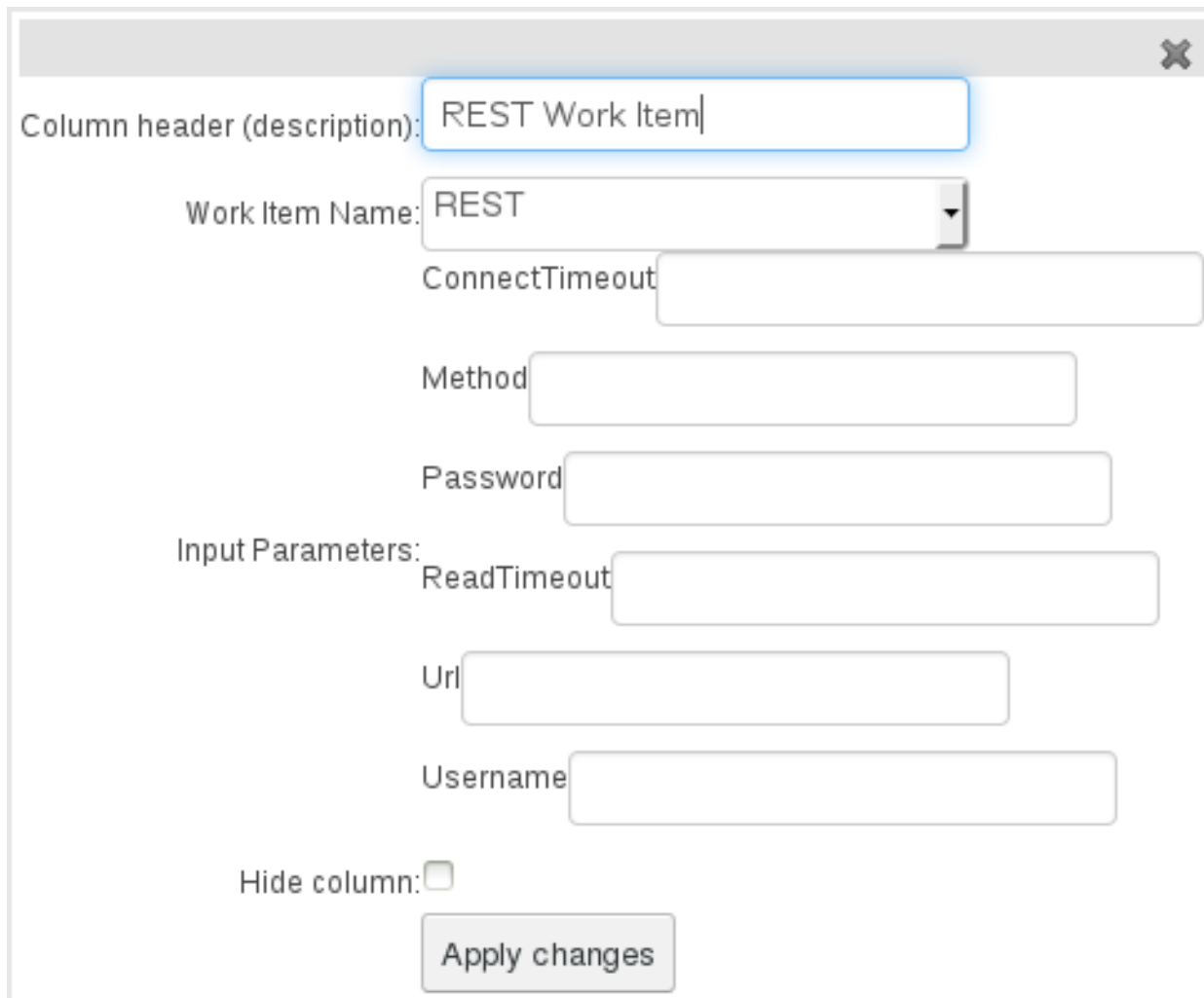
Figure 6.14. Execute a Work Item



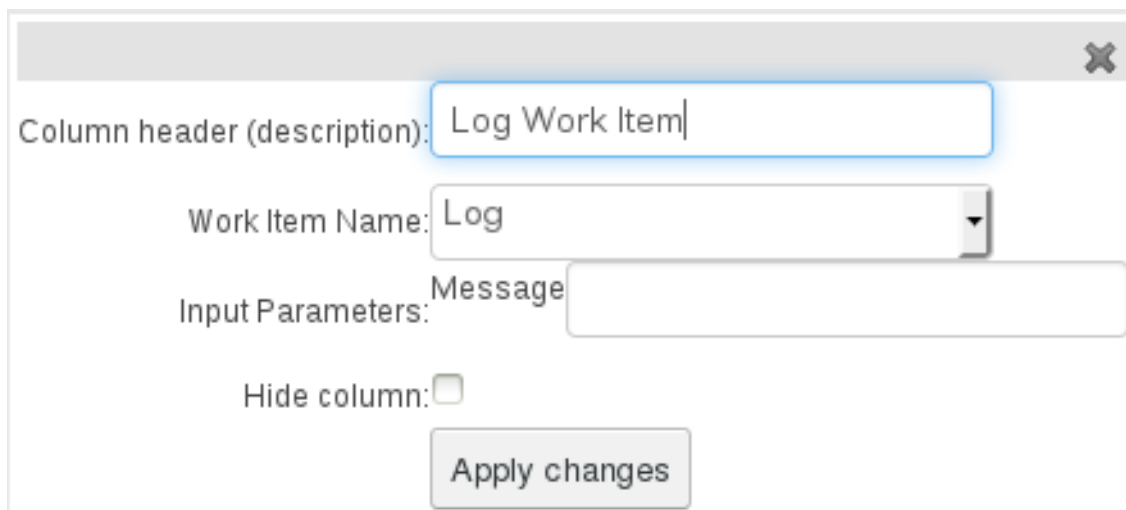
A dialog box titled "WS Work Item" with a close button (X) in the top right corner. It contains the following fields and controls:

- "Column header (description):" followed by a text input field containing "WS Work Item".
- "Work Item Name:" followed by a dropdown menu showing "WS" with a downward arrow.
- "Interface" followed by an empty text input field.
- "Mode" followed by an empty text input field.
- "Namespace" followed by an empty text input field.
- "Input Parameters:" followed by two stacked empty text input fields labeled "Operation" and "Parameter".
- "Url" followed by an empty text input field with a blue border and a vertical cursor.
- "Hide column:" followed by an unchecked checkbox.
- An "Apply changes" button at the bottom right.

Figure 6.15. WS Work Item

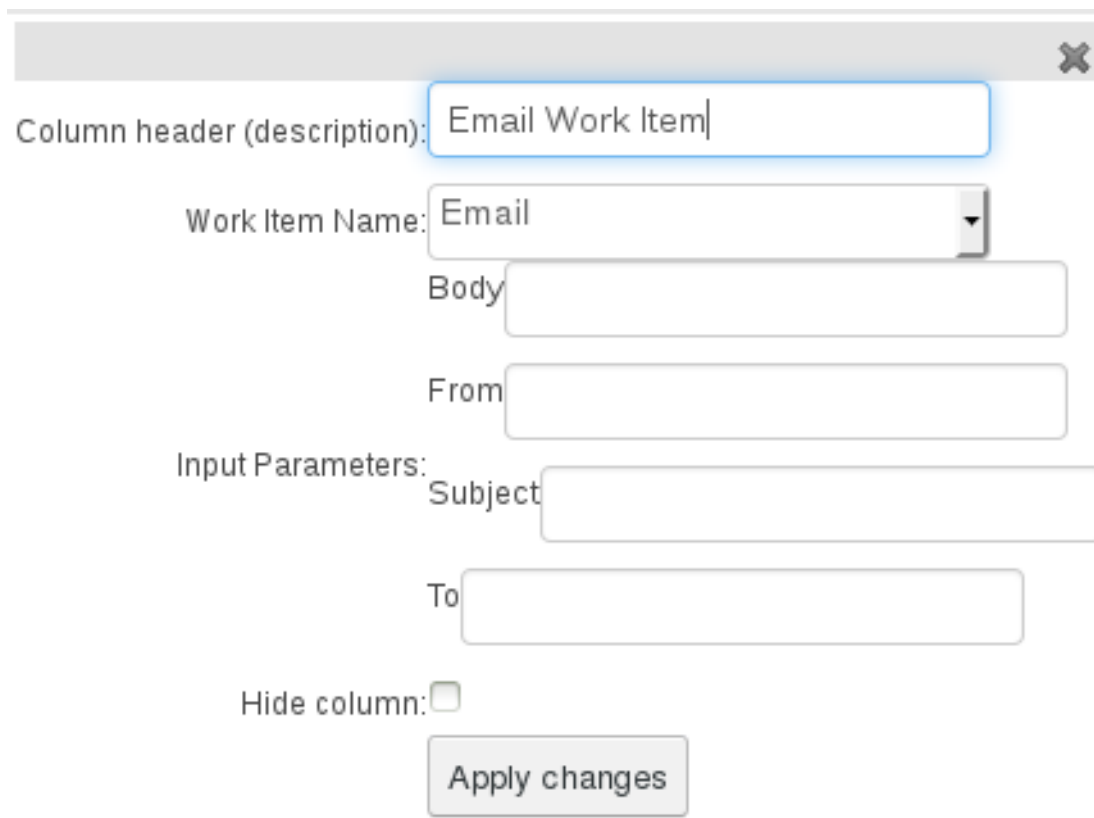


A screenshot of a configuration dialog box titled "REST Work Item". The dialog has a close button (X) in the top right corner. It contains several input fields and a checkbox. The "Column header (description):" field is highlighted with a blue border and contains the text "REST Work Item". Below it, the "Work Item Name:" dropdown menu is set to "REST". There are text input fields for "ConnectTimeout", "Method", "Password", "ReadTimeout", "Url", and "Username". The "Input Parameters:" label is positioned to the left of the "ReadTimeout" field. At the bottom, there is a "Hide column:" checkbox which is unchecked, and an "Apply changes" button.

Figure 6.16. REST Work Item

A screenshot of a configuration dialog box titled "Log Work Item". The dialog has a close button (X) in the top right corner. It contains several input fields and a checkbox. The "Column header (description):" field is highlighted with a blue border and contains the text "Log Work Item". Below it, the "Work Item Name:" dropdown menu is set to "Log". There is a text input field for "Message" under the "Input Parameters:" label. At the bottom, there is a "Hide column:" checkbox which is unchecked, and an "Apply changes" button.

Figure 6.17. Log Work Item



Column header (description): Email Work Item

Work Item Name: Email

Body

From

Input Parameters: Subject

To

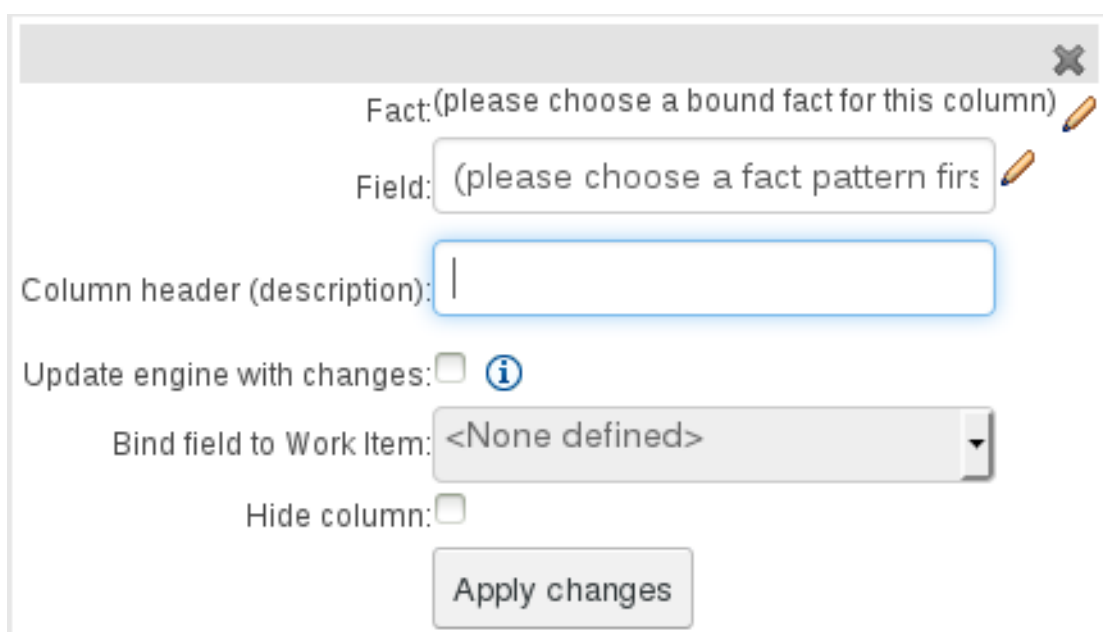
Hide column: ☐

Apply changes

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.



Fact: (please choose a bound fact for this column)

Field: (please choose a fact pattern first)

Column header (description):

Update engine with changes: ☐ ⓘ

Bind field to Work Item: <None defined>

Hide column: ☐

Apply changes

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.

Column header (description):

Hide column: ☐

THEN +

Ok Cancel

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.

| | # | Description | salience | name | age |
|---|----|-------------|----------|--------------|-------------|
| | | | | Person [\$p] | |
| | | | | name [:=] | age [:=] |
| + | 1 | | 1 | Bill | 30 |
| + | 2 | | 2 | | |
| + | 3 | | 3 | | |
| + | 4 | | 4 | | |
| + | 5 | | 5 | | |
| + | 6 | | 6 | Ben | <otherwise> |
| + | 7 | | 7 | Weed | 40 |
| + | 8 | | 8 | <otherwise> | 50 |
| + | 9 | | 9 | | |
| + | 10 | | 10 | | |
| + | 11 | | 11 | | |
| + | 12 | | 12 | | |

Add row...

Otherwise

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:
 - o If you are in the Project view, select the organizational unit, repository, and the project where you want to create the template.
 - o 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.

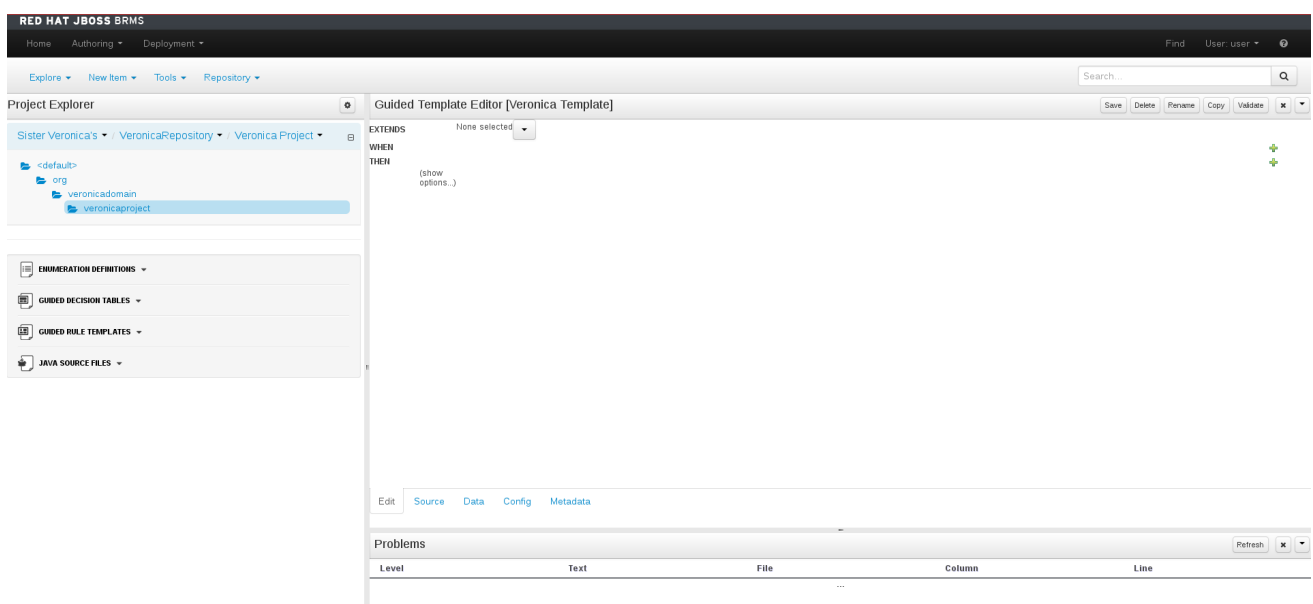


Figure 6.25. Guided Template Editor

**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** constraints 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 5.3, “Creating a Data Object \(Not Persistable\)”](#)), 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.

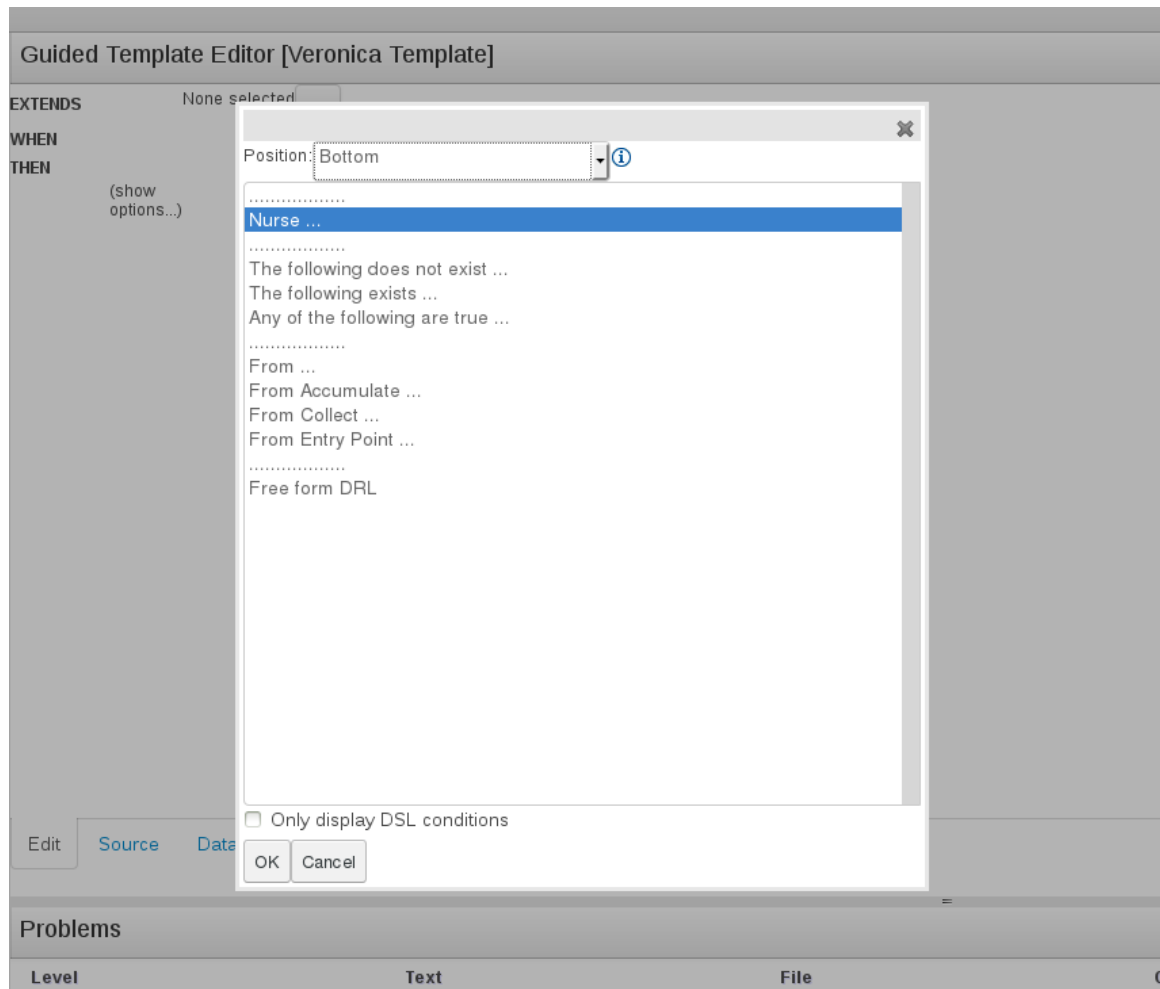


Figure 6.26. Nurse Roster WHEN Dialog Window

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.

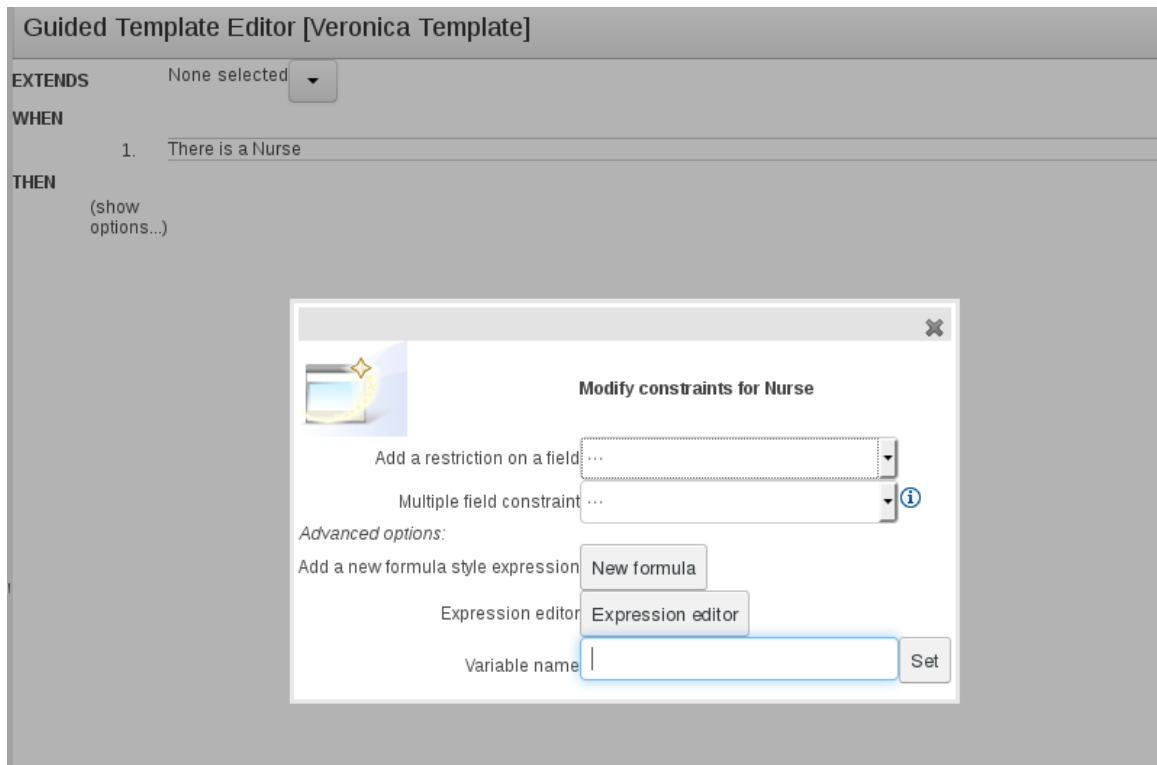


Figure 6.27. Modify Constraints Dialog

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.

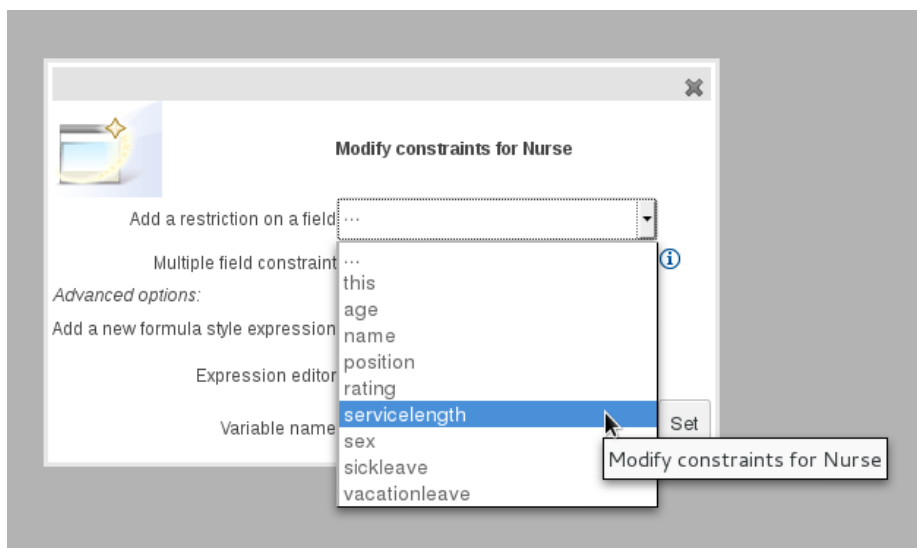


Figure 6.28. Title

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."

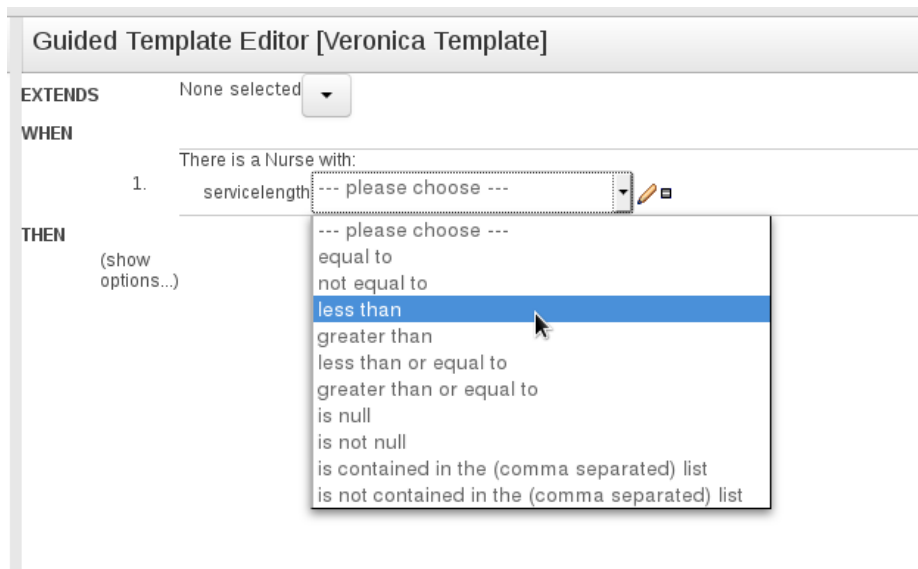



Figure 6.29. Restriction drop-down menu

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.

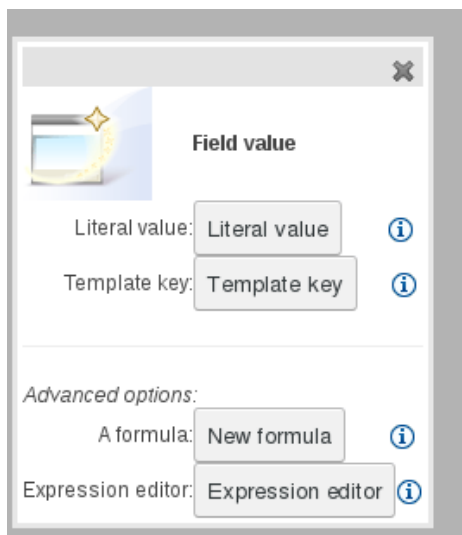


Figure 6.30. Field Value Options

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.

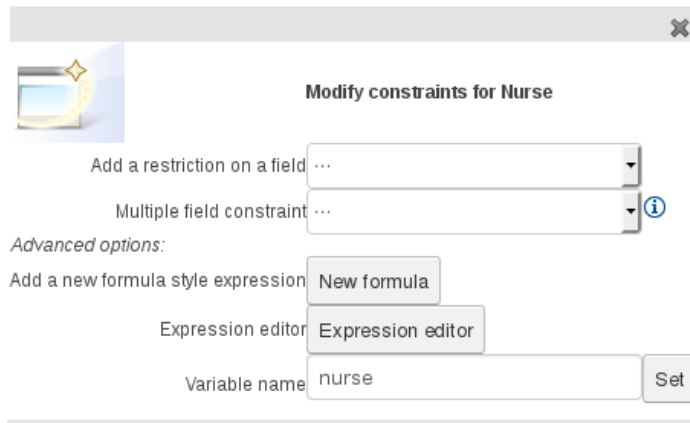


Figure 6.31. Setting a variable name

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

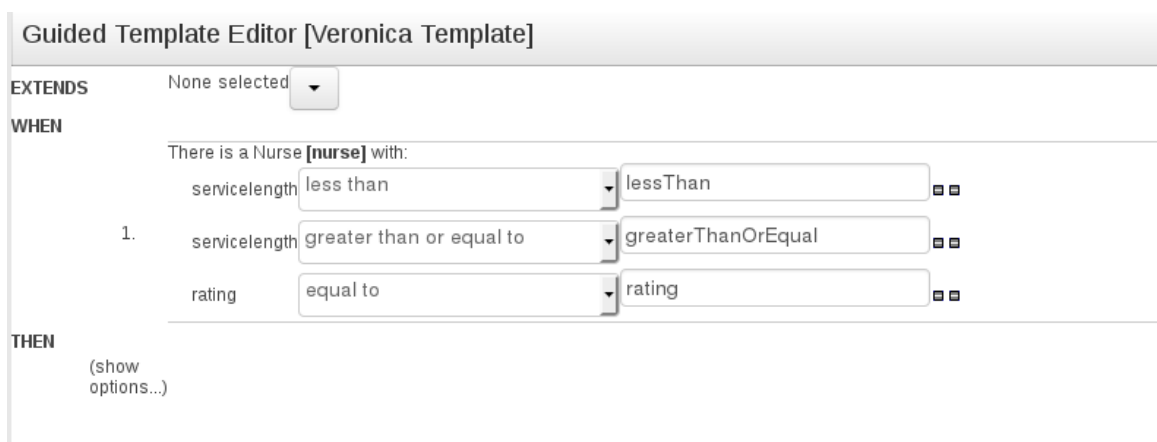



Figure 6.32. WHEN Constraints

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.

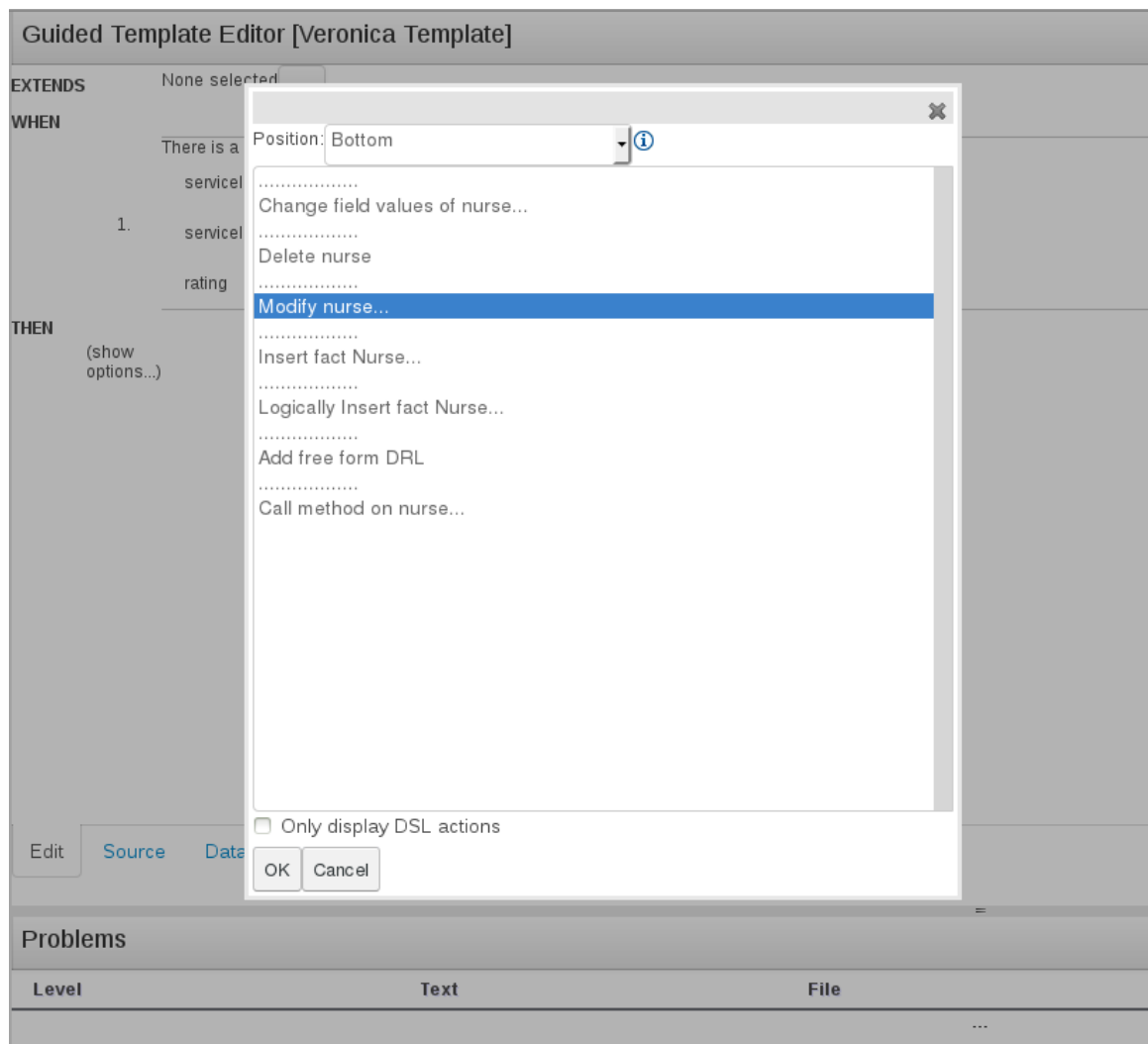



Figure 6.33. Nurse Roster THEN Dialog Window

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.

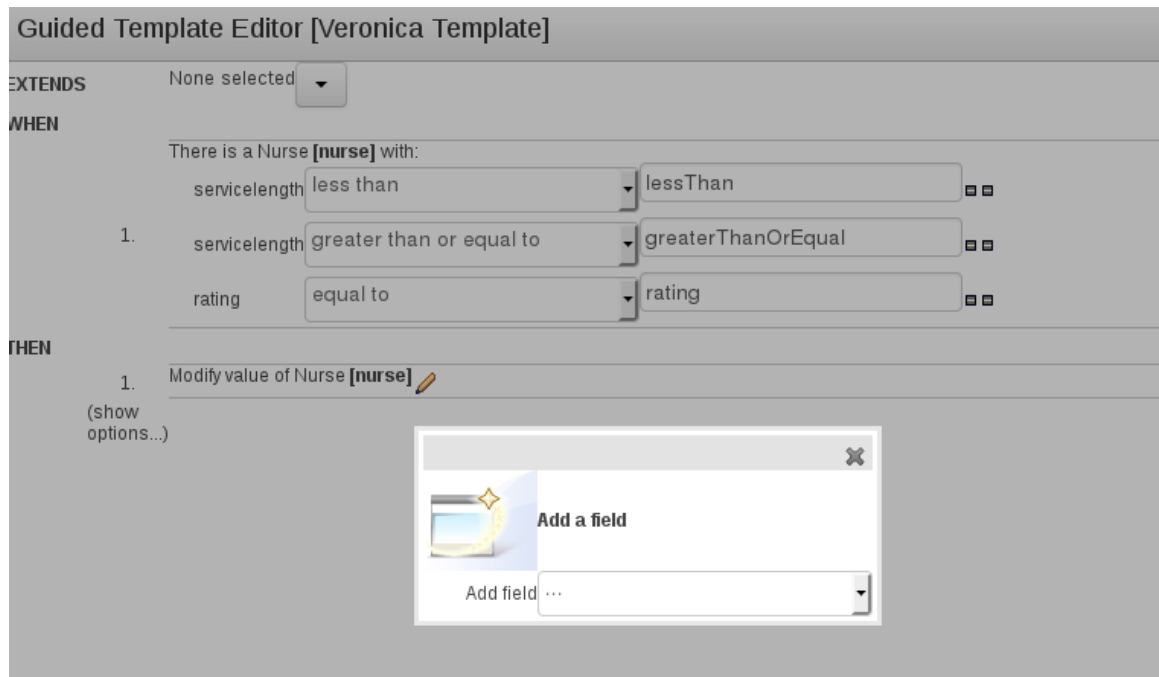



Figure 6.34. Add a field Dialog

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.

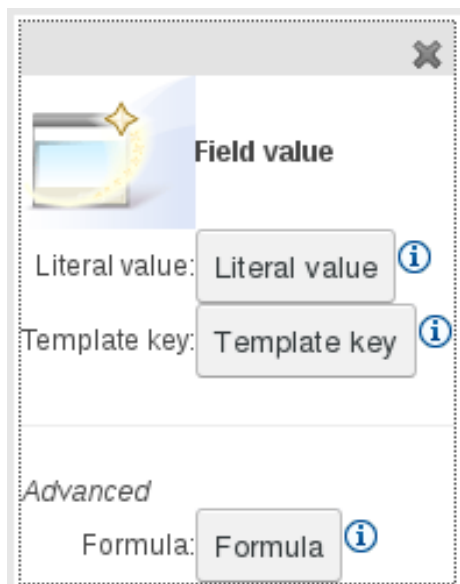
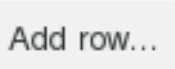


Figure 6.35. Field Value Options

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.

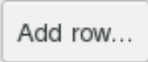











| Guided Template Editor [Veronica Template] | | | | |
|-------------------------------------------------------------------------------------|-----------------|--------------------|----------------|--------------|
|  | | | | |
|  | ServiceLessThan | ServiceGreaterThan | EmployeeRating | VacationTime |
|  | | | | |
|  | | | | |
|  | | | | |
|  | | | | |
|  | | | | |
|  | | | | |
|  | | | | |
|  | | | | |
|  | | | | |
|  | | | | |

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.

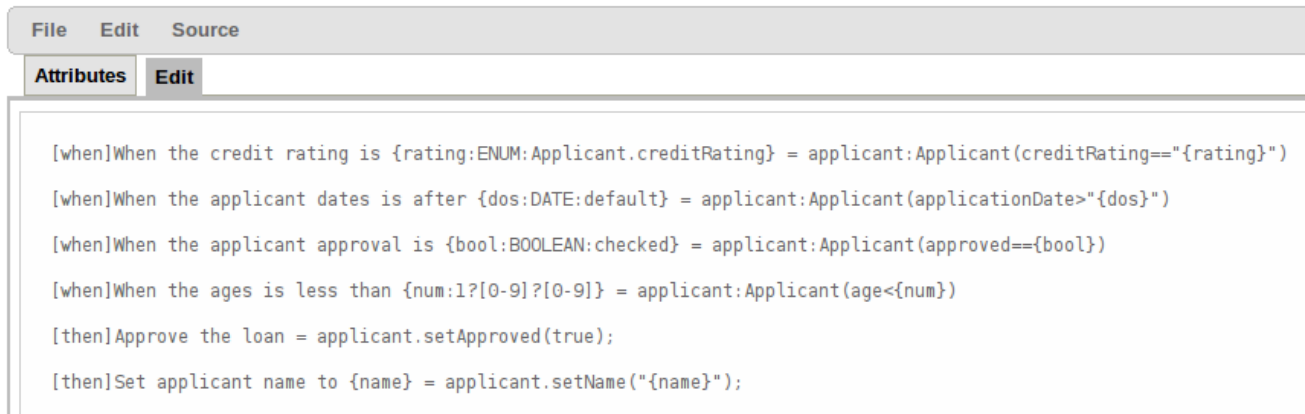


Figure 6.38. DSL Editor

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 allow 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:
 - o If in the **Project** view of Project Explorer, select the organizational unit, repository and the project where you want to create the score card.
 - o 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.

6.9. VERIFICATION AND VALIDATION OF GUIDED DECISION TABLES

In Business Central, the guided decision tables are a way of representing conditional logic (rule) in a precise manner, and they are well suited to business level rules.

6.9.1. Introduction

Business Central provides verification and validation feature to help ensure that a guided decision table is complete and error free. The feature looks for gaps in the logic of a guided decision table and validates the relationship between different cells. Most of the issues that the Business Central verification and validation feature reports are gaps in the author's logic. The verification and validation feature does not prevent you from saving your work if you choose to ignore the verification and validation notifications.

When you edit a guided decision table, the verification and validation feature notifies you if you do something wrong. For example, if you forget to set an action for a row of the guided decision table or if you have a duplicate row, a new panel, Analysis, will open in the Business Central with a notification about the issue. The Analysis panel presents a list of issues found in the guided decision table, each item also pointing out the affected guided decision table rows. If you select an item in the list of issues, you will see a more in-depth description of the problem.

The validation and verification feature helps you resolve issues around:

- Redundancy

Redundancy happens when two rows in a decision table execute the same consequences for the same set of facts. For example, two rows checking a client's birthday and providing a birthday discount may result in double discount.

- Subsumption

Subsumption is similar to redundancy and exists when two rules execute the same consequences, but one executes on a subset of facts of the other. For example, these two rules:

- when Person age > 10 then Increase Counter
- when Person age > 20 then Increase Counter

In this case, if a person is 15 years old, only one rule fires and if a person is 20 years old, both rules fire. Such cases cause similar trouble during runtime as redundancy.

- Conflicts

A conflicting situation occurs when two similar conditions have different consequences. Sometimes, there may be conflicts between two rows (rules) or two cells in a decision table.

The example below illustrates conflict between two rows in a decision table:

- when Deposit > 20000 then Approve Loan
- when Deposit > 20000 then Refuse Loan

In this case, there is no way to know if the loan will be approved or not.

The example below illustrates conflict between two cells in a decision table:

- When Age > 25
- When Age < 25

A row with conflicting cells never execute.

- Deficiency

Deficiency is similar to a conflict and occurs due to incompleteness in the logic of a rule in a decision table. For example, consider the following two deficient rules:

- when Age > 20 then Approve Loan
- when Deposit < 20000 then Refuse Loan




These two rules may lead to confusion for a person who is over 20 years old and have deposited less than 20000. You may add more constraints to avoid the conflict after noticing the warning to make your rule logic complete.

- **Missing Columns**

In some cases, usually by accident, the user can delete all the condition or action columns. When the conditions are removed all the actions are executed and when the actions columns are missing the rows do nothing. Both may or may not be by design, usually though this is a mistake.

6.9.2. Reporting

The verification and validation feature of Business Central reports different issue levels in the Analysis panel while you are updating a guided decision table.

-  **Error:** This means a serious fault, which may lead to the guided decision table failing to work as designed at run time. Conflicts, for example, are reported as errors.
-  **Warning:** This is most likely a serious fault, however they may not prevent the guided decision table from working but need to be double checked. Warnings are used e.g. for subsumption.
-  **Information:** This kind of issues might be a design decision of the author of the guided decision table as well as a simple accident. This is used e.g. for a row missing any condition.



NOTE

Business Central verification and validation does not prevent you from saving an incorrect change. The feature only reports issues while editing and you can still continue to overlook those and save your changes.

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:

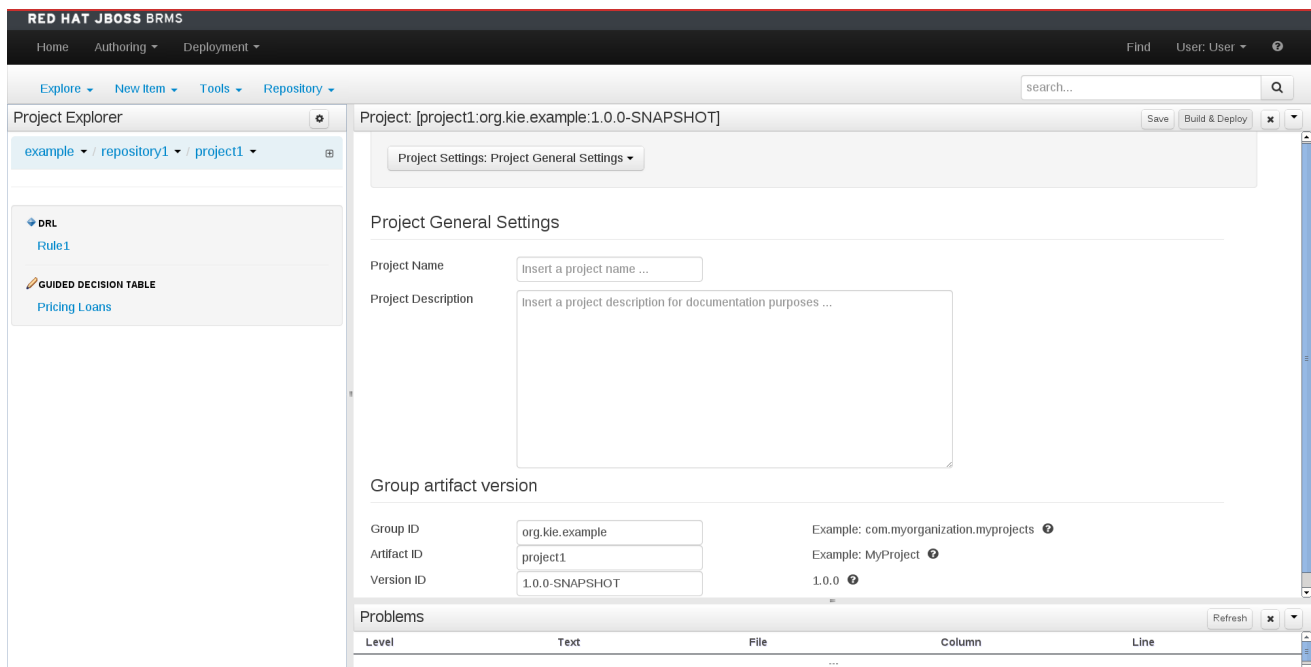


Figure 7.1. Build & Deploy Option

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. Similarly, 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.

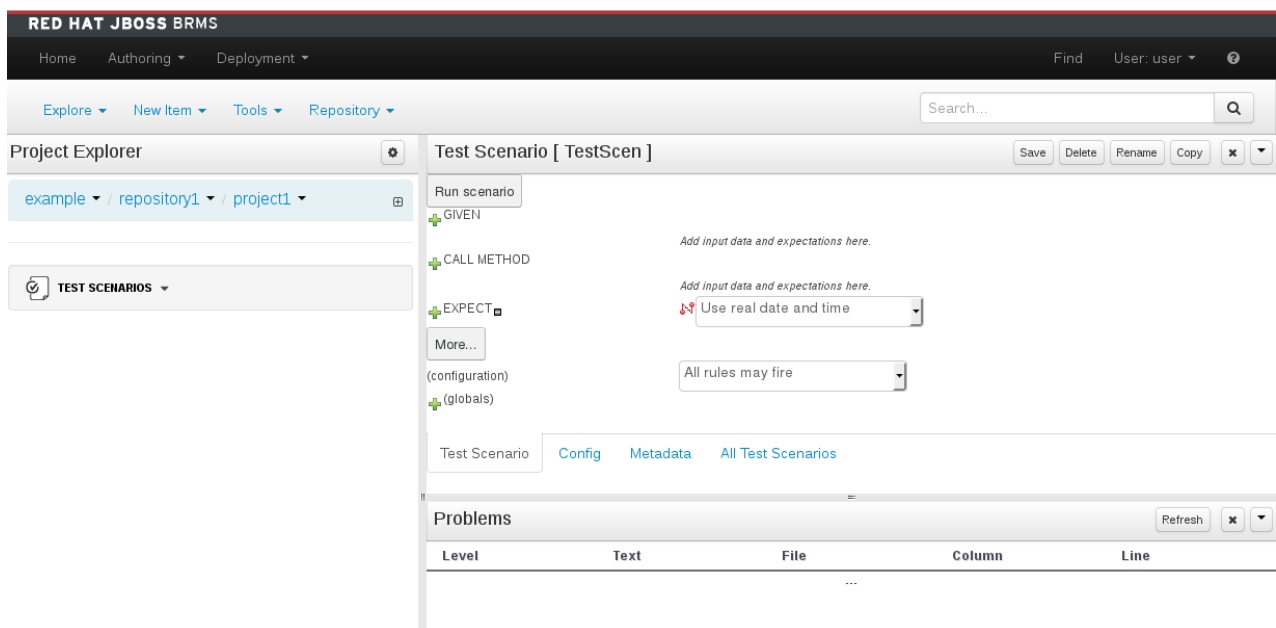


Figure 9.1. Test Scenario Screen

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.



New expectation


Rule:

Fact value:

Any fact that matches:

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

- o 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.
- o 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

Product 'product1' has values: 

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

- o 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**.



Test Scenario Config Metadata All Test Scenarios

Reporting

There were test failures


Text

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:

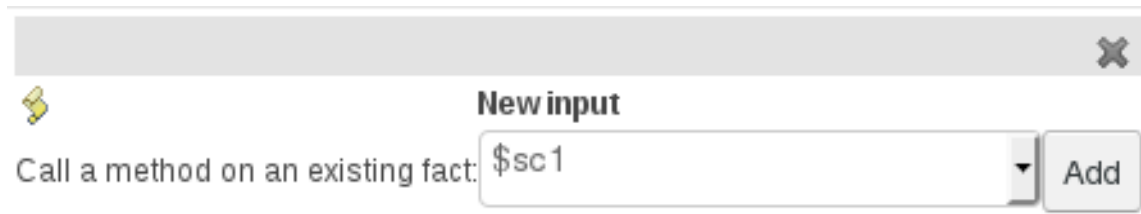



Figure 9.2. Call Method

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.

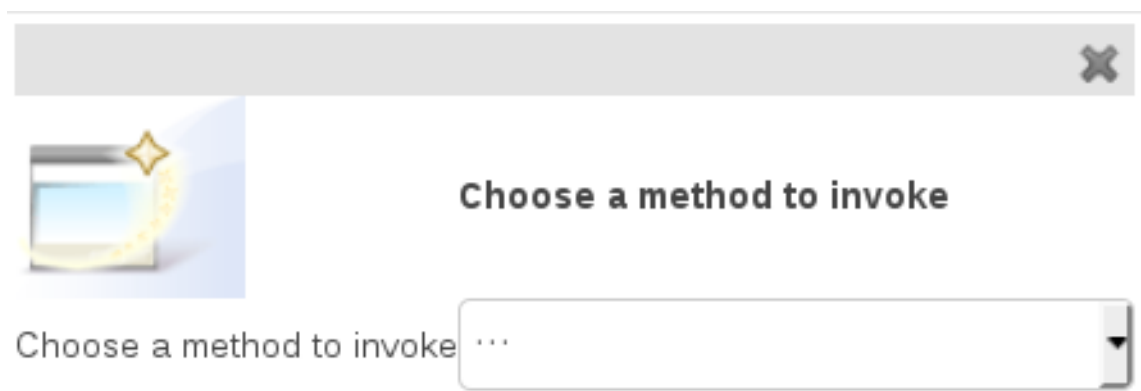



Figure 9.3. Invoke a Method

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.

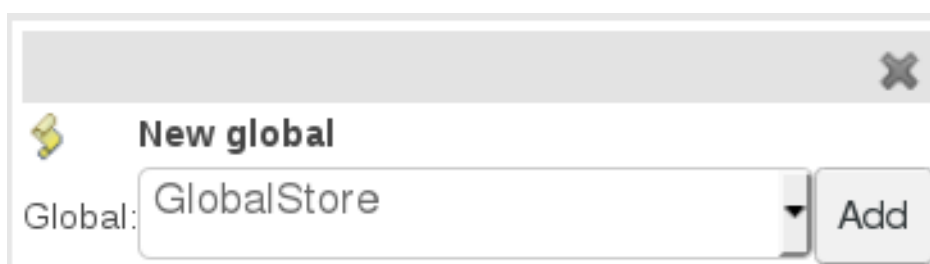


Figure 9.4. New Global

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.

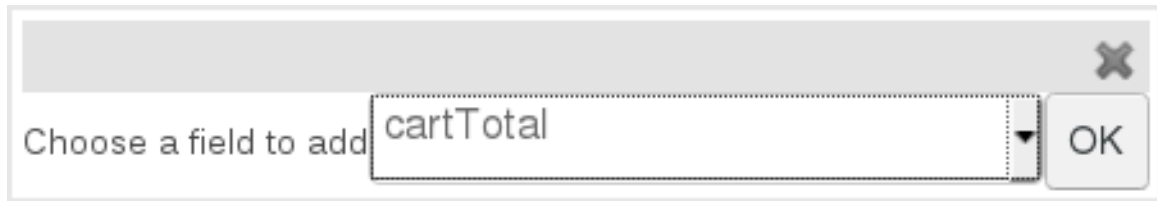



Figure 9.5. Apply Field

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.

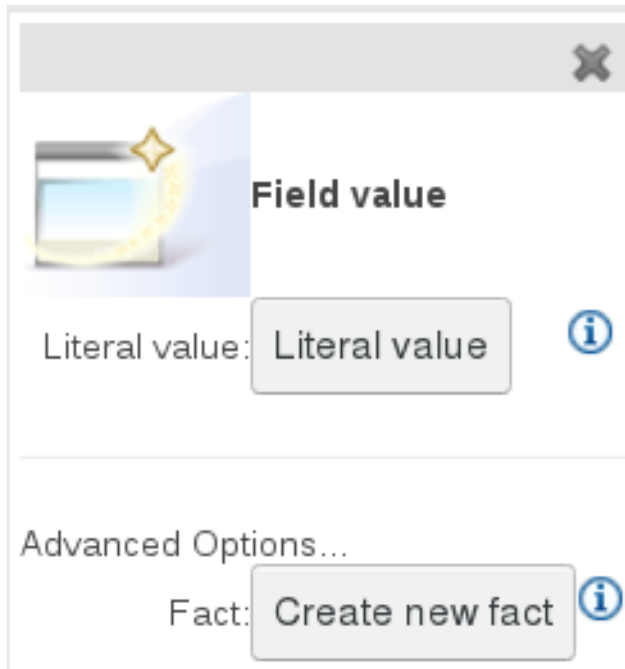


Figure 9.6. Field Values

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.

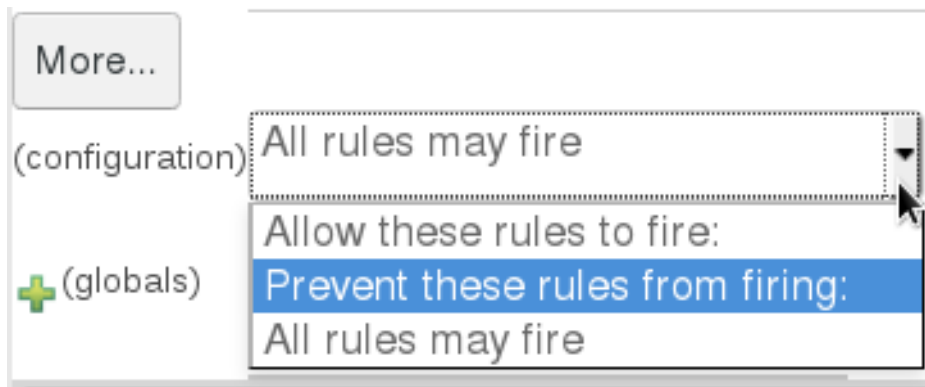



Figure 9.7. Configuration

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.

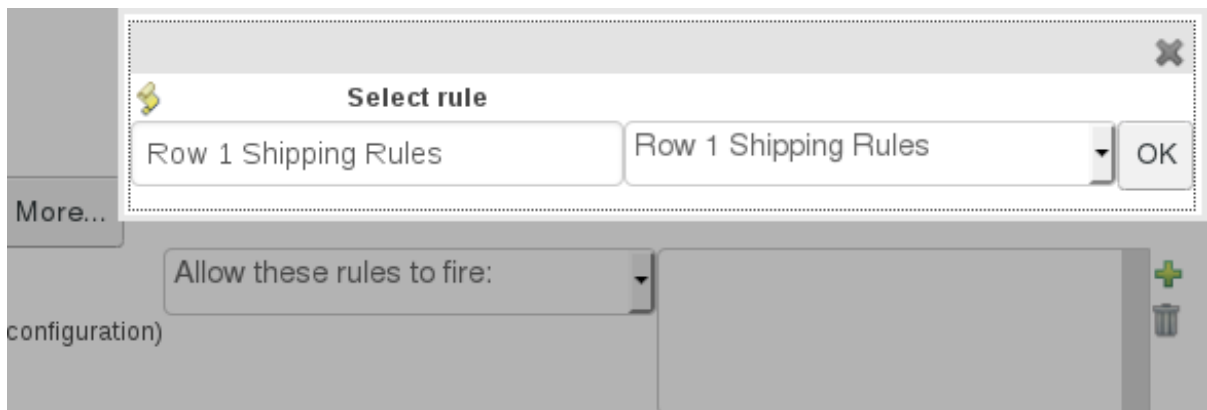


Figure 9.8. Selecting rules

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.

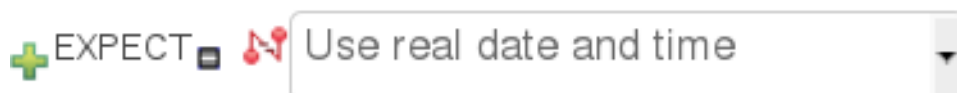


Figure 9.9. Real Date and Time

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.

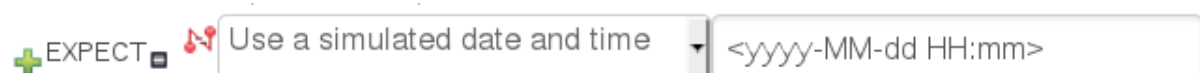


Figure 9.10. Title

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.

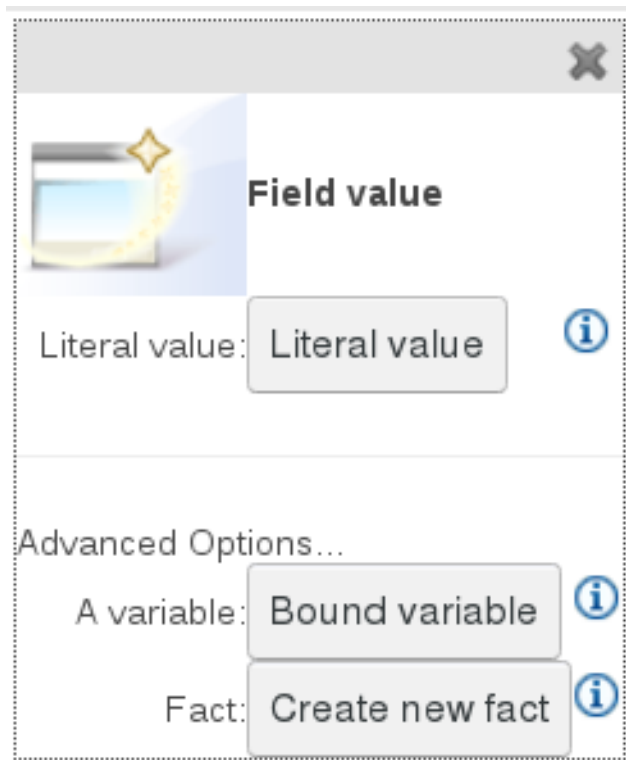


Figure 9.11. Advanced Options

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.

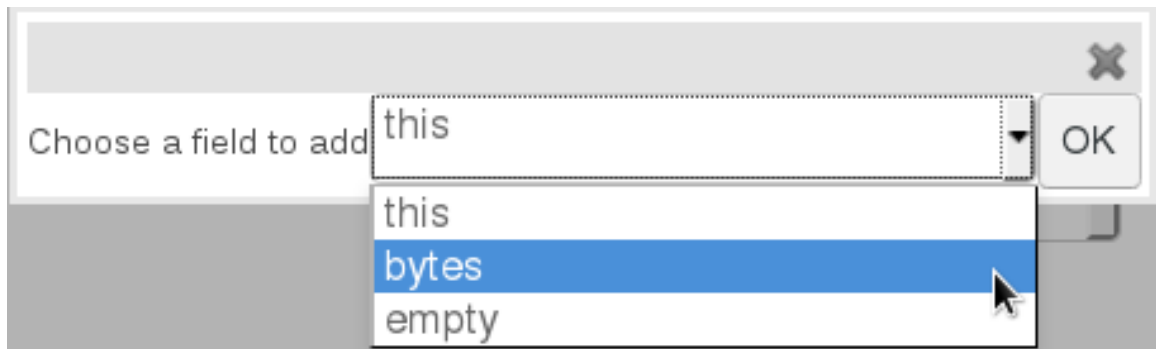


Figure 9.13. Create New Fact

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.

| | | |
|--------------------------|------------------------------------|------------------------------------|
| Modify an existing fact: | <input type="text" value="\$sc1"/> | <input type="button" value="Add"/> |
| Delete an existing fact: | <input type="text" value="\$sc9"/> | <input type="button" value="Add"/> |

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 through 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. The current version of the Realtime Decision Server ships with default extensions for both JBoss BRMS and JBoss BPM Suite.

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 Realtime Decision Server instances through Business Central. In this chapter, you will go through the steps required to setup a Realtime Decision Server, provision and connect to this server through 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`. When you install JBoss BRMS, this WAR file is installed and deployed in your web container by default. Consequently, this gives you automatic access to this web application with the URL of `http://SERVER:PORT/kie-server/`.

You can 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.

1. Once you have deployed the 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:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response type="SUCCESS" msg="Kie Server info">
  <kie-server-info>
    <capabilities>BPM</capabilities>
    <capabilities>KieServer</capabilities>
    <capabilities>BRM</capabilities>
    <location>http://localhost:8230/kie-
server/services/rest/server</location>
    <name>KieServer@/kie-server</name>
    <id>15ad5bfa-7532-3eea-940a-abbbbc89f1e8</id>
    <version>6.3.0.Final-redhat-5</version>
  </kie-server-info>
</response>
```

10.2. INSTALLING THE REALTIME DECISION SERVER IN OTHER CONTAINERS

10.2.1. JBoss Web Server 2.x/3.x, Tomcat 7.x/8.x

Use the following procedure to install the Realtime Decision Server in a Tomcat container.

Procedure 10.1.

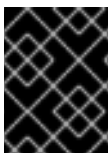
1. Download and unzip the Tomcat distribution.
2. Download `kie-server-6.3.0.Final-webc.war` and place it into `TOMCAT_HOME/webapps`.
3. Configure user(s) and role(s). Make sure that `TOMCAT_HOME/conf/tomcat-users.xml` contains the following username and role definition. The username and password should be unique, however ensure that the user has the role `kie-server`:

```
<role rolename="kie-server"/>
<user username="serveruser" password="my.s3cr3t.pass" roles="kie-
server"/>
```

4. Start the server by running `TOMCAT_HOME/bin/startup. [sh|bat]`. You can check out the Tomcat logs in `TOMCAT_HOME/logs` to see if the application deployed successfully. See [Section 10.3.1, “Bootstrap Switches”](#) for the bootstrap switches that can be used to properly configure the instance. For instance:

```
$ ./startup.sh -Dorg.kie.server.id=first-kie-server -
Dorg.kie.server.location=http://localhost:8080/kie-
server/services/rest/server
```

5. Verify the server is running. Access `http://SERVER:PORT/kie-server/services/rest/server/` and type the specified username and password. You should see simple XML message with basic information about the server.



IMPORTANT

You can not leverage the JMS interface when running on Tomcat, or any other Web container. The Web container version of the WAR contains only the REST interface.

10.3. REALTIME DECISION SERVER SETUP

10.3.1. Bootstrap Switches

The Realtime Decision Server accepts a number of bootstrap switches (system properties) to configure the behaviour of the server. The following is a table of all the supported switches.

Note that some properties are marked as required when using a controller. Thus, you need to set these properties when you handle Realtime Decision Server, container creation, and removal in Business Central. If you use the Realtime Decision Server separately, without any interaction with Business Central, you do not have to set these properties.

Table 10.1. Realtime Decision Server Bootstrap Switches

| Property | Value | Description | Required |
|--------------------------------|-----------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------------------------------------|
| org.drools.server.ext.disabled | boolean (default is "false") | If true, disables the BRM support (i.e. rules support). | No |
| org.jbpm.server.ext.disabled | boolean (default is "false") | If true, disables the BPM support (i.e. processes support) | No |
| org.kie.server.id | string | An arbitrary ID to be assigned to this server. If a remote controller is configured, this is the ID under which the server will connect to the controller to fetch the kie container configurations. | No. If not provided, an ID is automatically generated. |
| org.kie.server.user | string (default is "kieserver") | User name used to connect with the kieserver from the controller, required when running in managed mode | Yes when using a controller. This property needs to be set in Business Central system properties. |
| org.kie.server.pwd | string (default is "kieserver1!") | Password used to connect with the kieserver from the controller, required when running in managed mode | Yes when using a controller. This property needs to be set in Business Central system properties. |
| org.kie.server.controller | comma separated list of urls | List of urls to controller REST endpoint. E.g.: http://localhost:8080/kie-wb/rest/controller | Yes when using a controller |
| org.kie.server.controller.user | string (default is "kieserver") | Username used to connect to the controller REST api | Yes when using a controller |
| org.kie.server.controller.pwd | string (default is "kieserver1!") | Password used to connect to the controller REST api | Yes when using a controller |

| Property | Value | Description | Required |
|------------------------------------|---------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------|------------------------------|
| org.kie.server.location | URL location of Realtime Decision Server instance | The URL used by the controller to call back on this server. E.g.: <code>http://localhost:8230/kie-server/services/rest/server</code> | Yes when using a controller |
| org.kie.server.domain | string | JAAS LoginContext domain that shall be used to authenticate users when using JMS | No |
| org.kie.server.bypass.auth.user | boolean (default is "false") | Allows to bypass the authenticated user for task related operations e.g. queries | No |
| org.kie.server.repo | valid file system path (default is ".") | Location on local file system where Realtime Decision Server state files will be stored | No |
| org.kie.server.persistence.ds | string | DataSource JNDI name | Yes when BPM support enabled |
| org.kie.server.persistence.tm | string | Transaction manager platform for Hibernate properties set | Yes when BPM support enabled |
| org.kie.server.persistence.dialect | string | Hibernate dialect to be used | Yes when BPM support enabled |
| org.jbpm.ht.callback | string | One of supported callbacks for Task Service (default jaas) | No |
| org.jbpm.ht.custom.callback | string | Custom implementation of UserGroupCallback in case <code>org.jbpm.ht.callback</code> was set to 'custom' | No |
| kie.maven.settings.custom | valid file system path | Location of custom <code>settings.xml</code> for maven configuration | No |

| Property | Value | Description | Required |
|------------------------------------------------|-------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|----------|
| <code>org.kie.executor.interval</code> | integer (default is 3) | Number of time units between polls by executor | No |
| <code>org.kie.executor.pool.size</code> | integer (default is 1) | Number of threads in the pool for async work | No |
| <code>org.kie.executor.retry.count</code> | integer (default is 3) | Number of retries to handle errors | No |
| <code>org.kie.executor.timeunit</code> | TimeUnit (default is "SECONDS") | TimeUnit representing interval | No |
| <code>org.kie.executor.disabled</code> | boolean (default is "false") | Disables executor completely | No |
| <code>kie.server.jms.queues.response</code> | string (default is "queue/KIE.SERVER.RESPONSE") | JNDI name of response queue for JMS | No |
| <code>org.kie.server.controller.connect</code> | long (default is 10000) | Waiting time in milliseconds between repeated attempts to connect Realtime Decision Server to controller when Realtime Decision Server starts up | No |
| <code>org.drools.server.filter.classes</code> | boolean (default is "false") | If set to true then Drools Realtime Decision Server extension accepts custom classes being annotated by <code>XmlRootElement</code> or <code>Remotable</code> annotations only. | No |



IMPORTANT

A new `DataSource` for the Realtime Decision Server must point to a different database schema than the `DataSource` used by Business Central.

If you are running both the Realtime Decision Server and the Business Central, make sure that each uses a different `DataSource` (by modifying the `org.kie.server.persistence.ds` property), in order to avoid conflicts.

10.3.2. Managed Realtime Decision Server

A managed instance requires an available controller to start up the Realtime Decision Server.

A Controller is a component that keeps and manages a Realtime Decision Server configuration in a centralized way. Each controller can manage multiple configurations at once, and there can be multiple controllers in the environment. Managed Realtime Decision Server can be configured with a list of controllers, but will only connect to one at a time.



IMPORTANT

Controllers should be kept in sync to ensure that the same set of configuration is provided to the server, regardless of which controller it connects to.

When the Realtime Decision Server is configured with a list of controllers, it will attempt to connect to each of them at startup until a connection is successfully established with one of them. If for some reason a connection cannot be established, the server will not start, even if there is a local storage available with configuration. This ensures consistence, and prevents the server from running with redundant configuration.



NOTE

To run the Realtime Decision Server in standalone mode without connecting to controllers, see [Section 10.3.4, “Unmanaged Realtime Decision Server”](#).

10.3.3. Registering a Realtime Decision Server

To register a new managed Realtime Decision Server instance, set the following properties in `standalone.xml`:

```
<property name="org.kie.server.user" value="anton"></property>
<property name="org.kie.server.pwd" value="password1!"></property>
<property name="org.kie.server.location" value="http://localhost:8080/kie-
server/services/rest/server"></property>
<property name="org.kie.server.controller"
value="http://localhost:8080/business-central/rest/controller"></property>
<property name="org.kie.server.controller.user"
value="kieserver"></property>
<property name="org.kie.server.controller.pwd"
value="kieserver1!"></property>
<property name="org.kie.server.id" value="local-server-123"></property>
```

The `standalone.xml` file is located in the `$SERVER_HOME/standalone/configuration/` directory.

`org.kie.server.user` must have the `kie-server` role assigned.



NOTE

`org.kie.server.controller.user` and `org.kie.server.controller.pwd` must be set to `kieserver` and `kieserver1!` respectively due to a known issue in the 6.2.0 release.

**WARNING**

`org.kie.server.location` points to the same host as `org.kie.server.controller`. This is not suitable for production.

To create the `kieserver` user:

- Change into `$SERVER_HOME/bin/`.
- Execute the following command:

```
$ ./add-user.sh -a --user kieserver --password kieserver1! --role kie-server
```

To verify successful start of the Realtime Decision Server, send a GET request to `http://SERVER:PORT/kie-server/services/rest/server/`. Once authenticated, you will see the following XML response:

```
<response type="SUCCESS" msg="Kie Server info">
  <kie-server-info>
    <capabilities>KieServer</capabilities>
    <capabilities>BRM</capabilities>
    <capabilities>BPM</capabilities>
    <location>
      http://localhost:8080/kie-server/services/rest/server
    </location>
    <name>local-server-123</name>
    <id>local-server-123</id>
    <version>6.3.0.Final-redhat-5</version>
  </kie-server-info>
</response>
```

To verify successful registration, log into the Business Central and select **Deploy → Rule Deployments**. If successful, you will see the registered server ID.

10.3.4. Unmanaged Realtime Decision Server

An unmanaged Realtime Decision Server is a standalone instance, and therefore must be configured individually using REST/JMS API from the Realtime Decision Server itself. There is no controller involved. The configuration is automatically persisted by the server into a file and that is used as the internal server state, in case of restarts.

The configuration is updated during the following operations:

- Deploy KIE Container
- Undeploy KIE Container
- Start KIE Container

- Stop KIE Container




NOTE

If the Realtime Decision Server is restarted, it will attempt to re-establish the same state that was persisted before shutdown. Therefore, KIE Containers that were running will be started, but the ones that were stopped will not.

10.4. CREATING A CONTAINER

Once your Realtime 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 Realtime Decision Server. This will bring up the Create Container... dialogue.

You can enter the Artifact Id and Version (GAV) of your deployed package manually. Finish by providing a name for the Container and clicking the **Ok** button to select that instance.

You can also search Business Central for all packages that can be deployed. Click **Search** 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).

Create Container...



| Name | Path | Last Modifie | Open | Select |
|--------------|--------------|--------------|-------------------------------------|---------------------------------------|
| MyProject... | org/bpms/... | 2016 Jan ... | <input type="button" value="Open"/> | <input type="button" value="Select"/> |
| MyProject... | org/bpms/... | 2016 Jan ... | <input type="button" value="Open"/> | <input type="button" value="Select"/> |
| MvProject... | org/bpms/... | 2016 Jan ... | <input type="button" value="Open"/> | <input type="button" value="Select"/> |


The figure above shows that there are three deployable packages available to be used as containers on the Realtime Decision Server. Select the one that you want by clicking **Select**.

Enter a name for this Container and click **Ok**.


10.5. MANAGING CONTAINERS

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

Starting a Container

Once registered, a Container is in the 'Stopped' mode. Click  to select the Container, then click **Start**. You can select multiple Containers and start them all at the same time.

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

In case of errors,  appears next to Container(s) and the Realtime Decision Server that they are deployed on. For troubleshooting, check the logs of both the Realtime Decision Server and the Business Central before redeploying the Containers.


Stopping and Deleting a Container

Click  to select the running Container(s) and **Stop** or **Delete**.

Updating Container Deployments

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

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

To update deployments in a Container dynamically, click  next to the Container. This opens up the Container Info screen:


Container Info [MyContainer]

Interval

Start Scanner

Stop Scanner

Scan Now



Endpoints

| Endpoint | Status |
|---------------------------------------------------------------------------------------------------------------------------|--------|
| http://localhost:8080/kie-server/services/rest/server | UP |

Release Id

Group Id

Artifact Id

Version

Upgrade

org.brms

MyProject

1.0.0

Resolved Release Id

Group Id

Artifact Id

Version

The Container Info screen allows you to see the *endpoint* for the Container (for example, <http://localhost:8080/kie-server/services/rest/server>). You can also either manually or automatically refresh the Container if an update is available.

1. **Manual Update:** To manually update a provision, enter the new Version number in the Version box and click **Update**. You can update the Group Id or the Artifact Id, if these have changed as well.

2. Automatic Update: If you want a deployed Container to always have the latest version of your deployment without manually editing it, you need to set the Version property to the value of **LATEST** and start a Scanner. This will ensure that the deployed provision always contains the latest version. The Scanner can be started on demand by clicking **Scan Now** 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** will show you the actual, latest version number. Note that if the **Resolved Release Id** is empty, this is an issue not influencing the functionality of the upgrade feature.

10.6. THE REST API FOR REALTIME DECISION SERVER EXECUTION

You can communicate with the Realtime Decision Server through the REST API.

- The base URL for sending requests is the endpoint defined earlier (*http://SERVER:PORT/kie-server/services/rest/server/*).
- All requests require basic HTTP Authentication for the role kie-server.

Following methods support three formats of the requests: JSON, JAXB, and XSTREAM. You must provide following HTTP headers:

- Accept: application/json or application/xml
- Content-Type: application/json or application/xml

If you want to define the marshaller, provide also **X-KIE-ContentType**: has three possible values - JSON, JAXB, or XSTREAM. Set it according to the format of your request to the server.

The examples use the Curl utility. You can use any REST client. Curl commands use the following parameters:

- **-u**: specifies username:password for the Realtime Decision Server authentication.
- **-H**: specifies HTTP headers.
- **-X**: specifies the HTTP method of the request, that is [GET], [POST], [PUT], or [DELETE].



NOTE

BRMS Commands endpoints will work only if your Realtime Decision Server has BRM capability. The rest of the endpoints will work only if your Realtime Decision Server has BPM capabilities. Check the following URI for capabilities of your Realtime Decision Server: *http://SERVER:PORT/kie-server/services/rest/server*.

10.6.1. BRMS Commands

Table 10.2. BRMS Commands Endpoints

| URI | Method | Request Type | Response Type | Description |
|-------------------------------------|--------|------------------------------------------------------------------------------------------------------------------------------------------------|---------------------------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------|
| /containers/instances/{containerID} | POST | A single <code>org.drools.core.command.impl.GenericCommand</code> command or multiples commands in <code>BatchExecutionCommand</code> wrapper. | <code>org.kie.server.api.model.ServiceResponse<String></code> | Executes the commands sent to the specified containerId and returns the commands execution results. For more information, See supported commands below. |

List of supported commands:

- `AgendaGroupSetFocusCommand`
- `ClearActivationGroupCommand`
- `ClearAgendaCommand`
- `ClearAgendaGroupCommand`
- `ClearRuleFlowGroupCommand`
- `DeleteCommand`
- `InsertObjectCommand`
- `ModifyCommand`
- `GetObjectCommand`
- `InsertElementsCommand`
- `FireAllRulesCommand`
- `QueryCommand`
- `SetGlobalCommand`
- `GetGlobalCommand`
- `GetObjectsCommand`
- `BatchExecutionCommand`

For more information about the commands, see the `org.drools.core.command.runtime` package. Alternatively, see [Supported JBoss BRMS Commands](#) from the Red Hat JBoss Development Guide.

Example 10.1. [POST] Drools Commands Execution

1. Change into a directory of your choice and create `commandsRequest.json`:

```
{
  "lookup" : null,
  "commands" : [ {
    "insert" : {
      "object" : "testing",
      "disconnected" : false,
      "out-identifier" : null,
      "return-object" : true,
      "entry-point" : "DEFAULT"
    }
  }, {
    "fire-all-rules" : { }
  } ]
}
```

2. Execute the following command:

```
$ curl -X POST -H 'X-KIE-ContentType: JSON' -H 'Content-type:
application/json' -u 'kieserver:kieserver1!' --data
@commandsRequest.json http://localhost:8080/kie-
server/services/rest/server/containers/instances/myContainer
```

The command generates a request that sends the Insert Object and Fire All Rules commands to the server. **Lookup** specifies a ksession configured in your kjar. If you use a null lookup value, the default KIE session will be used.

An example response:

```
{
  "type" : "SUCCESS",
  "msg" : "Container hello successfully called.",
  "result" : "{\n  \"results\" : [ ],\n  \"facts\" : [ ]\n}"
}
```

10.6.2. Managing Processes

Use the following entry point: `server/containers/{containerId}/processes`. See table Process Management Endpoints for a list of endpoints.

Table 10.3. Process Management Endpoints

| URI | Method | Request Type | Response Type | Description |
|------------|--------|--------------|---------------|----------------------------------------------------------------------------------------|
| /instances | DELETE | N/A | N/A | Aborts multiple process instances specified by the query parameter instanceId . |

| URI | Method | Request Type | Response Type | Description |
|---------------------------------------------------|--------|--------------------------------------|----------------------------------------------------|----------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| /instances/{processInstanceId}/signals | GET | N/A | A list of Strings | Returns all the available signal names for processInstanceId as a list of Strings |
| /instances/{processInstanceId}/variable/{varName} | PUT | The variable marshalled value | N/A | Sets the value of variable varName for process instance processInstanceId . If successful, the return value is HTTP code 201. |
| /instances/{processInstanceId}/variable/{varName} | GET | N/A | The variable value | Returns the marshalled value of varName for processInstanceId . |
| /instances/{processInstanceId}/variables | POST | A map with variable names and values | N/A | Sets multiple processInstanceId variables. The request is a map, in which the key is the name of the variable and the value is the new value of the variable. |
| /instances/{processInstanceId}/variables | GET | N/A | A map with the variable names and values | Gets all variables for processInstanceId as a map, in which the key is the name of the variable and the value is the value of the variable |
| /instances/{processInstanceId}/workitems/ | GET | N/A | A list of WorkItemInstance objects | Gets all the work items of the given processInstanceId . |

| URI | Method | Request Type | Response Type | Description |
|-----------------------------------------------------------------|--------|-------------------------------------------------|-------------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| /instances/{processInstanceId}/workitems/{workItemId} | GET | N/A | A WorkItemInstance object | Gets the workItemId work item of the given processInstanceId . |
| /instances/{processInstanceId}/workitems/{workItemId}/aborted | PUT | N/A | N/A | Aborts the workItemId work item of the given processInstanceId . If successful, the return value is HTTP code 201. |
| /instances/{processInstanceId}/workitems/{workItemId}/completed | PUT | N/A | N/A | Completes the workItemId work item of the given processInstanceId . If successful, the return value is HTTP code 201. |
| /processId/instances | POST | A map with variables used to start the process. | Plain text with the process instance id. | Creates a business process instance specified by processId . Accepted input is a map with the process variables and its values. |
| instances/signal/{signalName} | POST | A marshalled object | N/A | Signals multiple process instances of a query parameter instanceId with signal signalName . You can provide the signal payload marshalled in the request body. |

| URI | Method | Request Type | Response Type | Description |
|----------------------------------------------------|--------|-------------------------------------------------|------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| instances/{processInstanceId} | DELETE | N/A | N/A | Aborts a process specified by processInstanceId . If successful, the return value is HTTP code 204. |
| instances/{processInstanceId} | GET | N/A | A ProcessInstance object | Returns the details of processInstanceId . You can request variable information by setting the withVars parameter as true. |
| instances/{processInstanceId}/signal/{signalName} | POST | A marshalled object | N/A | Signals the process instance processInstanceId with signal signalName . You can provide the signal payload marshalled in the request body. |
| {processId}/instances/correlation/{correlationKey} | POST | A map with variables used to start the process. | Plain text with the process instance id. | Creates a business process instance specified by the processId with the correlation key correlationKey . Accepted input is a map with the process variables and its values. |

Example 10.2. Managing Processes

- Create `person.js`:

```

    {
      "p" : {
        "org.kieserver.test.Person": { "id" : 13, "name":
"William" }
      }
    }

```

Start a process using a custom object (Person) as a parameter:

```
$ curl -X POST -u 'kieserver:kieserver1!' -H 'Content-type: application/json' -H 'X-KIE-ContentType: JSON' --data @person.js 'http://localhost:8080/kieserver/services/rest/server/containers/person/processes/proc-with-pojo.p-proc/instances'
```

- Create a new process instance of process definition **com.sample.rewards-basic** with parameters:

```
$ curl -X POST -u 'kieserver:kieserver1!' -H 'Content-type: application/json' -H 'X-KIE-ContentType: JSON' --data '{"employeeName": "William"}' 'http://localhost:8080/kieserver/services/rest/server/containers/rewards/processes/com.sample.rewards-basic/instances'
```

Returns process instance ID.

- Get the variables of process instance 3

```
$ curl -u 'kieserver:kieserver1!' -H 'Accept: application/json' 'http://localhost:8080/kieserver/services/rest/server/containers/rewards/processes/instances/3/variables'
```

Example response:

```
{
  "employeeName" : "William"
}
```

- Send a TEST signal to the process instance with ID 5

```
$ curl -X POST -u 'kieserver:kieserver1!' -H 'Content-type: application/json' -H 'X-KIE-ContentType: JSON' --data '"SIGNAL DATA"' 'http://localhost:8080/kieserver/services/rest/server/containers/test/processes/instances/signal/TEST?instanceId=5'
```

10.6.3. Managing Process Definitions

Use the following entry point: *server/containers/{containerId}/processes/definitions*. See table Process Queries Endpoints for a list of endpoints. To use pagination, use the **page** and **pageSize** parameters.

Table 10.4. Process Queries Endpoints

| URI | Method | Request Type | Response Type | Description |
|-----|--------|--------------|---------------|-------------|
|-----|--------|--------------|---------------|-------------|

| URI | Method | Request Type | Response Type | Description |
|------------------------------|--------|--------------|--------------------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| / | GET | N/A | A list of ProcessDefinition objects | Returns a list of process definitions available for the container containerId |
| /({processId})/variables | GET | N/A | A VariablesDefinition object | Returns a map of the variable definitions for processId . The map contains the name of the variable and its type. |
| /({processId})/tasks/service | GET | N/A | A ServiceTaskDefinition object | Returns all service tasks for processId . The return value is a map with the names and types of the service tasks. If no tasks are found, the return value is an empty list. |
| /({processId})/tasks/users | GET | N/A | A list of UserTaskDefinition objects | Returns all the user tasks for processId . The response also contains maps of the input and output parameters. The key is the name and the value is the type of a parameter. |
| /({processId})/subprocesses | GET | N/A | A SubProcessDefinition object | Returns a list of reusable sub-process IDs for processId . |
| /({processId})/entities | GET | N/A | An AssociatedEntitiesDefinition object | Returns a map with the entities associated with processId . |

| URI | Method | Request Type | Response Type | Description |
|--------------------------------------------------------|--------|--------------|------------------------------------------------|--------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| <code>/processId/tasks/users/{taskName}/inputs</code> | GET | N/A | A TaskInputsDefinition object | Returns a map with all the task input parameter definitions for taskName of processId . The key is the name of the input and the value is its type. |
| <code>/processId/tasks/users/{taskName}/outputs</code> | GET | N/A | A TaskOutputsDefinition object | Returns a map with all the task output parameter definitions for taskName of processId . The key is the name of the input and the value is its type. |

Example 10.3. [GET] Process Definitions for a Specified Container

The following command displays process definitions for the **rewards** container and uses pagination:

```
$ curl -u 'kieserver:kieserver1!' -H 'accept: application/json'
'http://localhost:8080/kieserver/services/rest/server/queries/containers/instances/rewards/processes/definitions?page=0&pageSize=1000'
```

An example response:

```
{
  "processes" : [ {
    "process-id" : "com.sample.rewards-basic",
    "process-name" : "Rewards Basic",
    "process-version" : "1",
    "package" : "com.sample",
    "container-id" : "rewards"
  } ]
}
```

Example 10.4. [GET] User Tasks for a Specified Process

The following command displays user tasks for the **com.sample.rewards-basic** process in the **rewards** container:

```
$ curl -u 'kieserver:kieserver1!' -H 'accept: application/json'
'http://localhost:8080/kie-
```

```
server/services/rest/server/containers/instances/rewards/processes/definitions/com.sample.rewards-basic/tasks/users'
```

An example response:

```
{
  "task" : [ {
    "task-name" : "Approval by PM",
    "task-priority" : 0,
    "task-skippable" : false,
    "associated-entities" : [ "PM" ],
    "task-inputs" : {
      "Skippable" : "Object",
      "TaskName" : "java.lang.String",
      "GroupId" : "Object"
    },
    "task-outputs" : {
      "_approval" : "Boolean"
    }
  }, {
    "task-name" : "Approval by HR",
    "task-priority" : 0,
    "task-skippable" : false,
    "associated-entities" : [ "HR" ],
    "task-inputs" : {
      "Skippable" : "Object",
      "TaskName" : "java.lang.String",
      "GroupId" : "Object"
    },
    "task-outputs" : {
      "_approval" : "Boolean"
    }
  } ]
}
```

Example 10.5. [GET] Variable Definitions for Specified Process

The following command displays the variable definitions of the `com.sample.rewards-basic` process in the `rewards` container:

```
$ curl -u 'kieserver:kieserver1!' -H 'accept: application/json'
'http://localhost:8080/kie-
server/services/rest/server/containers/instances/rewards/processes/definitions/com.sample.rewards-basic/variables'
```

An example response:

```
{
  "variables" : {
    "result" : "String",
    "hrApproval" : "Boolean",
    "pmApproval" : "Boolean",
    "employeeName" : "String"
  }
}
```



10.6.4. Managing User Tasks

Use this base URI: `server/containers/{containerId}/tasks`.

10.6.4.1. Managing Task Instances

Use this base URI: `server/containers/{containerId}/tasks/{taskId}/states/`. If successful, the return value is HTTP code 201. See table Task Instance Endpoints for a list of endpoints.

Table 10.5. Task Instance Endpoints

| URI | Method | Request Type | Response Type | Description |
|-----------|--------|---------------------------------------------|---------------|-----------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| activated | PUT | N/A | N/A | Activates the taskId task. |
| claimed | PUT | N/A | N/A | Claims the taskId task. |
| started | PUT | N/A | N/A | Starts the taskId task. |
| stopped | PUT | N/A | N/A | Stops the taskId task. |
| completed | PUT | A map with the output parameters name/value | N/A | Completes the taskId task. You can provide the output parameters in form of a map, where the key is the parameter name and the value is the value of the output parameter. |
| delegated | PUT | N/A | N/A | Delegates the taskId task to a user provided by the targetUser query parameter. |
| exited | PUT | N/A | N/A | Exits the taskId task. |
| failed | PUT | N/A | N/A | Fails the taskId task. |

| URI | Method | Request Type | Response Type | Description |
|-----------|--------|--------------|---------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| forwarded | PUT | N/A | N/A | Forwards the taskId task to the user provided by the targetUser query parameter. |
| released | PUT | N/A | N/A | Releases the taskId task. |
| resumed | PUT | N/A | N/A | Resumes the taskId task. |
| skipped | PUT | N/A | N/A | Skips the taskId task. |
| suspended | PUT | N/A | N/A | Suspends the taskId task. |
| nominated | PUT | N/A | N/A | Nominates the taskId task to the potential owners by the potOwner query parameter. You can use the parameter multiple times (for example: potOwner=usr1&potOwner=usr2). |

Example 10.6. Task Instances

- Start task with **taskId** 4 in the container **test**

```
$ curl -X PUT -u 'kieserver:kieserver1!'
http://localhost:8080/kie-
server/services/rest/server/containers/test/tasks/4/states/started
```

- Complete the task 1 by passing an output parameter

```
$ curl -X PUT -u 'kieserver:kieserver1!' -H 'Content-type:
application/json' -H 'X-KIE-ContentType: JSON' --data '{
  "_approval" : true }' 'http://localhost:8080/kie-
server/services/rest/server/containers/test/tasks/1/states/complet
ed'
```



NOTE

Certain operations are illegal, for example starting a completed task. If you make such a request, you will receive one of the following errors:

Unexpected error during processing User '[UserImpl:'{USER ID}']' does not have permissions to execute operation OPERATION on task id {TASK ID}

Unexpected error during processing: User '[UserImpl:'{USER ID}']' was unable to execute operation OPERATION on task id {TASK ID} due to a no 'current status' match

To remove it, make sure the operation you are executing is allowed for the current task status. If you use `-Dorg.kie.server.bypass.auth.user=true`, you also must provide a user using the query parameter `user`, who has the permission to execute the operation. If you do not use the parameter, the logged user will be used to perform the action.

10.6.4.2. Managing Task Instance Data

Use this base URI: `server/containers/{containerId}/tasks/{taskId}`. See table Task Instance Data Management Endpoints for a list of endpoints.

Table 10.6. Task Instance Data Management Endpoints

| URI | Method | Request Type | Response Type | Description |
|-------------|--------|-------------------------------|---------------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| / | GET | N/A | A TaskInstance object | Gets task instance details of a taskId task. |
| attachments | POST | The content of the attachment | N/A | Adds a new attachment for the task. The ID of the created content is returned in the response, which is HTTP code 201. The name of the attachment is set using the query parameter name . It is not an idempotent method, thus if you make multiples request, it will create multiple attachements. |

| URI | Method | Request Type | Response Type | Description |
|------------------------------------|--------|--------------------------------------|--------------------------------------------------|---------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| attachments | GET | N/A | A list of TaskAttachment objects | Gets all task attachments for the task. |
| attachments/{attachmentId} | GET | N/A | A TaskAttachment object | Gets the attachmentId task attachment. |
| attachments/{attachmentId} | DELETE | N/A | N/A | Removes the attachmentId attachment. |
| attachments/{attachmentId}/content | GET | N/A | An attachment type object | Gets the attachmentId task attachment content. |
| comments | POST | A TaskComment object | Long | Adds a new comment for the task. The ID of the created content is returned in the response, which HTTP code is 201. It is not an idempotent method, thus if you make multiples request, it will create multiple comments. |
| comments | GET | N/A | A list of TaskComment objects | Gets all task comments for the task. |
| comments/{commentId} | GET | N/A | A TaskComment object | Gets a task comment identified by commentId . |
| comments/{commentId} | DELETE | N/A | N/A | Removes the commentId comment for the task. |

| URI | Method | Request Type | Response Type | Description |
|----------------------|--------|---------------------------------------------|---------------------------------------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------|
| contents/input | GET | N/A | A map with the input parameters name/value | Gets the task input content in form of a map, where the key is the parameter name and the value is the value of the output parameter. |
| contents/output | PUT | A map with the output parameters name/value | N/A | Updates the task output parameters and returns HTTP 201 if successful. You must provide the output parameters in form of a map, where the key is the parameter name and the value is the value of the output parameter. |
| contents/output | GET | N/A | A map with the output parameters name/value | Gets the task output content in form of a map, where the key is the parameter name and the value is the value of the output parameter. |
| contents/{contentId} | DELETE | N/A | N/A | Deletes the task contentId content and returns HTTP code 204. |
| description | PUT | Marshaled String value | N/A | Updates the task description and returns HTTP code 201 if successful. You must provide the new value for description in the request body. |

| URI | Method | Request Type | Response Type | Description |
|------------|--------|-------------------------|---------------|-------------------------------------------------------------------------------------------------------------------------------------------------------------|
| expiration | PUT | Marshaled Date value | N/A | Updates the task expiration date and returns HTTP 201 if successful. You must provide the new value for the expiration date in the request body. |
| name | PUT | Marshaled String value | N/A | Updates the task name and returns HTTP code 201 if successful. You must provide the new value for name in the request body. |
| priority | PUT | Marshaled int value | N/A | Updates the task priority and returns HTTP code 201 if successful. You must provide the new value for priority in the request body. |
| skipable | PUT | Marshaled boolean value | N/A | Updates the task property <i>skipable</i> and returns HTTP 201 if successful. You must provide the new value for the skipable property in the request body. |

Example 10.7. User Task Instance Data

- Get a user task instance for container **test**:

```
$ curl -X GET -u 'kieserver:kieserver1!'
'http://localhost:8080/kie-
server/services/rest/server/containers/test/tasks/1'
```

Example response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<task-instance>
```

```

<task-id>1</task-id>
<task-priority>0</task-priority>
<task-name>Approval by PM</task-name>
<task-subject></task-subject>
<task-description></task-description>
<task-form>ApprovalbyPM</task-form>
<task-status>Ready</task-status>
<task-actual-owner></task-actual-owner>
<task-created-by></task-created-by>
<task-created-on>2016-02-15T13:31:10.624-02:00</task-created-
on>
  <task-activation-time>2016-02-15T13:31:10.624-02:00</task-
activation-time>
  <task-skipable>false</task-skipable>
  <task-workitem-id>1</task-workitem-id>
  <task-process-instance-id>1</task-process-instance-id>
  <task-parent-id>-1</task-parent-id>
  <task-process-id>com.sample.rewards-basic</task-process-id>
  <task-container-id>rewards</task-container-id>
</task-instance>

```

- Set priority to 3 for task 1:

```

$ curl -X PUT -u 'kieserver:kieserver1!' -H 'Content-type:
application/json' -H 'X-KIE-ContentType: JSON' --data '3'
'http://localhost:8080/kie-
server/services/rest/server/containers/test/tasks/1/priority'

```

- Add a comment to a task 2:

```

$ curl -X POST -u 'kieserver:kieserver1!' -H 'Content-type:
application/json' -H 'X-KIE-ContentType: JSON' --data '{ "comment"
: "One last comment", "comment-added-by": "kieserver"}'
'http://localhost:8080/kie-
server/services/rest/server/containers/test/tasks/2/comments'

```

- Get all task comments:

```

$ curl -u 'kieserver:kieserver1!' -H 'Accept: application/json'
'http://localhost:8080/kie-
server/services/rest/server/containers/test/tasks/2/comments'

```

Example response:

```

{
  "task-comment" : [ {
    "comment-id" : 1,
    "comment" : "Some task comment",
    "comment-added-by" : "kieserver"
  }, {
    "comment-id" : 3,
    "comment" : "One last comment",
    "comment-added-by" : "kieserver"
  } ]
}

```

10.7. THE REST API FOR REALTIME DECISION SERVER ADMINISTRATION

This section provides information about the Rest API for both managed and unmanaged Realtime Decision Server environments. You must set correct HTTP headers for the servers. Refer to [The REST API for Realtime Decision Server Execution](#) section for further information about HTTP headers.

10.7.1. Managed Realtime Decision Server Environment

When you have a managed Realtime Decision Server setup, you need to manage Realtime Decision Server and containers through a controller. Usually, it is done through Business Central, but you may also use Controller REST API.

- The controller base URL is provided by kie-wb war deployment, which is the same as `org.kie.server.controller` property (for example `http://localhost:8080/kie-wb/rest/controller`).
- All requests require basic HTTP Authentication for the role `kie-server`.

[GET] /admin/servers

Returns a list of Realtime Decision Server instances.

Example 10.8. Example Server Response

```
<kie-server-instance-list>
  <kie-server-instance>
    <server-id>local-server-123</server-id>
    <server-setup>
      <server-containers>
        <containers container-id="MyContainer" status="STARTED">
          <release-id>
            <artifact-id>project1</artifact-id>
            <group-id>com.sample</group-id>
            <version>1.0.0</version>
          </release-id>
        </containers>
      </server-containers>
    </server-setup>
    <server-managed-instances>
      <managedInstances>
        <location>http://localhost:8080/kie-
server/services/rest/server</location>
        <status>UP</status>
        <capabilities>
          <capabilities>KieServer</capabilities>
          <capabilities>BRM</capabilities>
          <capabilities>BPM</capabilities>
        </capabilities>
      </managedInstances>
    </server-managed-instances>
    <server-name>local-server-123</server-name>
    <server-status>UP</server-status>
```

```

    <server-version>6.3.0.Final</server-version>
  </kie-server-instance>
</kie-server-instance-list>

```

[GET] /admin/server/{id}

Returns a Realtime Decision Server instance.

Example 10.9. Server Response

```

<kie-server-instance>
  <server-id>local-server-123</server-id>
  <server-setup>
    <server-containers>
      <containers container-id="MyContainer" status="STARTED">
        <release-id>
          <artifact-id>project1</artifact-id>
          <group-id>com.sample</group-id>
          <version>1.0.0</version>
        </release-id>
      </containers>
    </server-containers>
  </server-setup>
  <server-managed-instances>
    <managedInstances>
      <location>http://localhost:8080/kie-
server/services/rest/server</location>
      <status>UP</status>
      <capabilities>
        <capabilities>KieServer</capabilities>
        <capabilities>BRM</capabilities>
        <capabilities>BPM</capabilities>
      </capabilities>
    </managedInstances>
  </server-managed-instances>
  <server-name>local-server-123</server-name>
  <server-status>UP</server-status>
  <server-version>6.3.0.Final</server-version>
</kie-server-instance>

```

[PUT] /admin/server/{id}

Creates a new Realtime Decision Server instance with the specified id.

Example 10.10. Example Request to Create a New Realtime Decision Server Instance

```

<kie-server-info>
  <capabilities>KieServer</capabilities>
  <capabilities>BRM</capabilities>
  <location>http://localhost:8080/kie-
server/services/rest/server</location>
  <name>local-server-456</name>

```

```

<id>local-server-456</id>
<version>6.3.0.Final</version>
</kie-server-info>

```

[DELETE] /admin/server/{id}

Deletes a Realtime Decision Server instance with the specified id.

[GET] /admin/server/{id}/containers/{containerId}

Returns the container information including its release id.

Example 10.11. Server Response

```

<kie-container container-id="MyContainer" status="STARTED">
  <release-id>
    <artifact-id>project1</artifact-id>
    <group-id>com.sample</group-id>
    <version>1.0.0</version>
  </release-id>
</kie-container>

```

[PUT] /admin/server/{id}/containers/{containerId}

Creates a new container with the specified containerId and the given release id.

Example 10.12. Server Request

```

<kie-container container-id="MyContainer">
  <release-id>
    <artifact-id>project1</artifact-id>
    <group-id>com.sample</group-id>
    <version>1.0.0</version>
  </release-id>
</kie-container>

```

[DELETE] /admin/server/{id}/containers/{containerId}

Disposes a container with the specified containerId.

[POST] /admin/server/{id}/containers/{containerId}/status/started

Starts the container. No request body required.

[POST] /admin/server/{id}/containers/{containerId}/status/stopped

Stops the Container. No request body required.

10.7.2. Unmanaged Realtime Decision Server Environment

The execution server supports the following commands through the REST API. Note the following before using these commands:

- The base URL for these will remain as the endpoint defined earlier (*http://SERVER:PORT/kie-server/services/rest/server/*).
- All requests require basic HTTP Authentication for the role kie-server.

[GET] /

Returns the execution server information.

Example 10.13. Server Response

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

[POST] /config

Using POST HTTP method, you can execute various commands on the execution server, for example: create-container, list-containers, dispose-container and call-container. Following is the list of supported commands:

- `CreateContainerCommand`
- `GetServerInfoCommand`
- `ListContainersCommand`
- `CallContainerCommand`
- `DisposeContainerCommand`
- `GetContainerInfoCommand`
- `GetScannerInfoCommand`
- `UpdateScannerCommand`
- `UpdateReleaseIdCommand`

For more information about the commands, see the `org.kie.server.api.commands` package.

Example 10.14. Request to Create a Container

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<script>
  <create-container>
    <container container-id="command-script-container">
      <release-id>
        <artifact-id>baz</artifact-id>
        <group-id>foo.bar</group-id>
```



```

    <version>2.1.0.GA</version>
  </release-id>
</container>
</create-container>
<call-container container-id="command-script-container">
  <payload><?xml version="1.0" encoding="UTF-8" standalone="yes"?>
    <batch-execution lookup="defaultKieSession">
      <insert out-identifier="message" return-object="true" entry-
point="DEFAULT" disconnected="false">
        <object xsi:type="message"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
          <text>HelloWorld</text>
        </object>
      </insert>
      <fire-all-rules max="-1"/>
    </batch-execution>
  </payload>
</call-container>
<dispose-container container-id="command-script-container"/>
</script>

```

[GET] /containers

Returns a list of containers on the server.

Example 10.15. Server Response

```

<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 the specified container.

Example 10.16. Server Response

```

<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 execution server.

Example 10.17. Request to create a container

```

<kie-container container-id="MyRESTContainer">
  <release-id>
    <artifact-id>Project1</artifact-id>
    <group-id>com.redhat</group-id>
    <version>1.0</version>
  </release-id>
</kie-container>

```

Example 10.18. Example Server Response When Creating a Container

```

<response type="SUCCESS" msg="Container MyRESTContainer successfully
deployed with module com.redhat:Project1:1.0">
  <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>

```

[DELETE] /containers/{id}

Disposes a container specified by the id.

Example 10.19. Server Response disposing a container

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

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

Returns the full release id for the specified container.

Example 10.20. Server 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>
```

The server will respond with a success or error message, such as:

Example 10.21. Server Response

```
<response type="SUCCESS" msg="Release id successfully updated.">
  <release-id>
    <artifact-id>Project1</artifact-id>
    <group-id>com.redhat</group-id>
    <version>1.0</version>
  </release-id>
</response>
```

[GET] /containers/{id}/scanner

Returns information about the scanner for container's automatic updates.

Example 10.22. Server 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.

Example 10.23. Example Server Request to Start the Scanner

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

Example 10.24. Server 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.

10.8. REALTIME DECISION SERVER JAVA CLIENT API OVERVIEW

In [Chapter 4](#) from the Red Hat BRMS Getting Started Guide, you used the Realtime Decision Server Java Client API (that is a Maven structure) to send requests using REST API. This section further explains:

- [Maven Dependencies](#)
- [Client Configuration](#)
- [Server Response](#)
- [Inserting and Executing Commands](#)
- [Server Capabilities](#)
- [Containers](#)
- [Handling Containers](#)
- [Available Realtime Decision Server Clients](#)
- [List of Available Business Processes](#)

10.8.1. Maven Dependencies

To import Maven dependencies, you can use the Bill of Materials (BOM). Alternatively, you can follow the Hello World example.

Example 10.25. Maven Dependency using BOM

```
<dependencyManagement>
  <dependencies>
    <dependency>
      <groupId>org.jboss.bom.brms</groupId>
      <artifactId>jboss-brms-bpmsuite-bom</artifactId>
      <version>${version.org.jboss.bom.brms}</version>
      <type>pom</type>
      <scope>import</scope>
    </dependency>
  </dependencies>
</dependencyManagement>
<dependencies>
  <dependency>
    <groupId>org.kie.server</groupId>
```

```

        <artifactId>kie-server-client</artifactId>
      </dependency>
    </dependencies>

```

Use **6.2.1.GA-redhat-2** in `version.org.jboss.bom.brms` for Red Hat JBoss BRMS 6.2.0. Follow [Maven BOM page](#) to see all the available versions.

Example 10.26. Hello World Maven Dependency

```

<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>6.3.0.Final-redhat-7</version>
</dependency>

```

10.8.2. Client Configuration

You need to declare a configuration object and set server communication aspects, such as the protocol (REST or JMS), credentials and the payload format (XStream, JAXB or JSON). For additional example, follow the Hello World project.

Example 10.27. Client Configuration

```

import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;

public class DecisionServerTest {

    private static final String URL = "http://localhost:8080/kie-
server/services/rest/server";
    private static final String USER = "kieserver";
    private static final String PASSWORD = "kieserver1!";

    private static final MarshallingFormat FORMAT =
MarshallingFormat.JSON;

    private KieServicesConfiguration conf;
    private KieServicesClient kieServicesClient;

    @Before
    public void initialize() {
        conf = KieServicesFactory.newRestConfiguration(URL, USER,
PASSWORD);
        conf.setMarshallingFormat(FORMAT);
        kieServicesClient =
KieServicesFactory.newKieServicesClient(conf);
    }
}

```

Example 10.28. JMS Client Configuration

```

import java.util.Properties;

import javax.jms.ConnectionFactory;
import javax.jms.Queue;
import javax.naming.Context;
import javax.naming.InitialContext;

import org.junit.Test;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;

public class DecisionServerTest {

    private static final String REMOTING_URL = new
String("remote://localhost:4447");

    private static final String USER = "kieserver";
    private static final String PASSWORD = "kieserver1!";

    private static final String INITIAL_CONTEXT_FACTORY = new
String("org.jboss.naming.remote.client.InitialContextFactory");
    private static final String CONNECTION_FACTORY = new
String("jms/RemoteConnectionFactory");
    private static final String REQUEST_QUEUE_JNDI = new
String("jms/queue/KIE.SERVER.REQUEST");
    private static final String RESPONSE_QUEUE_JNDI = new
String("jms/queue/KIE.SERVER.RESPONSE");

    private KieServicesConfiguration conf;
    private KieServicesClient kieServicesClient;

    @Test
    public void testJms() throws Exception {
        final Properties env = new Properties();
        env.put(Context.INITIAL_CONTEXT_FACTORY,
INITIAL_CONTEXT_FACTORY);
        env.put(Context.PROVIDER_URL, REMOTING_URL);
        env.put(Context.SECURITY_PRINCIPAL, USER);
        env.put(Context.SECURITY_CREDENTIALS, PASSWORD);
        InitialContext context = new InitialContext(env);

        Queue requestQueue = (Queue)
context.lookup(REQUEST_QUEUE_JNDI);
        Queue responseQueue = (Queue)
context.lookup(RESPONSE_QUEUE_JNDI);
        ConnectionFactory connectionFactory = (ConnectionFactory)
context.lookup(CONNECTION_FACTORY);

        conf =
KieServicesFactory.newJMSConfiguration(connectionFactory, requestQueue,
responseQueue, USER, PASSWORD);

        kieServicesClient =

```

```
KieServicesFactory.newKieServicesClient(conf);

    }
}
```

Note that you must assign the the *guest* role to the user kieserver. Additionally, you must declare JMS dependency:

```
<dependency>
  <groupId>org.jboss.as</groupId>
  <artifactId>jboss-as-jms-client-bom</artifactId>
  <version>7.5.6.Final-redhat-2</version>
  <type>pom</type>
</dependency>
```

10.8.3. Server Response

Service responses are represented by the `org.kie.server.api.model.ServiceResponse<T>` object, where T represents the payload type. It has following attributes:

- **String msg:** returns the response message.
- **ResponseType type:** returns SUCCESS or FAILURE.
- **T result:** returns the requested object.

Example 10.29. Hello World Server Response

```
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.RuleServicesClient;

RuleServicesClient ruleClient =
client.getServicesClient(RuleServicesClient.class);
ServiceResponse<String> response = ruleClient.executeCommands(container,
batchCommand);
System.out.println(response.getResult());
```

10.8.4. Inserting and Executing Commands

You must use the `org.kie.api.command.KieCommands` class to insert commands. Use `org.kie.api.KieServices.get().getCommands()` to instantiate the `KieCommands` class. If you want to add multiple commands, you must use the `BatchExecutionCommand` wrapper. For additional example, follow the Hello World project.

Example 10.30. Inserting and Executing Commands

```
public void executeCommands() {
    String containerId = "hello";
    System.out.println("== Sending commands to the server ==");
    RuleServicesClient rulesClient =
kieServicesClient.getServicesClient(RuleServicesClient.class);
```

```

KieCommands commandsFactory = KieServices.Factory.get().getCommands();
Command<?> insert = commandsFactory.newInsert("Some String OBJ");
Command<?> fireAllRules = commandsFactory.newFireAllRules();
Command<?> batchCommand =
commandsFactory.newBatchExecution(Arrays.asList(insert, fireAllRules));
ServiceResponse<String> executeResponse =
rulesClient.executeCommands(containerId, batchCommand);
if(executeResponse.getType() == ResponseType.SUCCESS) {
    System.out.println("Commands executed with success! Response: ");
    System.out.println(executeResponse.getResult());
}
else {
    System.out.println("Error executing rules. Message: ");
    System.out.println(executeResponse.getMsg());
}
}

```

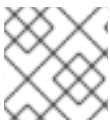
You must add the **org.drools:drools-compiler** Maven dependency:

```

<!-- Valid for version 6.2.0 -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-compiler</artifactId>
  <version>6.3.0.Final</version>
</dependency>

```

10.8.5. Server Capabilities



NOTE

Following sections expand topics not covered by the Hello World project.

From the version 6.2, Realtime Decision Server supports the business process execution. To know your server capabilities, use the **org.kie.server.api.model.KieServerInfo** object.

Example 10.31. Server Capabilities

```

public void listCapabilities() {
    KieServerInfo serverInfo =
kieServicesClient.getServerInfo().getResult();
    System.out.print("Server capabilities:");
    for(String capability: serverInfo.getCapabilities()) {
        System.out.print(" " + capability);
    }
    System.out.println();
}

```

10.8.6. Containers

Containers are represented by the `org.kie.server.api.model.KieContainerResource` object. List of resources is represented by the `org.kie.server.api.model.KieContainerResourceList` object.

Example 10.32. Print a List of Containers

```
public void listContainers() {
    KieContainerResourceList containersList =
kieServicesClient.listContainers().getResult();
    List<KieContainerResource> kieContainers =
containersList.getContainers();
    System.out.println("Available containers: ");
    for (KieContainerResource container : kieContainers) {
        System.out.println("\t" + container.getContainerId() + " (" +
container.getReleaseId() + ")");
    }
}
```

10.8.7. Handling Containers

You can use the Realtime Decision Server Java client to create and dispose containers. If you dispose a container, `ServiceResponse` will be returned with `void` payload. If you create a container, `KieContainerResource` object will be returned.

Example 10.33. Dispose and Create a Container

```
public void disposeAndCreateContainer() {
    System.out.println("== Disposing and creating containers ==");
    List<KieContainerResource> kieContainers =
kieServicesClient.listContainers().getResult().getContainers();
    if (kieContainers.size() == 0) {
        System.out.println("No containers available...");
        return;
    }
    KieContainerResource container = kieContainers.get(0);
    String containerId = container.getContainerId();
    ServiceResponse<Void> responseDispose =
kieServicesClient.disposeContainer(containerId);
    if (responseDispose.getType() == ResponseType.FAILURE) {
        System.out.println("Error disposing " + containerId + ". Message: ");
        System.out.println(responseDispose.getMsg());
        return;
    }
    System.out.println("Success Disposing container " + containerId);
    System.out.println("Trying to recreate the container...");
    ServiceResponse<KieContainerResource> createResponse =
kieServicesClient.createContainer(containerId, container);
    if (createResponse.getType() == ResponseType.FAILURE) {
        System.out.println("Error creating " + containerId + ". Message: ");
        System.out.println(responseDispose.getMsg());
        return;
    }
}
```

```

    }
    System.out.println("Container recreated with success!");
}

```

10.8.8. Available Realtime Decision Server Clients

KieServicesClient serves also as an entry point for other clients with the ability to perform various operations, such as JBoss BRMS commands and manage processes. Following services are available in the `org.kie.server.client` package:

- **JobServicesClient** is used to schedule, cancel, requeue, and get job requests.
- **ProcessServicesClient** is used to start, signal, and abort processes or work items.
- **QueryServicesClient** is used to query processes, process nodes, and process variables.
- **RuleServicesClient** is used to send commands to the server to perform rule-related operations (for example insert objects into the working memory, fire rules, ...).
- **UserTaskServicesClient** is used to perform all user-task operations (start, claim, cancel a task) and query tasks by specified field (process instances id, user, ...)

The **getServicesClient** method provides access to any of these clients:

```

RuleServicesClient rulesClient =
kieServicesClient.getServicesClient(RuleServicesClient.class);

```

10.8.9. List of Available Business Processes

Use **QueryClient** to list available process definitions. **QueryClient** methods use pagination, therefore in addition to the query you make, you must provide the current page and the number of results per page. In the provided example, the query starts on page 0 and lists the first 1000 results.

Example 10.34. List Processes

```

public void listProcesses() {
    System.out.println("== Listing Business Processes ==");
    QueryServicesClient queryClient =
kieServicesClient.getServicesClient(QueryServicesClient.class);
    List<ProcessDefinition> findProcessesByContainerId =
queryClient.findProcessesByContainerId("rewards", 0, 1000);
    for (ProcessDefinition def : findProcessesByContainerId) {
        System.out.println(def.getName() + " - " + def.getId() + " v" +
def.getVersion());
    }
}

```

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 or 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 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","jobId":"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 RepositoryRequest 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

Note that revision numbers relate to the edition of this manual, not to version numbers of Red Hat JBoss BRMS.

| | | |
|--------------------------------------------------------------------------------------------------|------------------------|--------------------|
| Revision 6.2.0-4 Updated with latest fixes. | Thu Apr 28 2016 | Tomas Radej |
| Revision 6.2.0-3 Build for release update 2 of JBoss BRMS. | Tue Mar 29 2016 | Tomas Radej |
| Revision 6.2.0-2 Added note about versions in Revision History, fixed changelog dates. | Mon Nov 30 2015 | Tomas Radej |
| Revision 6.2.0-1 Initial build for release 6.2.0 of JBoss BRMS. | Mon Nov 30 2015 | Tomas Radej |