



Red Hat JBoss BRMS 6.0

User Guide

The user guide for Red Hat JBoss BRMS 6

Red Hat JBoss BRMS 6.0 User Guide

The user guide for Red Hat JBoss BRMS 6

Kanchan Desai
kadesai@redhat.com

Doug Hoffman

Eva Kopalova

Red Hat Content Services

Legal Notice

Copyright © 2014 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 for business users working with Red Hat JBoss BRMS 6

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 JBOSS BRMS	4
1.3. ASSETS	5
1.4. INTEGRATED MAVEN DEPENDENCIES	6
CHAPTER 2. BUSINESS CENTRAL	8
2.1. LOGGING ON TO BUSINESS CENTRAL	8
2.2. THE HOME SCREEN	9
2.3. PROJECT AUTHORIZING	9
2.4. PROJECT EDITOR	12
2.5. ADMINISTRATION MENU	18
2.6. RENAME, COPY, DELETE ASSETS	18
2.7. DEPLOYMENT MENU: THE ARTIFACT REPOSITORY	19
CHAPTER 3. SETTING UP A NEW PROJECT	21
3.1. CREATING AN ORGANIZATIONAL UNIT	21
3.2. CREATING A REPOSITORY	22
3.3. CLONING A REPOSITORY	23
3.4. CREATING A PROJECT	25
3.5. CREATING A NEW PACKAGE	26
3.6. ADDING DEPENDENCIES	27
3.7. DEFINING KIE BASES AND SESSIONS	27
3.8. CREATING A RESOURCE	29
CHAPTER 4. DATA MODELS	30
4.1. DATA MODELER	30
4.2. CREATING A DATA OBJECT	30
CHAPTER 5. WRITING RULES	33
5.1. CREATING A RULE	33
5.2. THE ASSET EDITOR	33
5.3. DECISION TABLES	38
5.4. WEB BASED GUIDED DECISION TABLES	40
5.5. RULE TEMPLATES	56
5.6. THE DOMAIN SPECIFIC LANGUAGE EDITOR	65
5.7. DATA ENUMERATIONS	66
5.8. SCORECARDS	67
CHAPTER 6. BUILDING AND DEPLOYING ASSETS	69
CHAPTER 7. MANAGING ASSETS	70
7.1. CATEGORIES	70
7.2. VERSIONS AND STORAGE	70
7.3. DISCUSSION	70
CHAPTER 8. TESTING	71
8.1. TEST SCENARIOS	71
8.2. CREATING A TEST SCENARIO	71
CHAPTER 9. REST API	75
9.1. KNOWLEDGE STORE REST API	75
9.2. REST SUMMARY	79

APPENDIX A. REVISION HISTORY 81

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 as a technical preview with this release.

[Report a bug](#)

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

BRMS comprises a high-performance rule engine from the Drools project, a rule repository and easy to use rule authoring tools from the Drools Guvnor project, and Complex Event Processing rule engine extensions from the Drools Fusion project. It also includes Business Resource Planner, a solver for complex planning problems, as a technology preview.

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 to the visitors of the solution relevant, additional products during the quotation process.

JBoss BRMS was integrated with the insurance providers's infrastructure so that when a request for insurance was processed, BRMS was consulted and appropriate additional products were presented with the insurance quotation:

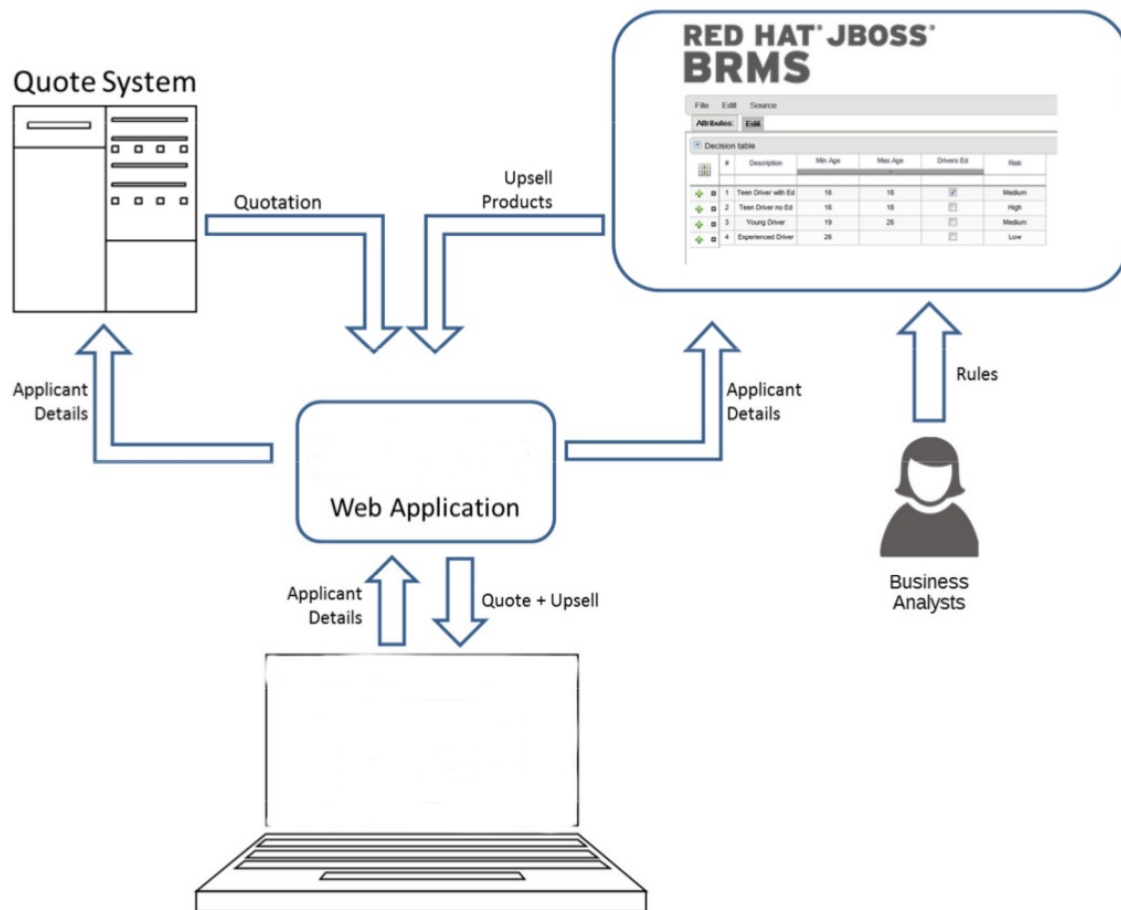


Figure 1.1. BRMS Use Case: Insurance Industry Decision Making

BRMS provided the decision management functionality, i.e. the automatic determination of the products to present to the applicant based on the rules defined by business analysts. The rules were implemented as decision tables, so they could be easily understood and modified without requiring additional support from IT.

[Report a bug](#)

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.

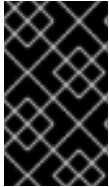
Business Processes

Business Processes are flow charts that describe the steps necessary to achieve business goals (see

the *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 language, 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 Enterprise 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.

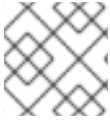
[Report a bug](#)

1.4. INTEGRATED MAVEN DEPENDENCIES

Throughout the Red Hat JBoss BRMS and BPM Suite documentation, various code samples are presented with KIE API for the 6.0.x releases. These code samples will require maven dependencies in the various `pom.xml` file and should be included like the following example:

```
<dependency>
  <groupId>commons-logging</groupId>
  <artifactId>commons-logging</artifactId>
  <version>1.1.1-redhat-2</version>
  <scope>compile</scope>
</dependency>
```

All the Red Hat JBoss related product dependencies can be found at the following location: [Red Hat Maven Repository](#)

**NOTE**

This repository is a technology preview and is likely to change in future releases.

[Report a bug](#)

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.

[Report a bug](#)

2.1. 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`.

[Report a bug](#)

2.2. 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:

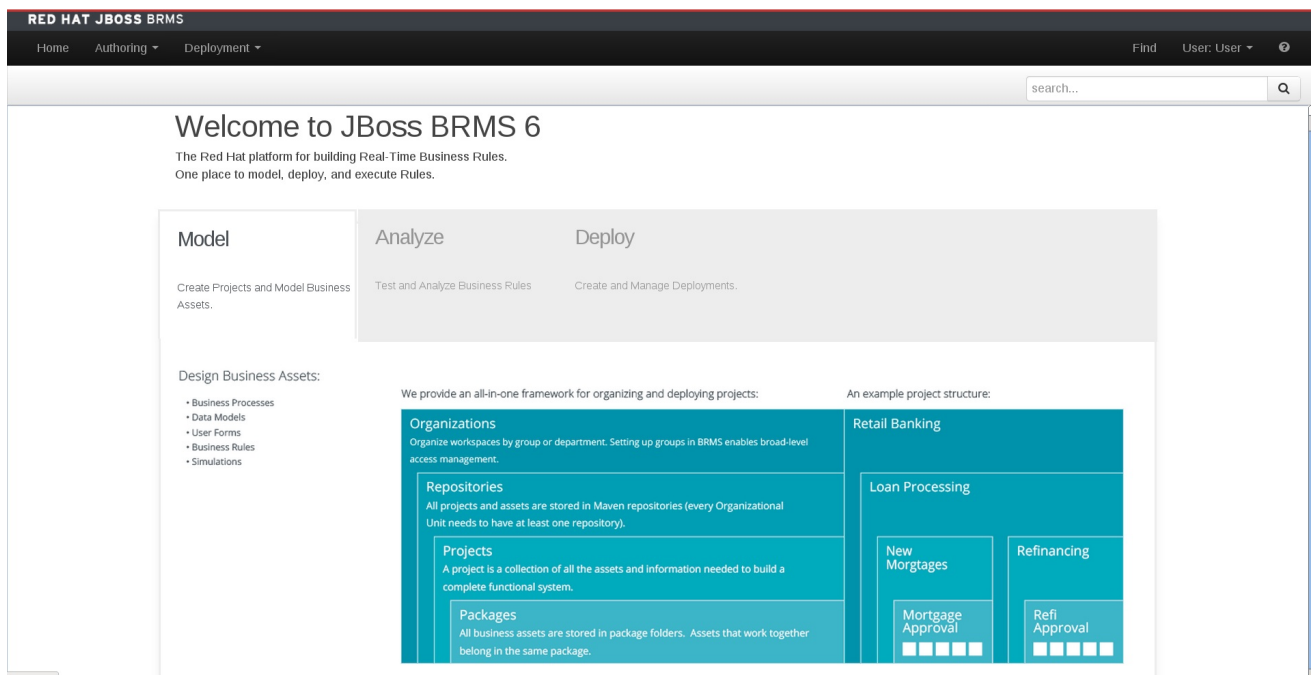


Figure 2.1. Business central home screen

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.

[Report a bug](#)

2.3. 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**.

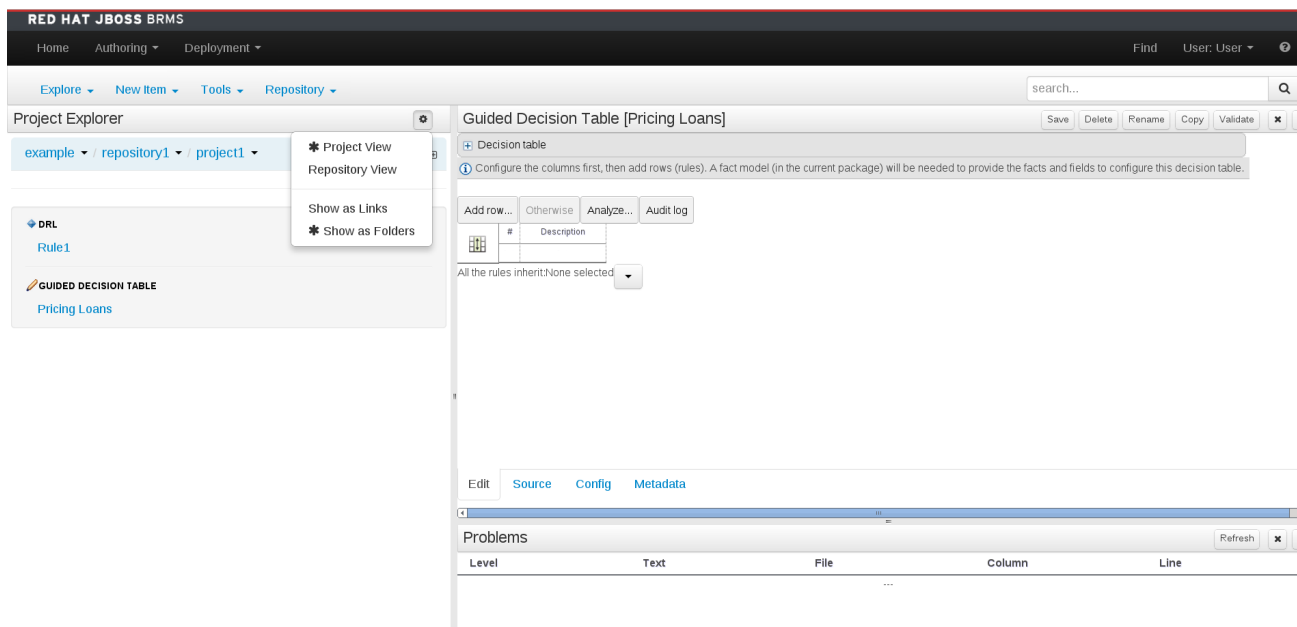



Figure 2.2. The Project Explorer screen

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.


[Report a bug](#)


2.3.1. 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 .
- 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:

- Move the mouse pointer on the title of the panel. The pointer changes to .
- 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.

[Report a bug](#)

2.3.2. Creating new assets

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

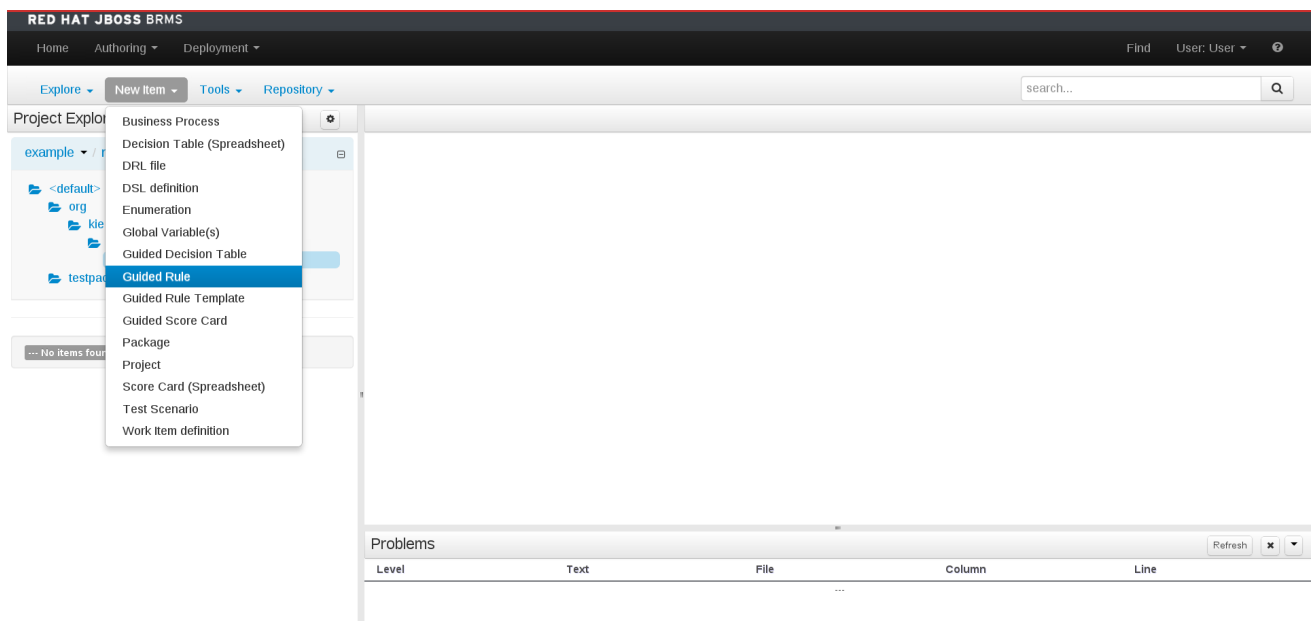
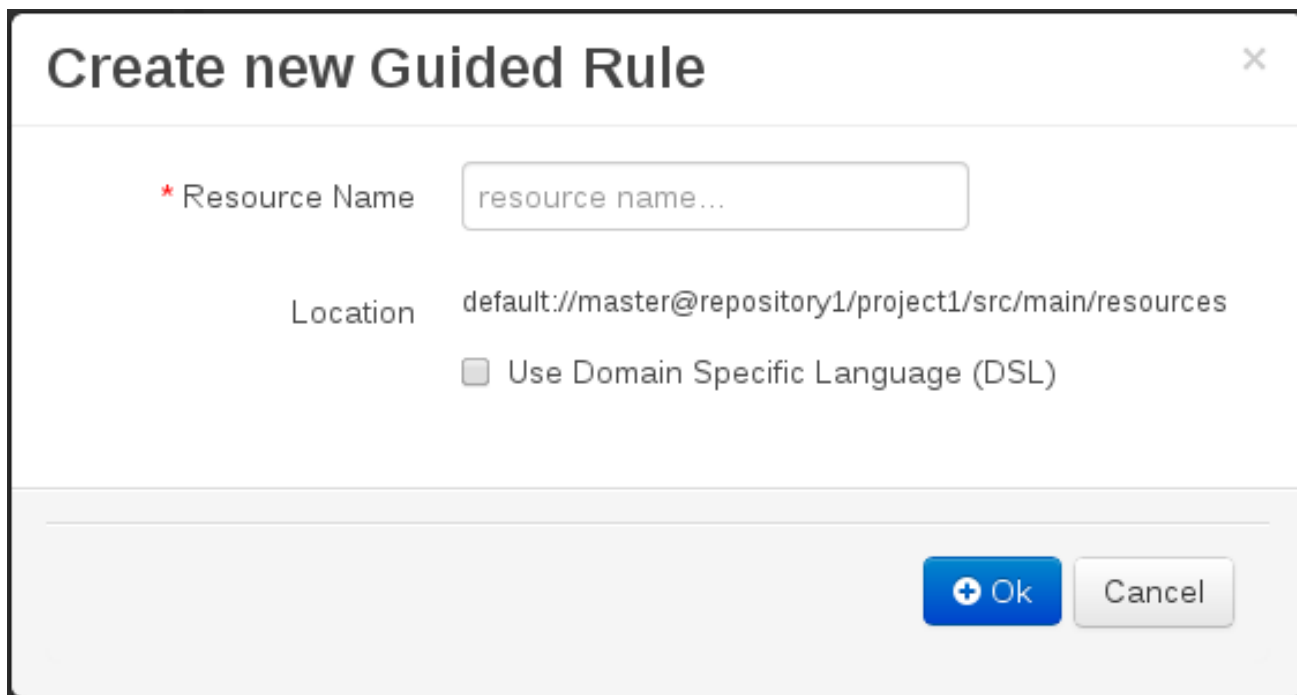


Figure 2.3. Creating new Asset screen

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.



The image shows a 'Create new Guided Rule' dialog box. It has a title bar with a close button (X) in the top right corner. The main area contains a form with the following fields: a required field for 'Resource Name' with a red asterisk and a text input field containing 'resource name...'; a 'Location' field with the text 'default://master@repository1/project1/src/main/resources'; and a checkbox labeled 'Use Domain Specific Language (DSL)' which is currently unchecked. At the bottom right, there are two buttons: a blue '+ Ok' button and a grey 'Cancel' button.

Figure 2.4. Create new pop-up dialog

[Report a bug](#)

2.4. PROJECT EDITOR

2.4.1. The Project Editor

The Project Editor helps a user to build and deploy projects. This view provides access to the various properties of a 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.

[Report a bug](#)

2.4.2. 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.

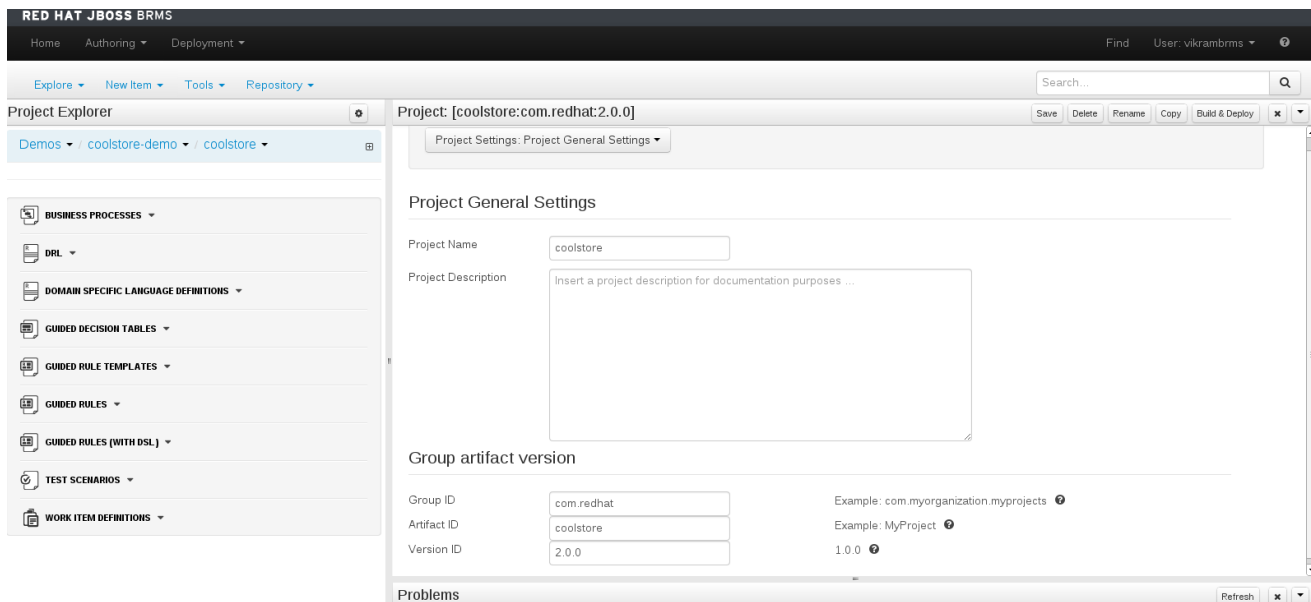


Figure 2.5. Project Editor - Project Settings

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.

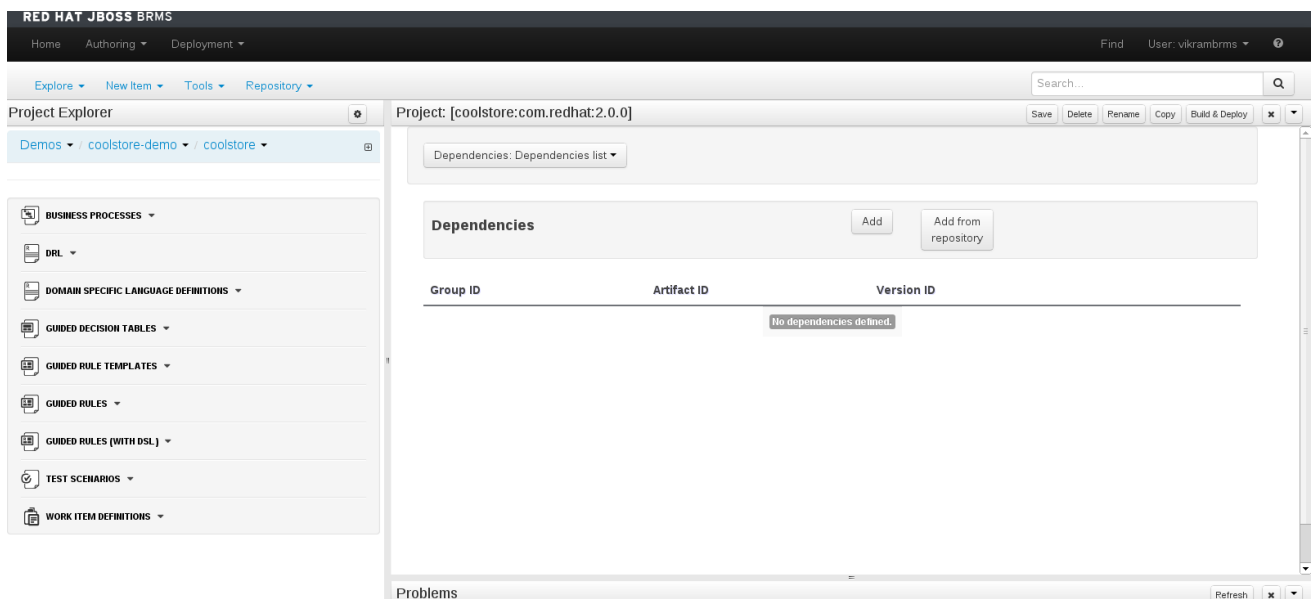



Figure 2.6. Project Editor - Project Dependencies

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 meta data** option provides Subject, Type, External Link, and Source miscellaneous meta data options for the asset.

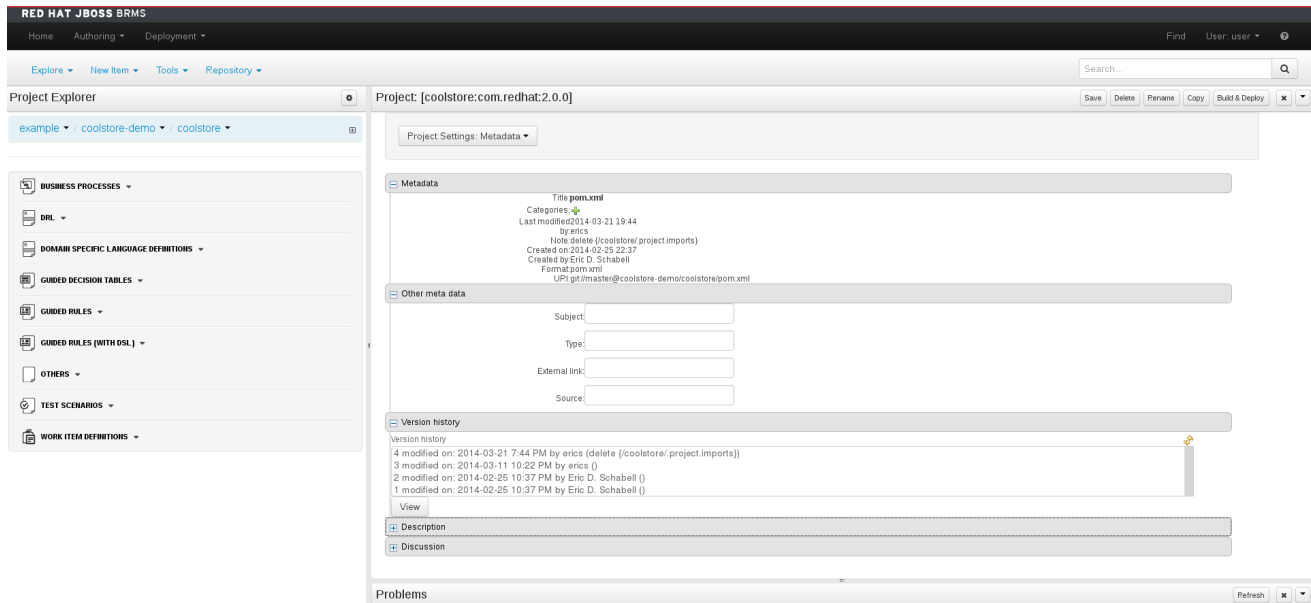


Figure 2.7. Project Editor - Metadata

The **Version history** option displays the modified history of the asset. You can select a modification

and press the **View** button to see advanced modification notes. You can press the **Restore** button to reload older versions of the file.

The **Description** option allows business friendly descriptions of the asset. This section is encouraged to record business logic rules before editing an asset.

The **Discussion** option is available to contribute reviews about the development of the asset. By

clicking the **Add a discussion comment** button, you can contribute to the discussion and then

click OK to display the comment. The **Erase all comments** button removes all the discussion posts.

[Report a bug](#)

2.4.3. 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.

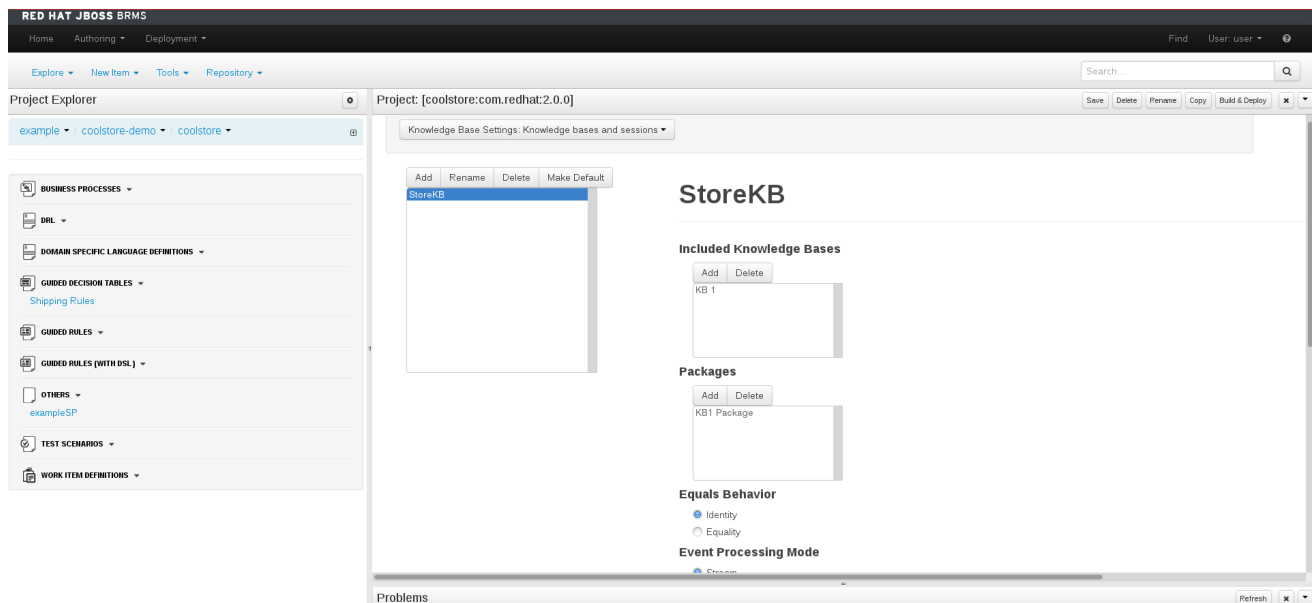


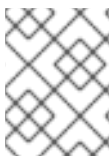
Figure 2.8. Project Editor - Knowledge Base Settings

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 **C**loud and **S**tream 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 2.4.2, “Project Settings”](#) for more information about Metadata.

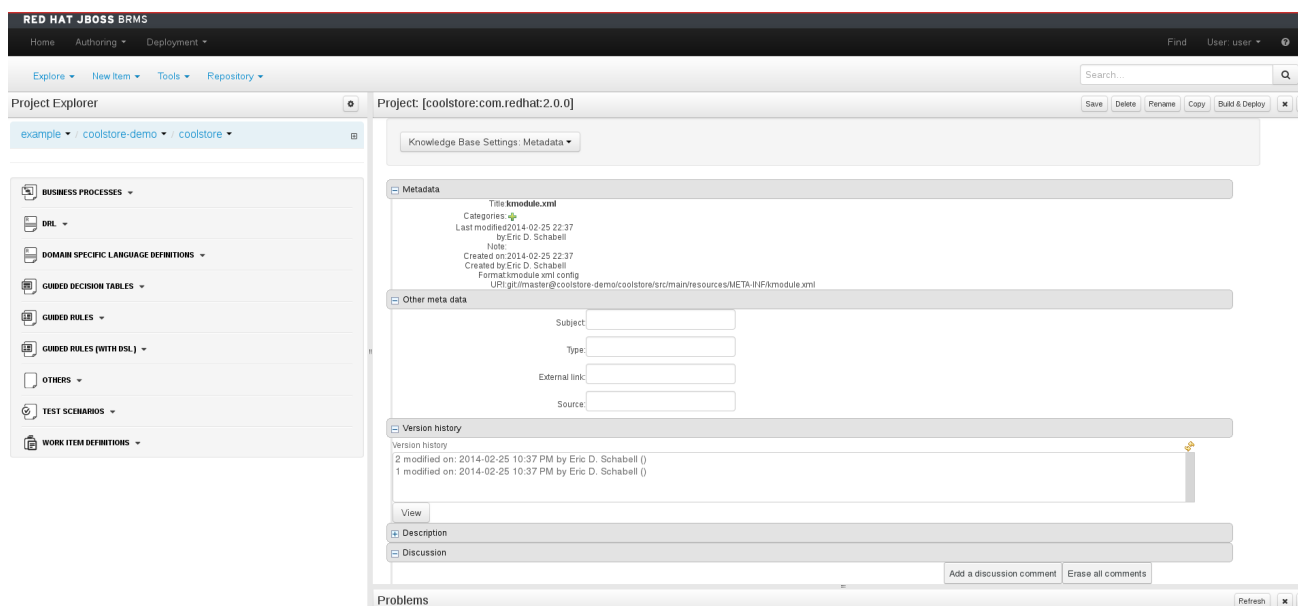


Figure 2.9. Knowledge Base Settings - Metadata

[Report a bug](#)

2.4.4. 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.

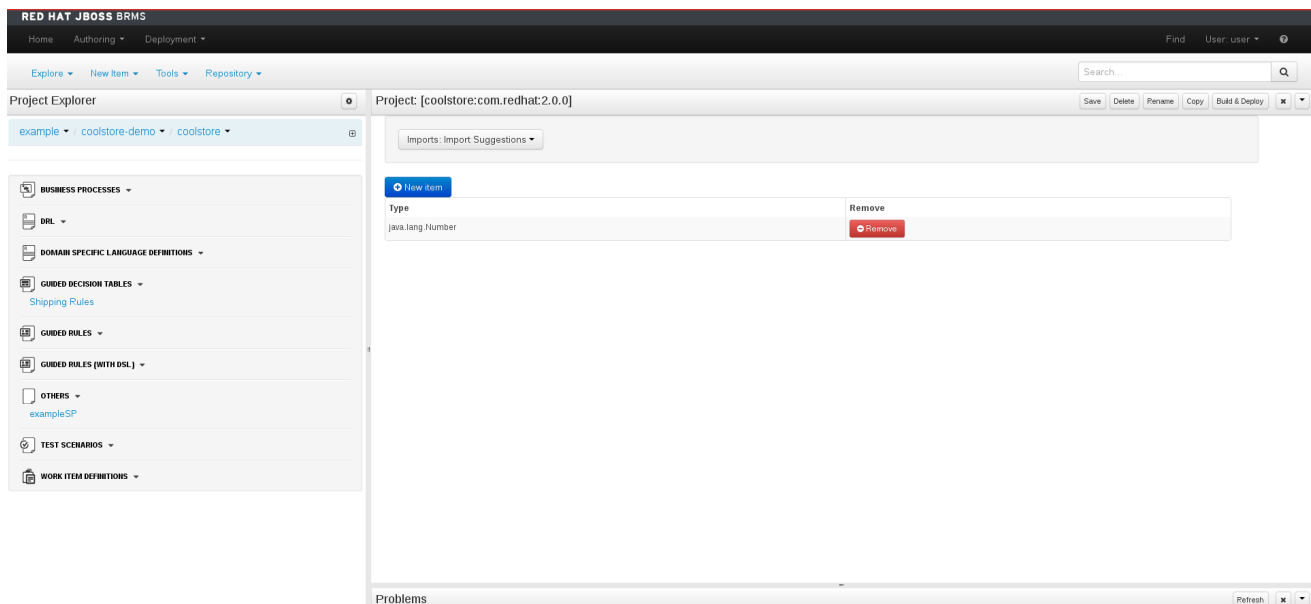



Figure 2.10. Project Editor - Imports

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

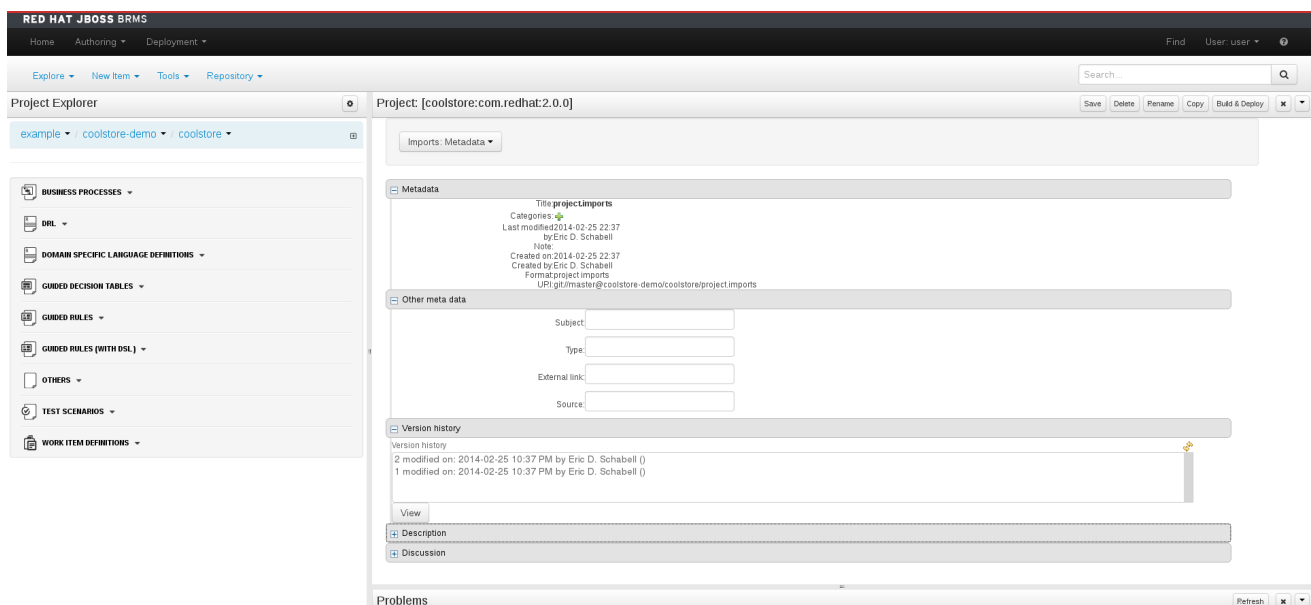


Figure 2.11. Imports - Metadata

[Report a bug](#)

2.5. 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](#).

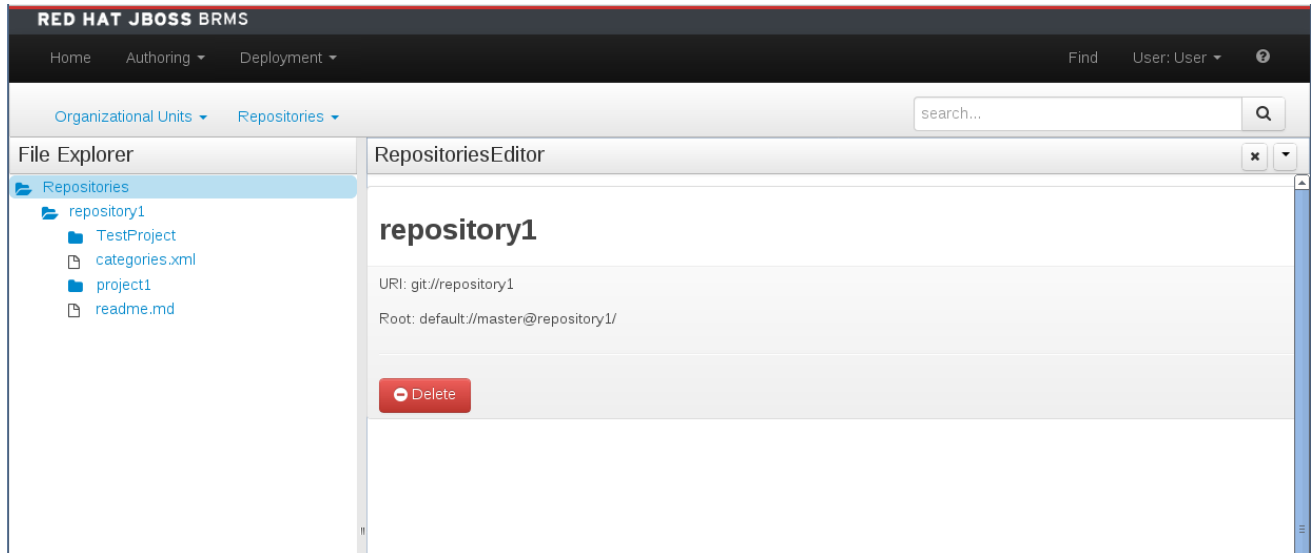


Figure 2.12. The Administration Screen

[Report a bug](#)

2.6. RENAME, COPY, DELETE ASSETS

2.6.1. 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.

[Report a bug](#)

2.6.2. 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**.

- 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).
- 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.

[Report a bug](#)

2.6.3. Copying a file or folder

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

- To copy a file or a folder, open **Project Explorer** by selecting **Authoring → Project Authoring**.
- 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).
- 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.

[Report a bug](#)

2.7. 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 **Deployment → Artifact Repository** menu on the toolbar.

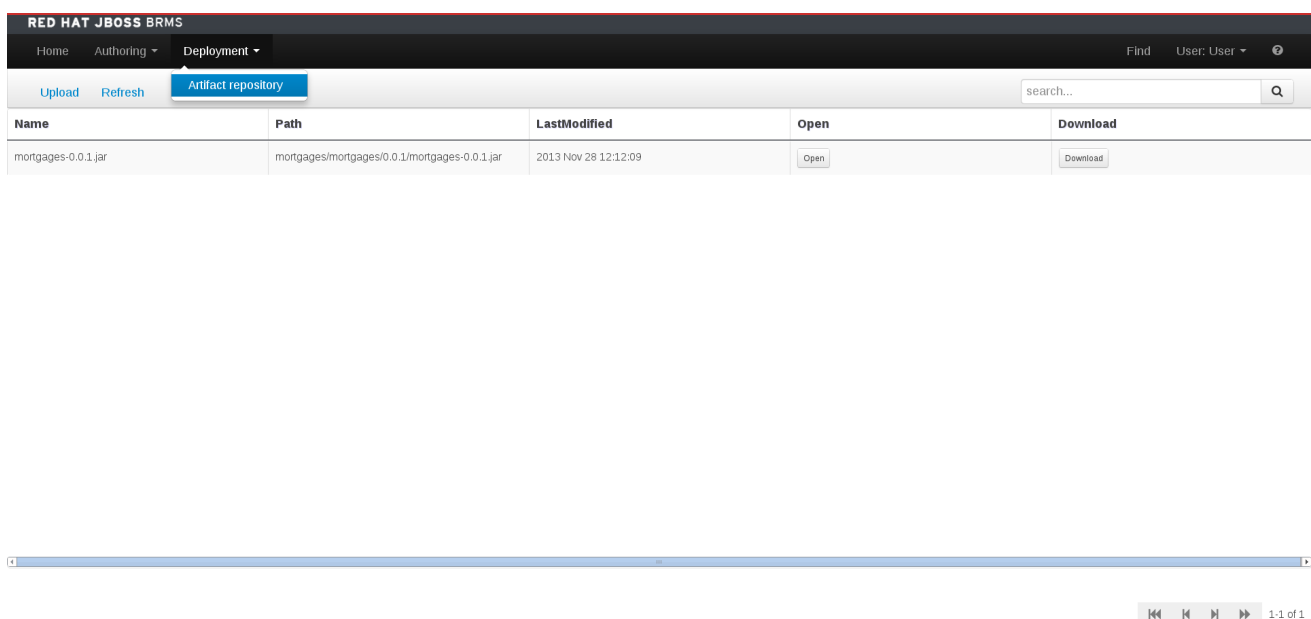


Figure 2.13. The Artifact Repository Screen

[Report a bug](#)

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.

[Report a bug](#)

3.1. CREATING AN ORGANIZATIONAL UNIT



IMPORTANT

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

Procedure 3.1. Creating an organizational unit

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

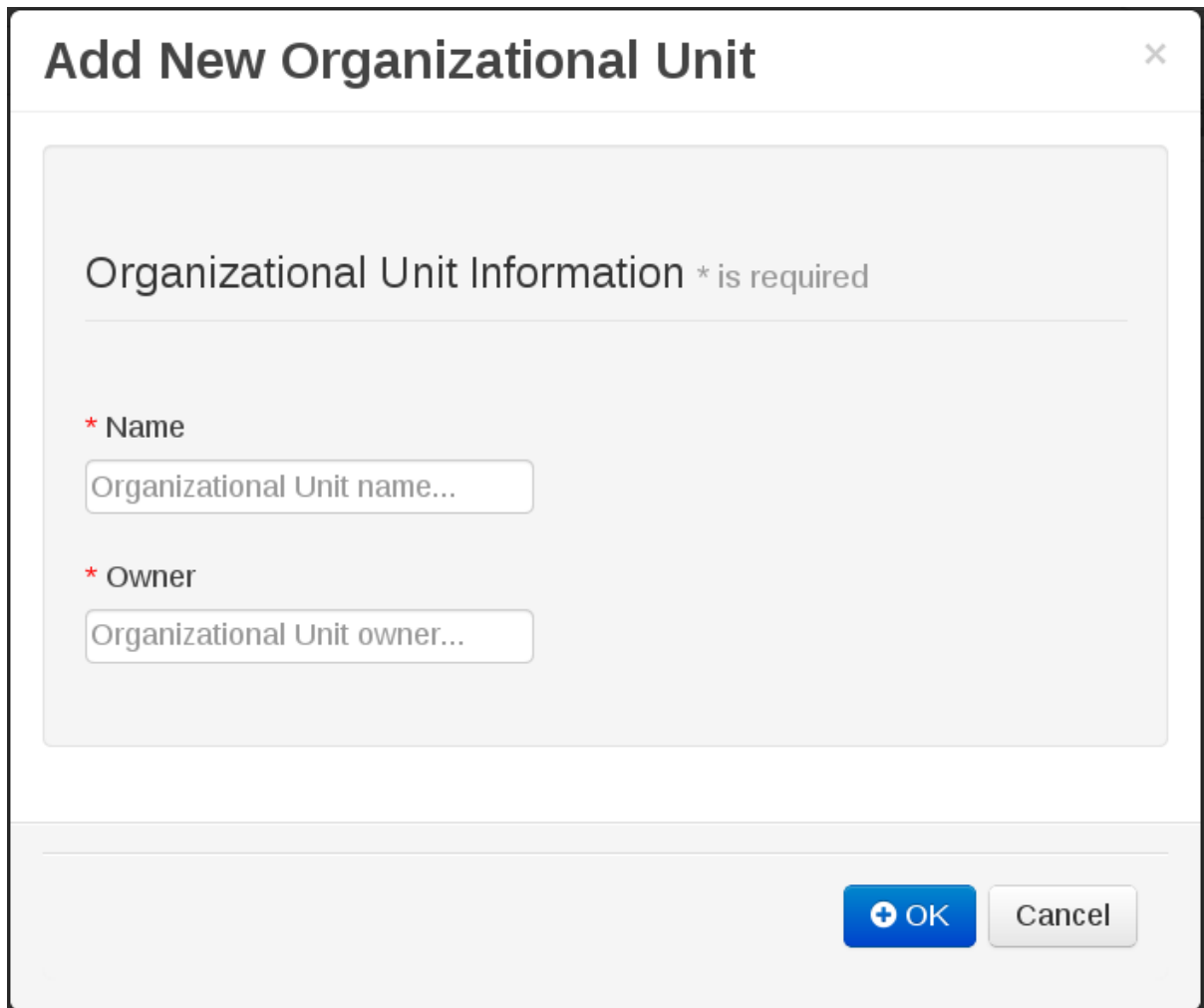
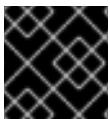


Figure 3.1. Add New Organizational Unit Pop-up

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

[Report a bug](#)

3.2. CREATING A REPOSITORY

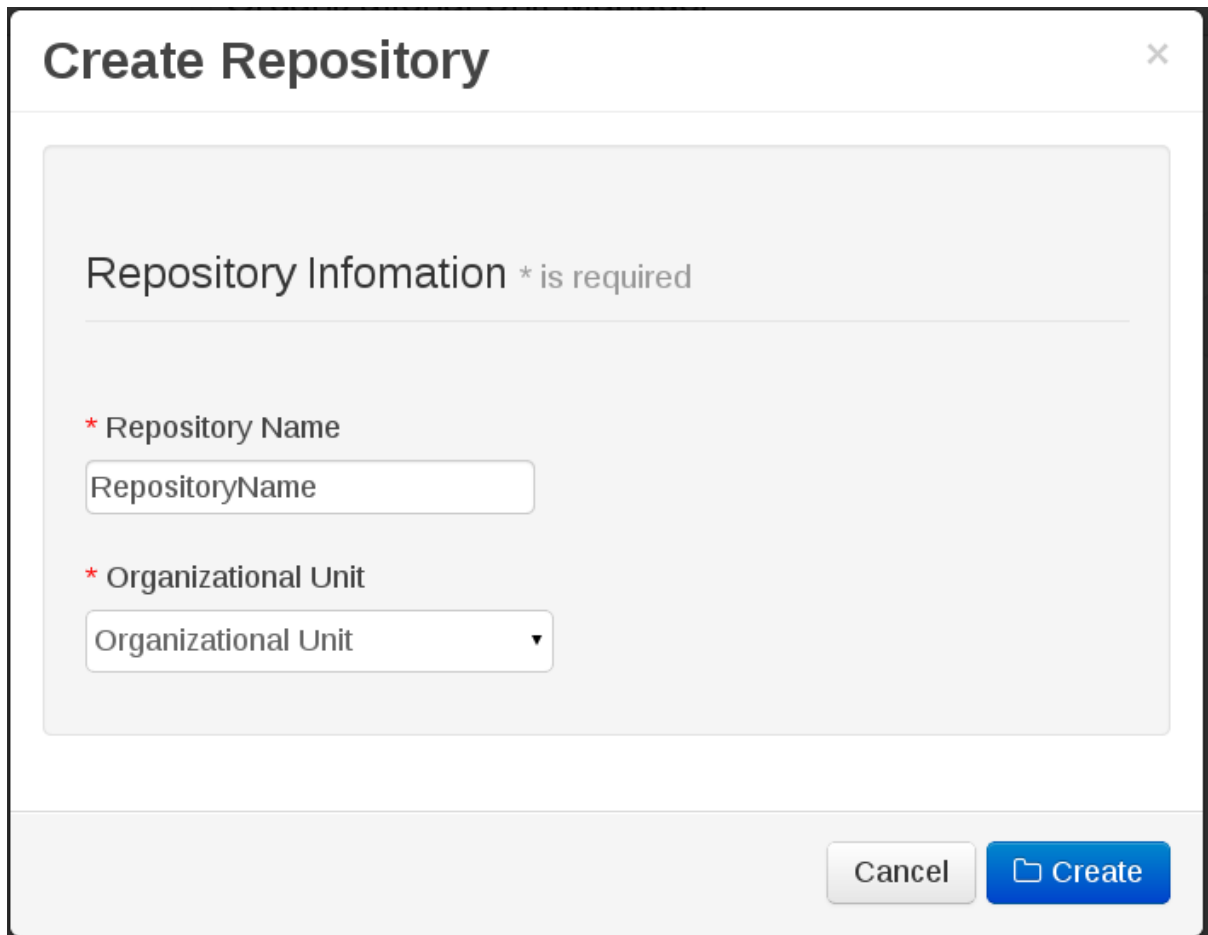


IMPORTANT

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.



The image shows a 'Create Repository' dialog box. At the top, the title 'Create Repository' is displayed with a close button (X) on the right. Below the title bar, the text 'Repository Information * is required' is shown. There are two mandatory fields: '* Repository Name' with a text input field containing 'RepositoryName', and '* Organizational Unit' with a dropdown menu showing 'Organizational Unit'. At the bottom right, there are two buttons: 'Cancel' and 'Create' (which is highlighted in blue).

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.

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

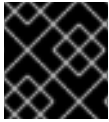
5. Click **Create**

6. A confirmation prompt with an **OK** button is displayed which notifies the user that the repository is created successfully. Click **OK**.

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

[Report a bug](#)

3.3. CLONING A REPOSITORY

**IMPORTANT**

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

Procedure 3.3. Cloning a repository

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

Figure 3.3. Clone Repository Pop-up

4. In the **Clone Repository** dialog window, enter the repository details:
 - a. Enter the **Repository Name** to be used as the repository identifier in the Asset repository and select the **Organizational Unit** it should be added to.
 - b. Enter the URL of the GIT repository:
 - For a Local Repository: `file:///path-to-repository/reponame`

- For a Remote or preexisting Repository: `git://hostname/reponame`



NOTE

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

- c. If applicable, enter the **User Name** and **Password** to be used for authentication when cloning the repository.
5. Click **Clone**.
6. A confirmation prompt with an **OK** button is displayed which notifies the user that the repository is created successfully. Click **OK**.

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

[Report a bug](#)

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 **Resource Name** text box, enter the project name.

Figure 3.4. New Project Screen

**NOTE**

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

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

Figure 3.5. New Project Wizard Pop-up

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 `Mortgages`
 - **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`
 - **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.

[Report a bug](#)

3.5. CREATING A NEW PACKAGE

Procedure 3.4. Creating a new package

1. Open the **Project Authoring** perspective: on the main menu, click **Authoring** → **Project Authoring**.
2. In the **Project Explorer** view, do the following:
 - If in the **Project** view of **Project Explorer**, select the organizational unit, repository and the project where you want to create the package.
 - If in the **Repository** view of **Project Explorer**, navigate to the project root, where you want to create the package.
3. In the perspective menu, go to **New Item** → **Package**.
4. In the **Create new Package** dialog window, define the package details:
 - a. In the **Resource Name** text box, enter the package name and click **OK**.
5. The new package is now created under the selected project.

[Report a bug](#)

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

[Report a bug](#)

3.7. DEFINING KIE BASES AND SESSIONS

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

Defining KIE bases and sessions in the Project Editor

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

1. Open your project properties with the Project Editor: in the Project Explorer, locate your project root. In the top menu, go to **Tools** → **Project Editor**.
2. In the **Project** Screen tab on the right, select "Knowledge Base Settings: Knowledge bases and sessions" option. This brings up a user interface for editing the `kmodule.xml` file.
3. Click on **Add** button to define and add your bases. Scroll down in this screen to add your knowledge sessions.
4. Click the **save** button in the top right once done.

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

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

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

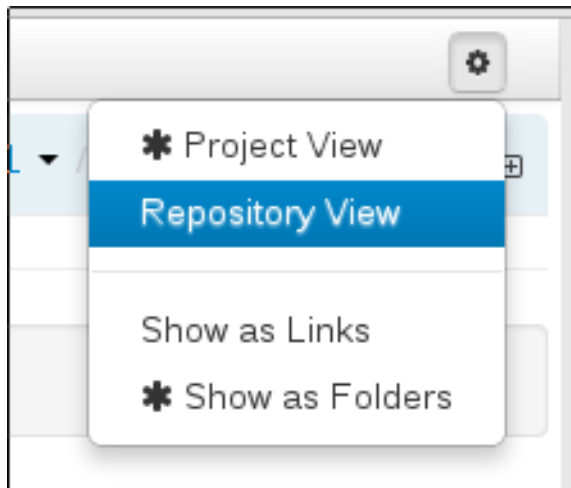


Figure 3.6. Changing to repository view

2. Click on `src` folder, and then drill down to locate the `META-INF` folder (`/src/main/resources/META-INF`). You will see the `kmodule.xml` file. Click on this file and its raw view will open up in a panel on the right hand side.
3. Define your bases and sessions in this file panel. An example of what you need to enter is shown below.

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

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



NOTE

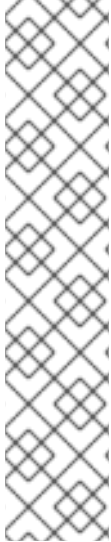
You can switch between the Project Editor view and the Repository view to look at the changes you make in each view, but to do so, you must close and reopen the view each time a change is made.

[Report a bug](#)

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.

[Report a bug](#)

CHAPTER 4. DATA MODELS

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

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



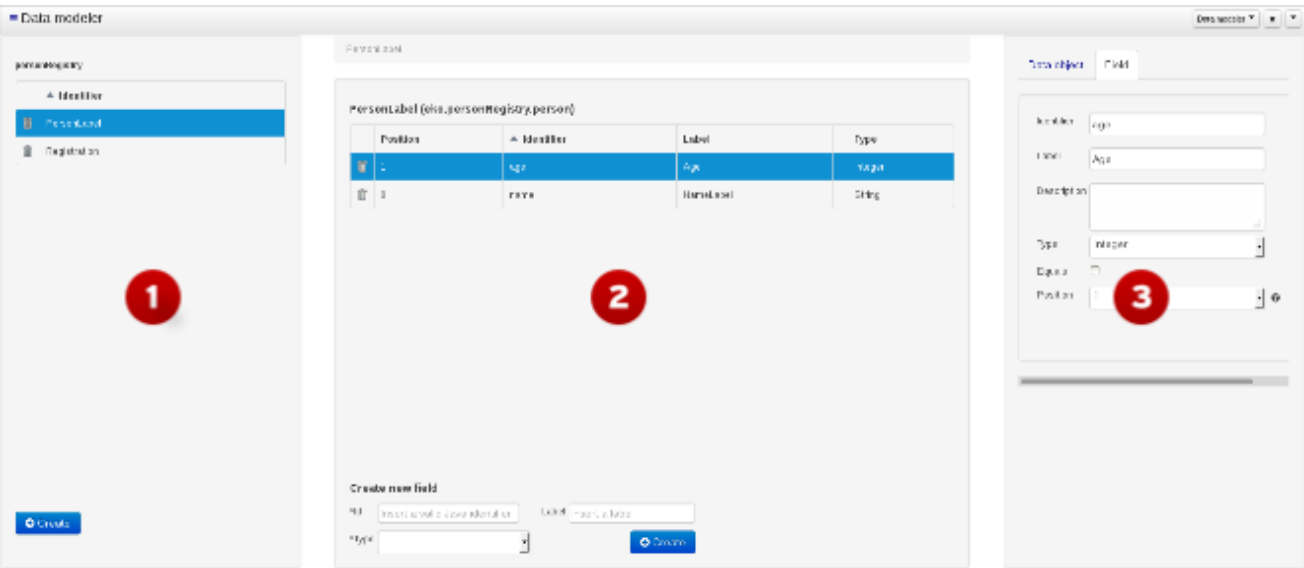
IMPORTANT

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

[Report a bug](#)

4.1. 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 **Tools** → **Data Modeler** on the perspective menu.



1	The Objects panel contains a list of data objects that constitute the data model in the given Project.
2	The Fields panel contains a list of fields of the data object selected in the Objects panel.
3	The Properties panel displays the properties of the data field selected in the Field panel.

Figure 4.1. Data Modeler environment

[Report a bug](#)

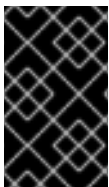
4.2. CREATING A DATA OBJECT

1. Open the Data Modeler: in the Project Authoring perspective, click **Tools** → **Data Modeler** on the perspective menu.
2. Create a data object:
 - a. In the **Objects** panel, click the **Create** button and provide the data object details:
 - **Identifier**: name of the data object unique within the project.

**WARNING**

This identifier name *must* be unique within the whole project, and not just the package.

- **Label**: name of the data object to be displayed in the **Objects** panel
 - **New package**: a new package the object should be created in
 - **Existing package**: an existing package the object should be created in
 - **Superclass**: a data object to be used as the superclass for the data object (the data objects extends this class, that is, it inherits all its fields)
3. Create fields of the data object:
 - a. Select the object in the **Objects** panel.
 - b. In the **Create new field** part of the **Fields** panel, define the field properties:
 - **Id**: field ID unique within the data object
 - **Label**: label to be used in the **Fields** panel
 - **Type**: data type of the field
 - c. Click **Create**.

**IMPORTANT**

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

**WARNING**

When the Data Modeler opens a project's model objects, it checks if the model has been externally modified. If it has been, then it will notify you that the object was externally modified and will still allow you to modify the model.

However, any such subsequent changes made in the model via the Data Modeler won't get propagated to the external source where it was original created/modified. You risk losing changes to your model as a full roundtrip is not implemented.

[Report a bug](#)

CHAPTER 5. WRITING RULES

5.1. CREATING A RULE

Procedure 5.1. 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.

[Report a bug](#)

5.2. THE ASSET EDITOR

5.2.1. 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.

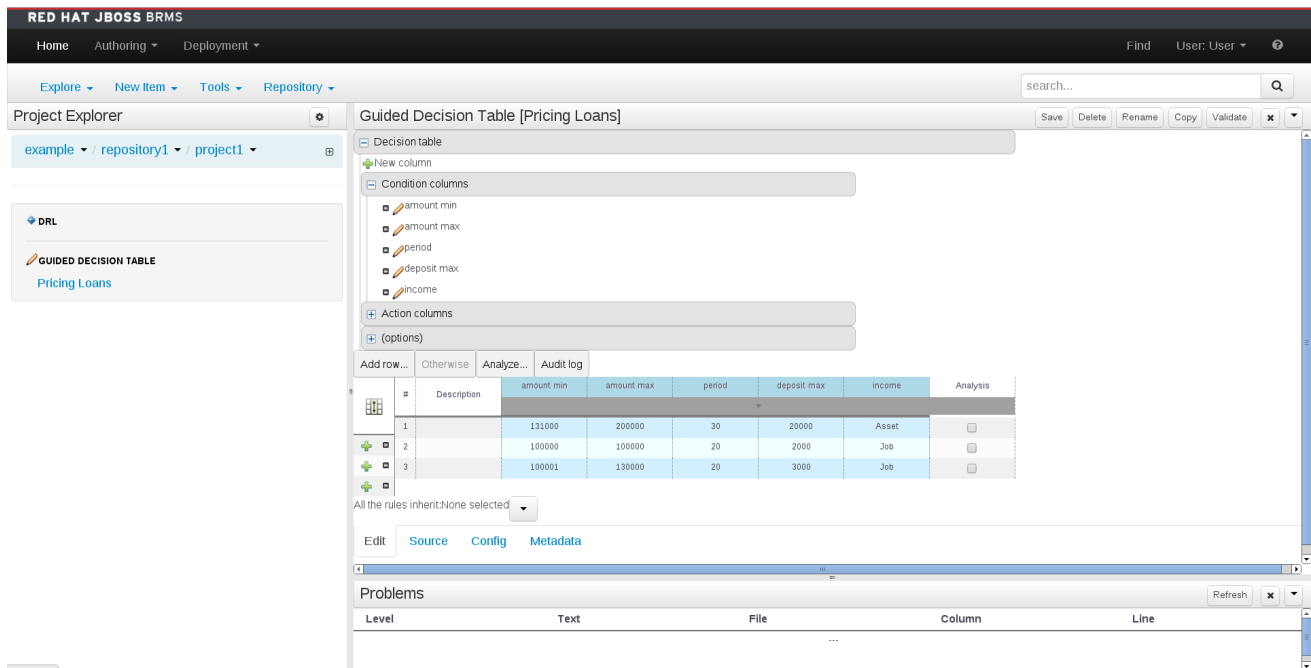


Figure 5.1. The asset editor - Edit tab

Source tab

Source tab shows the DRL source for a selected asset.

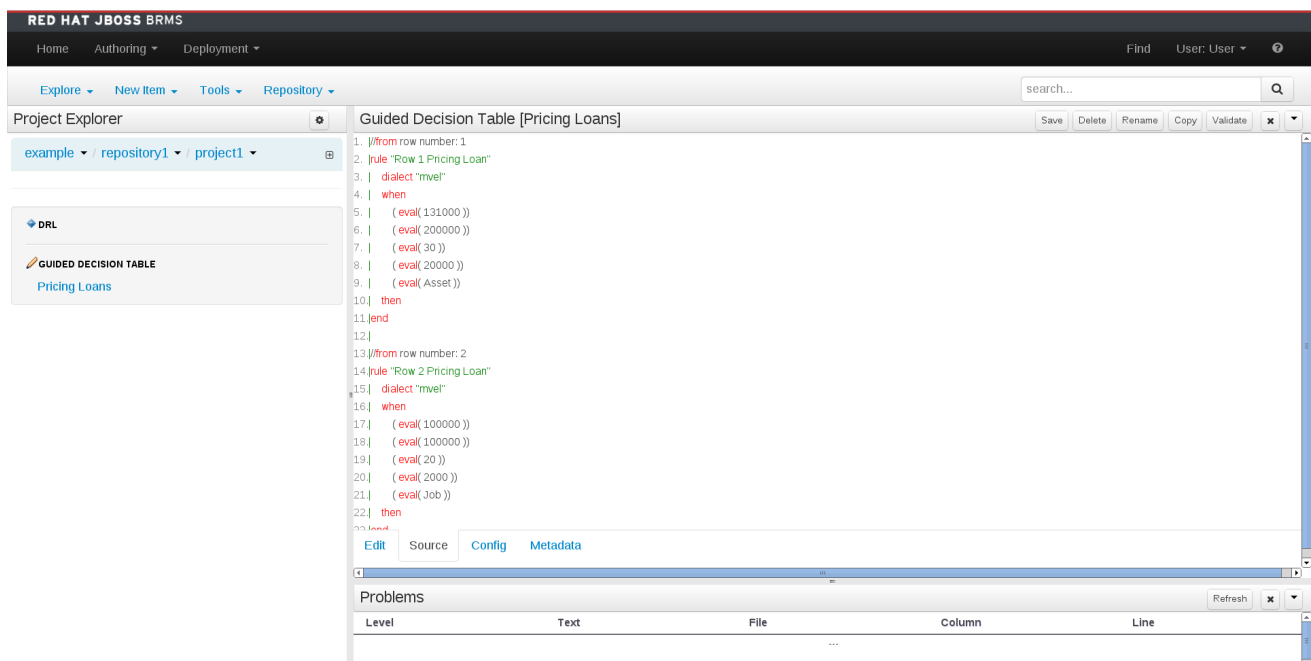


Figure 5.2. The asset editor - Source tab

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.

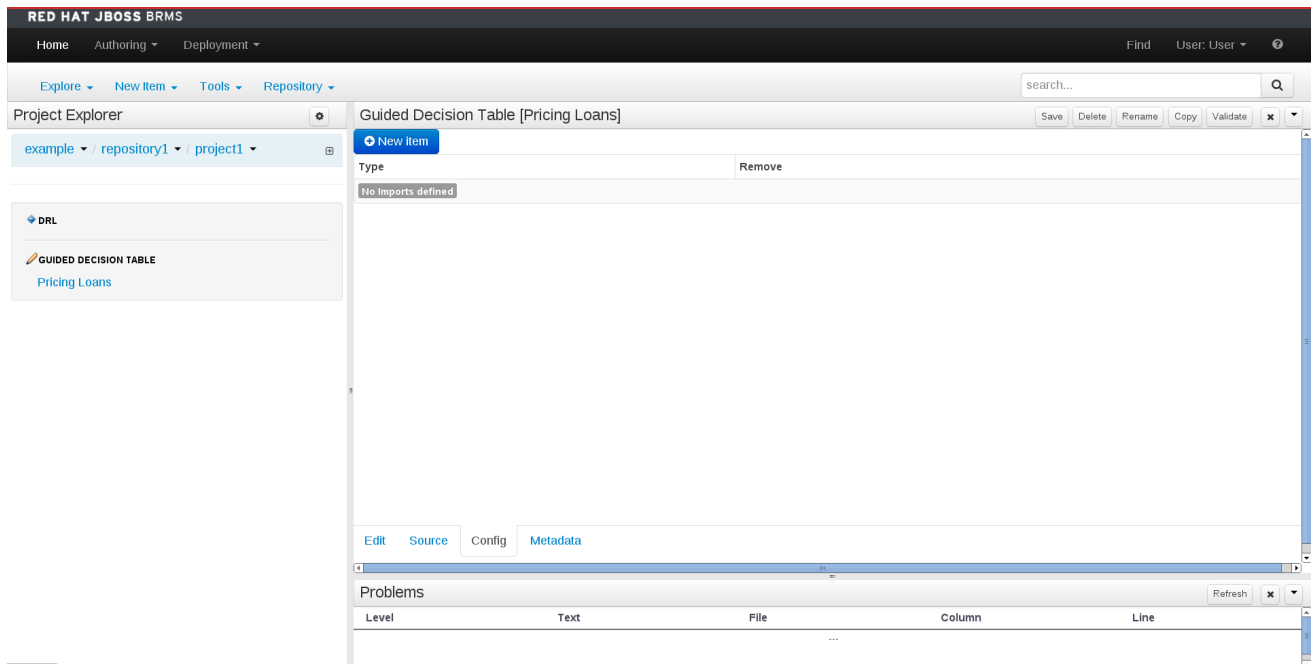


Figure 5.3. The asset editor - Config tab

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.

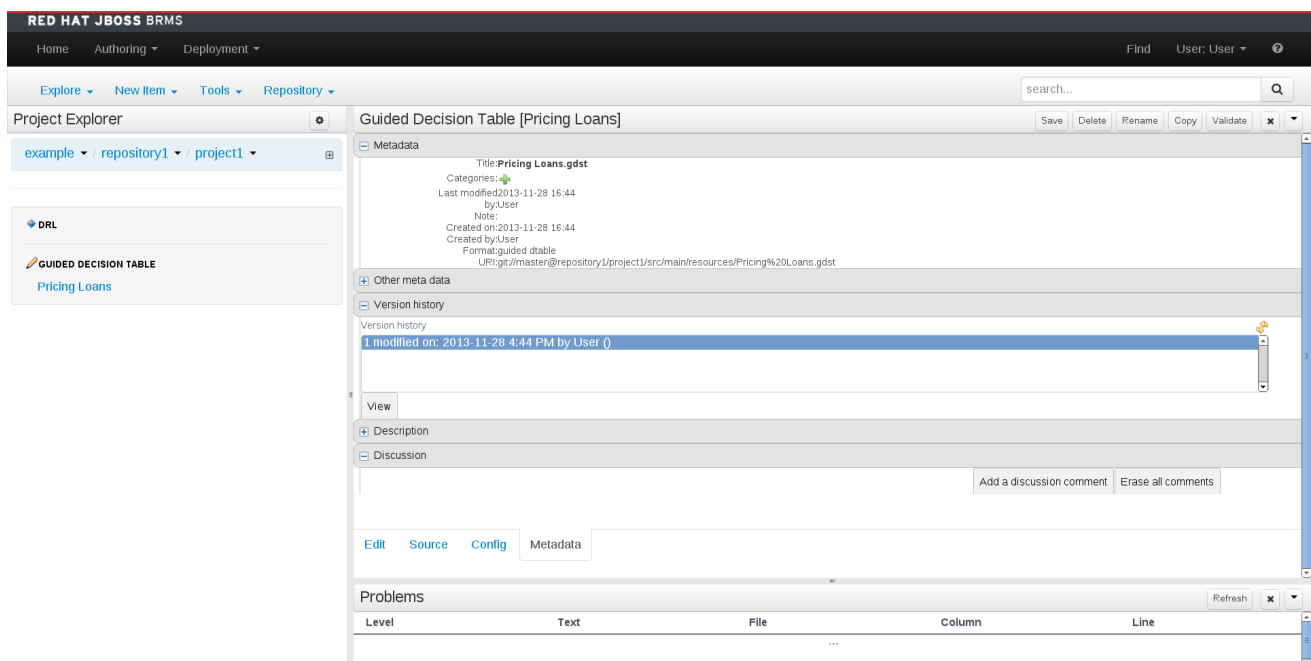


Figure 5.4. The asset editor - Metadata tab

[Report a bug](#)

5.2.2. 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 5.1. The guided editor

The screenshot displays the BRL guided editor interface. It is divided into two main sections: **WHEN** (conditions) and **THEN** (actions).

WHEN Section:

- 1. There is a LoanApplication [application]
- 2. There is an Applicant with:
 - age: less than (dropdown menu) 21

THEN Section:

- 1. Set value of LoanApplication [application]
 - approved: false (dropdown menu)
 - explanation: Underage (text input field)
- 2. Retract LoanApplication [application]

At the bottom left of the THEN section, there is a link: (show options...). On the right side of the editor, there are several icons for adding, deleting, and reordering rules, and a vertical scrollbar.

[Report a bug](#)

5.2.3. 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.

[Report a bug](#)

5.2.4. 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.

[Report a bug](#)

5.2.5. Adding Conditions or Actions to Rules

Procedure 5.2. 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.

[Report a bug](#)

5.2.6. 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 5.3. 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.

[Report a bug](#)

5.2.7. Technical Rules (DRL)

Technical (DRL) rules are stored as text and can be managed in the JBoss Enterprise 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.

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 Enterprise 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 5.5. Technical Rule (DRL)

[Report a bug](#)

5.3. DECISION TABLES

5.3.1. 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.

[Report a bug](#)

5.3.2. Uploading Spreadsheet Decision Tables

Procedure 5.4. 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.

[Report a bug](#)

5.3.3. 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 5.1. 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 5.2. 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	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 5.3. 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.

[Report a bug](#)

5.4. WEB BASED GUIDED DECISION TABLES

5.4.1. 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 5.4.2, “Types of decision tables”](#)). Optionally select to use the Guided Decision Table Wizard.

Create new Guided Decision Table ×

* Resource Name

Location

☐ Use Wizard

☒ Extended entry, values defined in table body

☐ Limited entry, values defined in columns

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

[Report a bug](#)

5.4.2. 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.

[Report a bug](#)

5.4.3. 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.

#	Description	amount min	amount max	period	deposit max	income	Loan approved	LMI	rate
1		131000	200000	30	20000	Asset	true	0	2
2		10000	100000	20	2000	Job	true	0	4
3		100001	130000	20	3000	Job	true	10	6

Figure 5.6. Column Configuration

[Report a bug](#)

5.4.4. 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:

Type of column:

- Add a simple Condition
- Add a Condition BRL fragment
-
- Set the value of a field
- Set the value of a field on a new fact
- Delete an existing fact
- 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

☒ Include advanced options

OK

Figure 5.7. 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

[Report a bug](#)

5.4.5. Column Types

5.4.5.1. 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
```

[Report a bug](#)

5.4.5.2. Metadata Columns

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

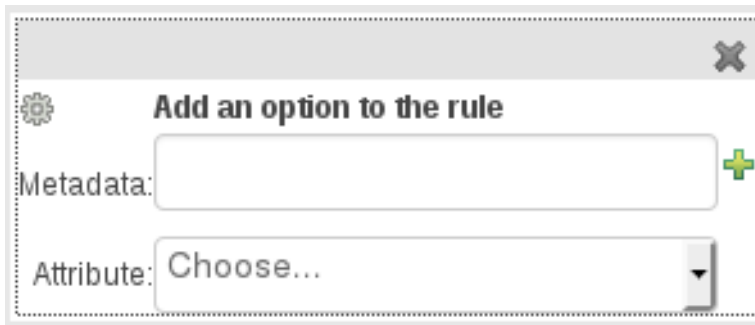


Figure 5.8. Attribute and MetaData Option

[Report a bug](#)

5.4.5.3. 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.

Pattern:

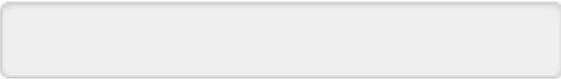

Calculation type: ☒ Literal value ☐ Formula ☐ Predicate

Field: (please select a pattern first)

Operator: (please select a pattern first)

From Entry Point: |

Column header (description):

(optional) value list:  

Binding:

Hide column: ☐

Apply changes

Figure 5.9. Simple Condition Column

[Report a bug](#)

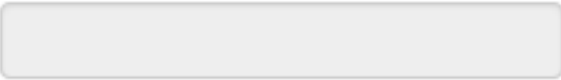

5.4.5.4. Field Value Columns


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.

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

Field: (please choose a fact pattern first)

Column header (description): |

(optional) value list:  

Update engine with changes: ☐ 

Hide column: ☐

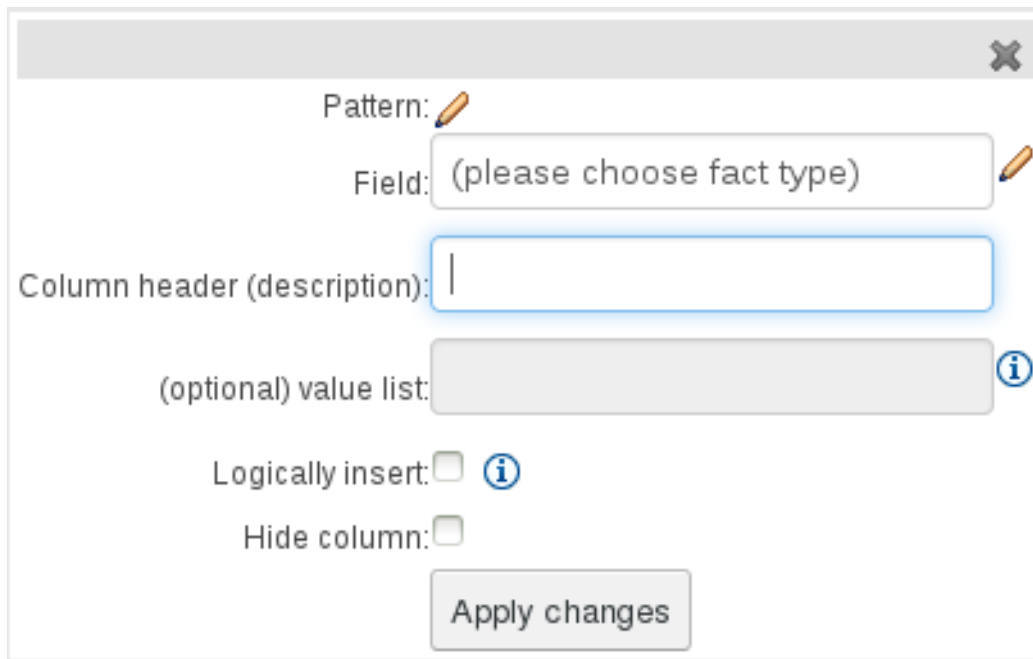
Apply changes


Figure 5.10. Set the value of a field


[Report a bug](#)

5.4.5.5. 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.




Pattern: 

Field: (please choose fact type) 

Column header (description):

(optional) value list: 

Logically insert: ☐ 

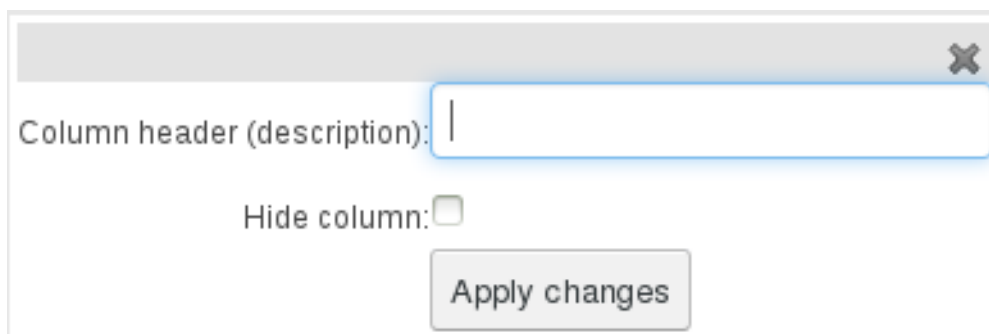
Hide column: ☐

Figure 5.11. Set the value of a field on a new fact

[Report a bug](#)

5.4.5.6. Delete Existing Fact Columns

The implementation of an Action to delete a bound Fact.



Column header (description):

Hide column: ☐

Figure 5.12. Delete an existing fact

[Report a bug](#)

5.4.6. Advanced Column Types

5.4.6.1. 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.

Column header (description):

Hide column: ☐

WHEN

1. If customer spends \$

2. There is a ShoppingCart ☐

3. From Entry Point

Any of the following are true:

3. There is a ShoppingCart with: greater than

Figure 5.13. Add a Condition BRL fragment

[Report a bug](#)

5.4.6.2. Execute Work Item Columns

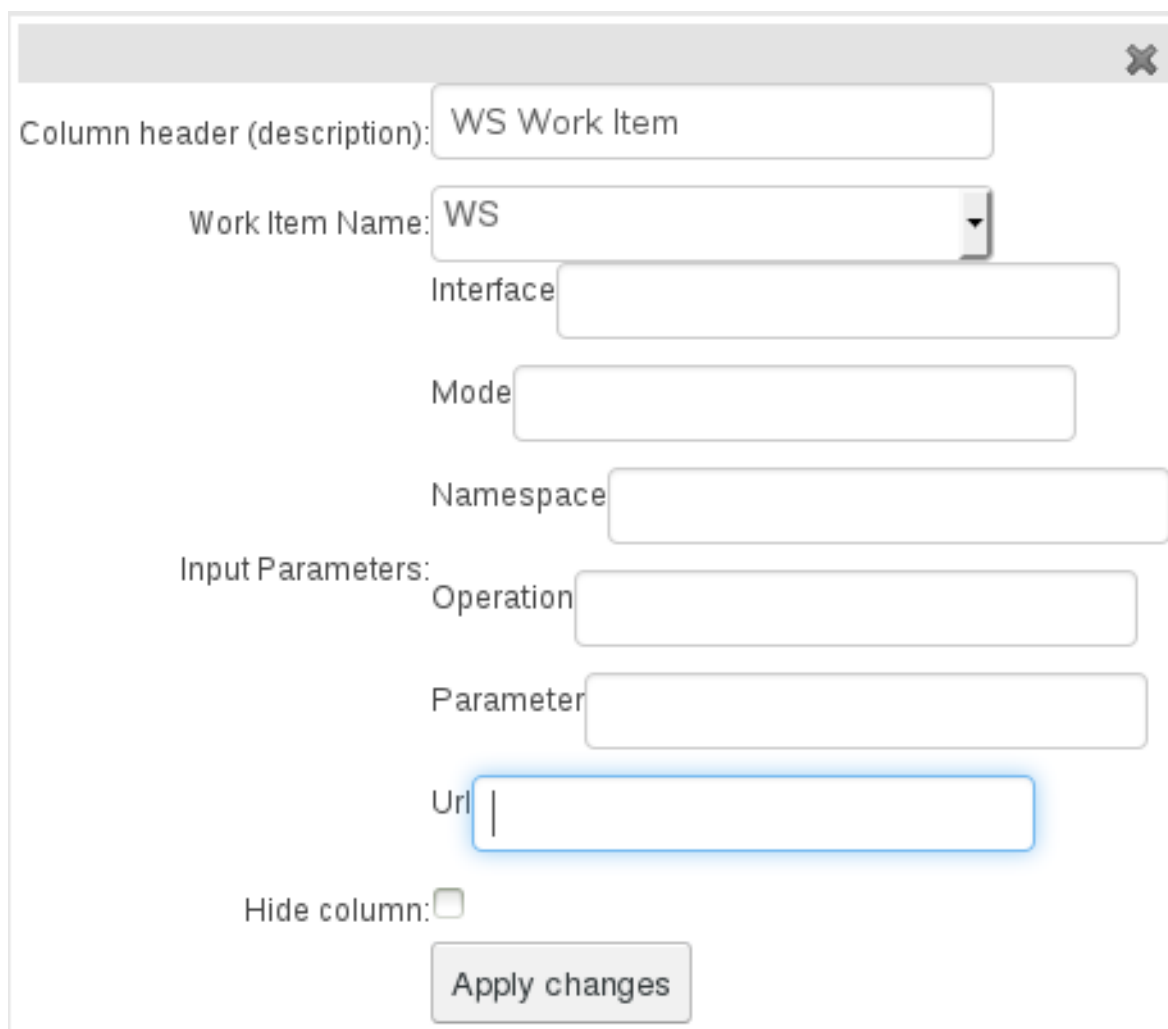
This implements an Action to invoke a Red Hat JBoss Business Process Management Suite Work Item Handler. It sets its input parameters to bound Facts/Facts' fields values. This works for any Work Item Definition.

Column header (description):

Work Item Name:

Hide column: ☐

Figure 5.14. Execute a Work Item



A configuration dialog box titled "WS Work Item" with a close button (X) in the top right corner. The dialog contains several input fields and a checkbox. The "Column header (description)" field is set to "WS Work Item". The "Work Item Name" field is a dropdown menu currently showing "WS". Below it are text fields for "Interface", "Mode", and "Namespace". Under the "Input Parameters:" label, there are text fields for "Operation" and "Parameter". The "Url" field is currently empty and has a blue highlight. At the bottom left is a "Hide column:" checkbox, which is unchecked. At the bottom center is an "Apply changes" button.

Column header (description): WS Work Item

Work Item Name: WS

Interface

Mode

Namespace

Input Parameters:

Operation

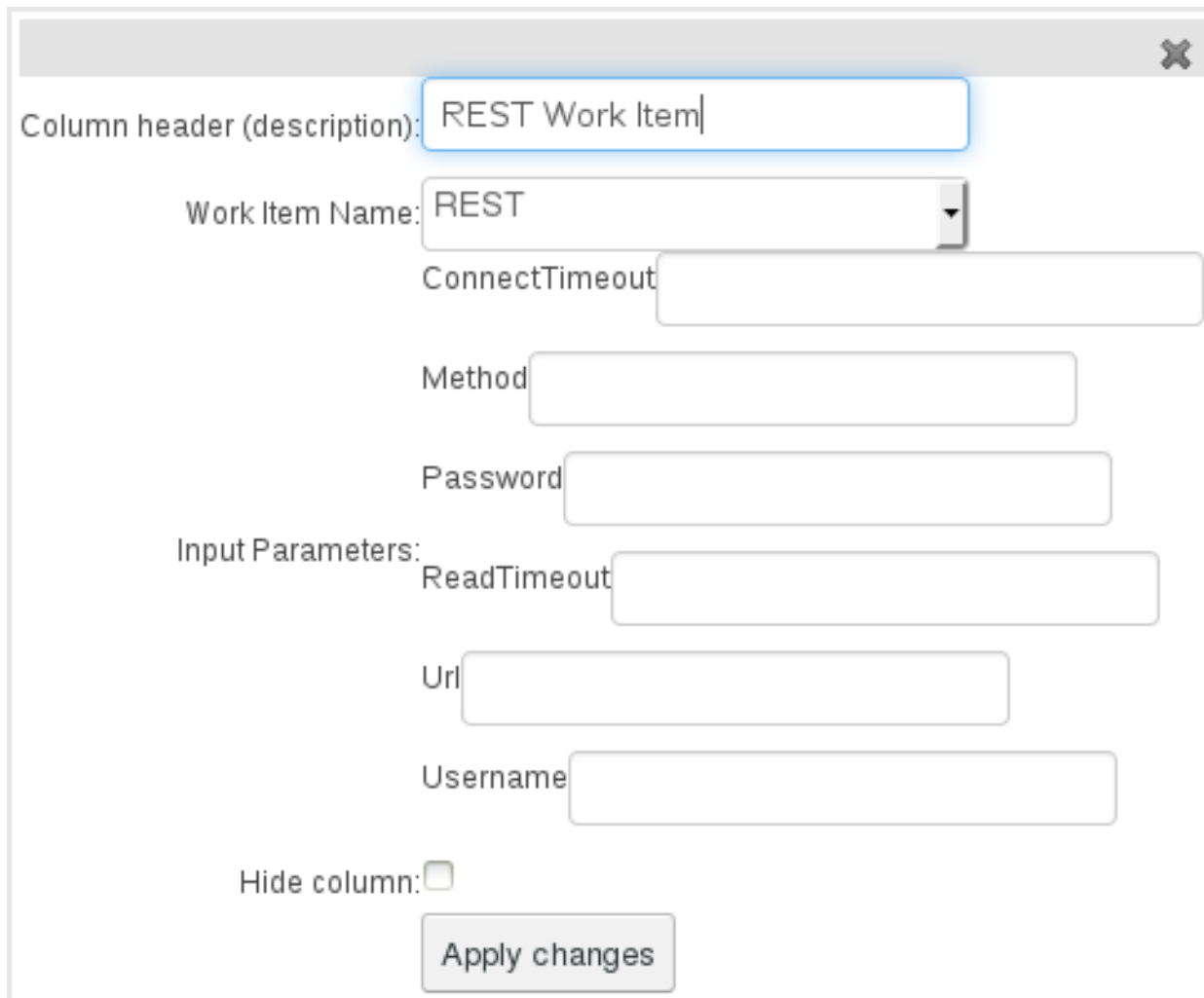
Parameter

Url

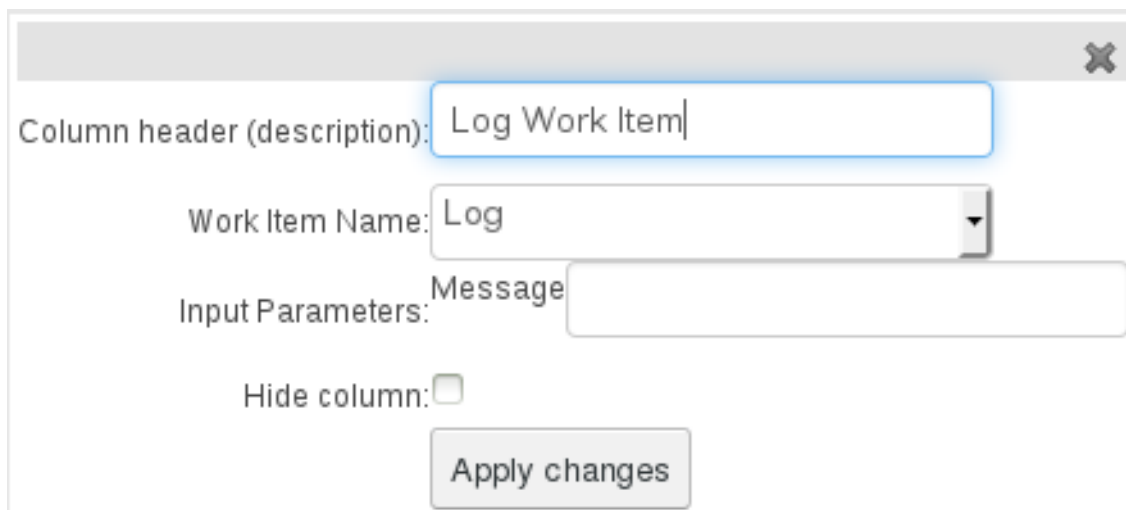
Hide column: ☐

Apply changes

Figure 5.15. WS Work Item

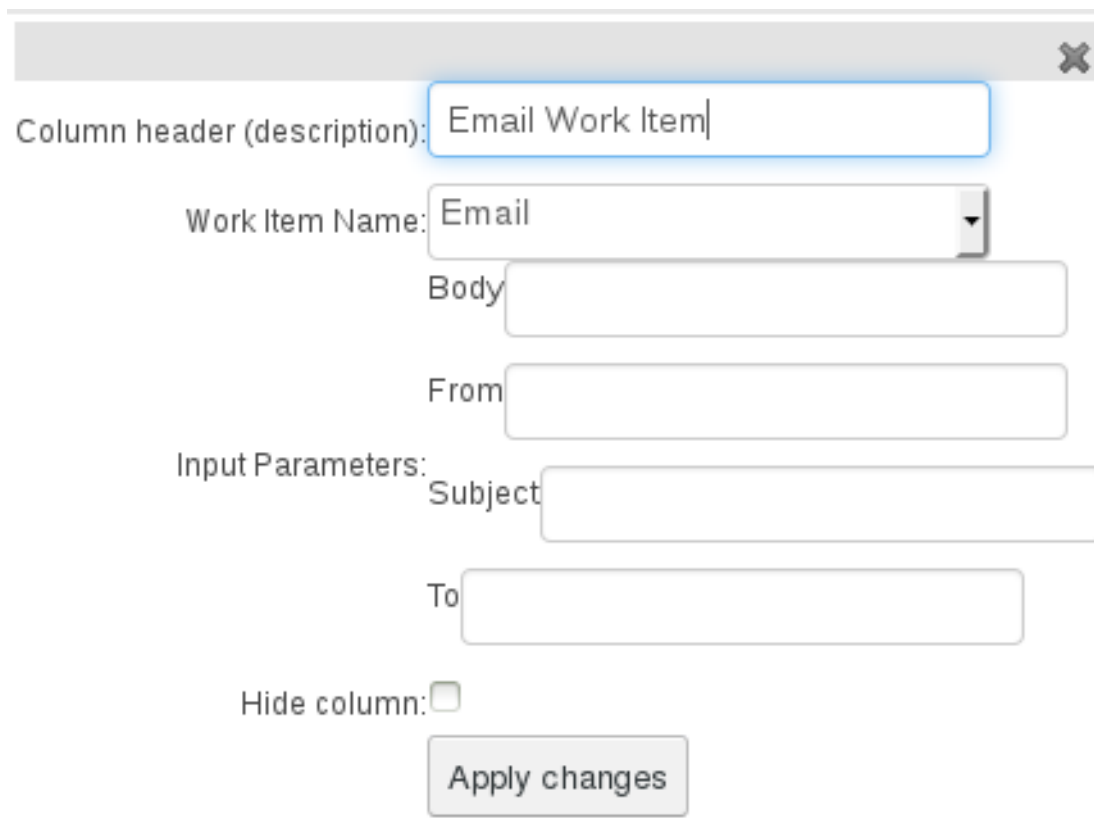


A screenshot of a configuration dialog box titled "REST Work Item". The dialog has a close button (X) in the top right corner. It contains several input fields and a checkbox. The "Column header (description):" field is highlighted with a blue border and contains the text "REST Work Item". Below it, 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 5.16. 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 5.17. Log Work Item



Column header (description): Email Work Item

Work Item Name: Email

Body

From

Input Parameters: Subject

To

Hide column: ☐

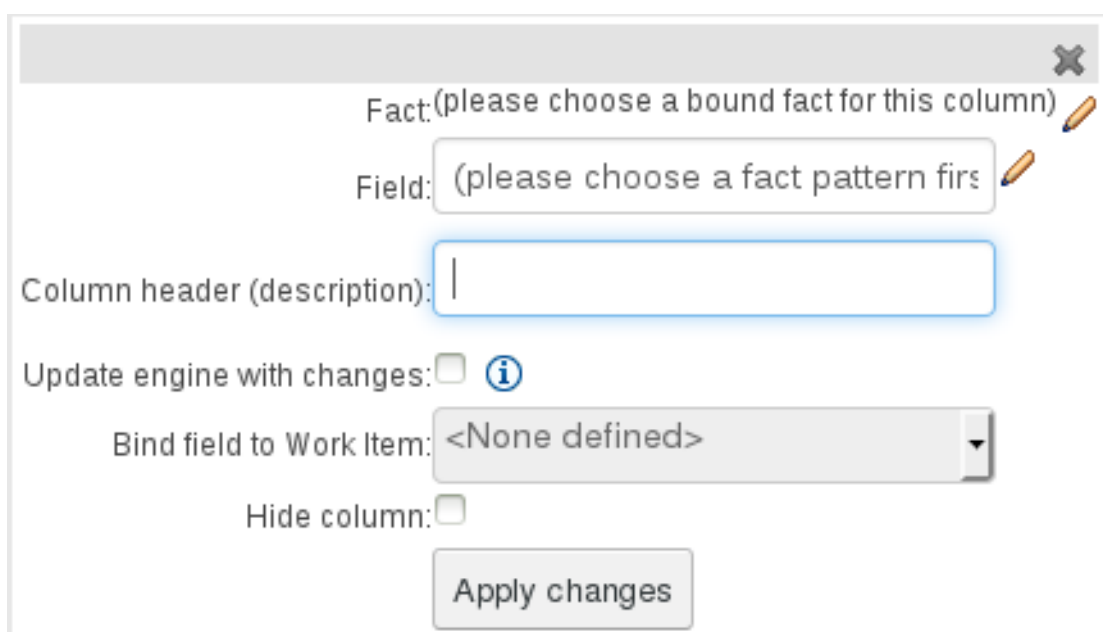
Apply changes

Figure 5.18. Email Work Item

[Report a bug](#)

5.4.6.3. 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 5.19. 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.

[Report a bug](#)

5.4.6.4. 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 5.20. 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.

[Report a bug](#)

5.4.6.5. 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 +

+

OkCancel

Figure 5.21. Simple layout for Adding an Action BRL fragment

[Report a bug](#)

5.4.7. 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 5.22. Rule Definition

[Report a bug](#)

5.4.8. 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 5.23. Cell Merging

[Report a bug](#)

5.4.9. 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 5.24. Cell Grouping

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

[Report a bug](#)

5.4.10. 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
```

[Report a bug](#)

5.5. RULE TEMPLATES

5.5.1. 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 5.5. 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.

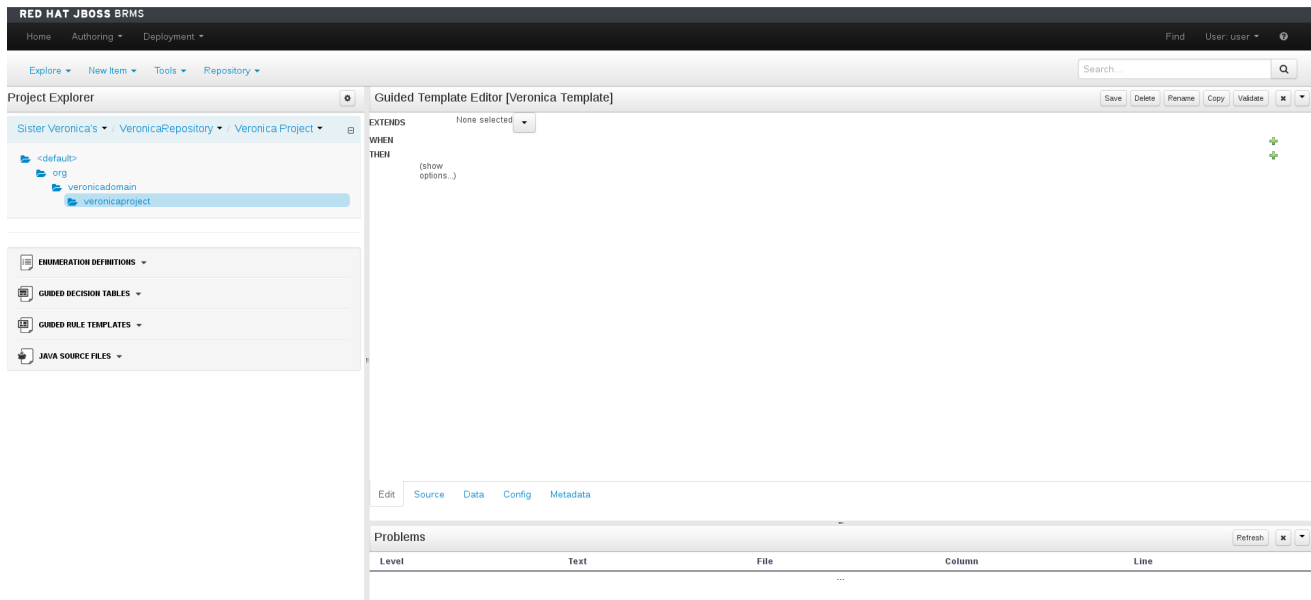


Figure 5.25. Guided Template Editor



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.

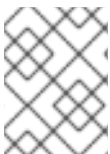
[Report a bug](#)

5.5.2. 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 5.6. 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 4.2, “Creating a data object”](#)), select the plus icon  to the right of the **WHEN** section of the Guided Template Editor.
2. A dialog window will appear with available condition templates to choose from. In the example below, we select the **Nurse . . .** condition from the list.

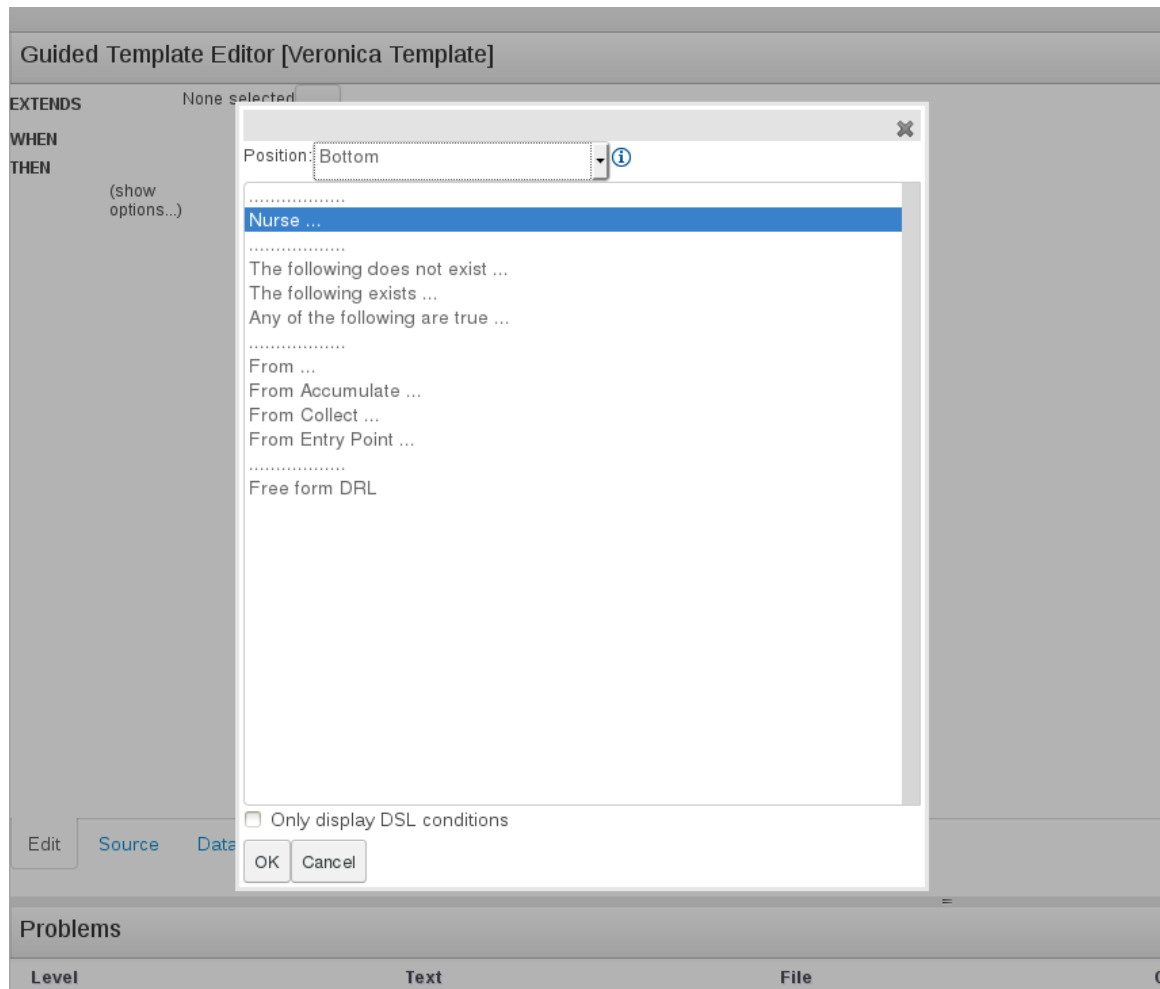


Figure 5.26. Nurse Roster WHEN Dialog Window

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.

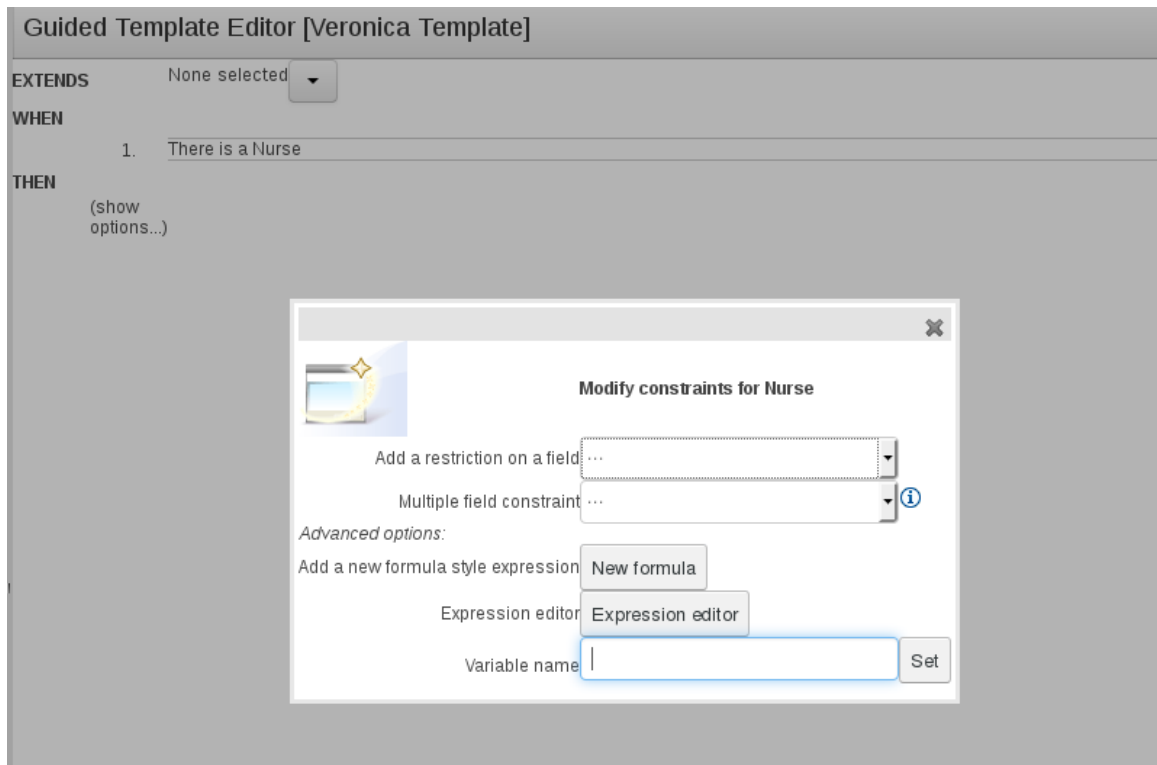


Figure 5.27. Modify Constraints Dialog

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.

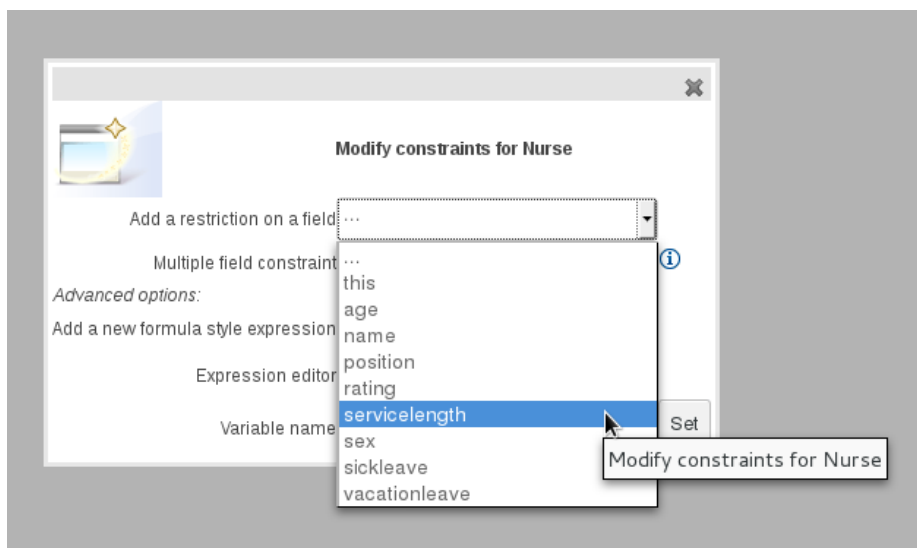


Figure 5.28. Title

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."

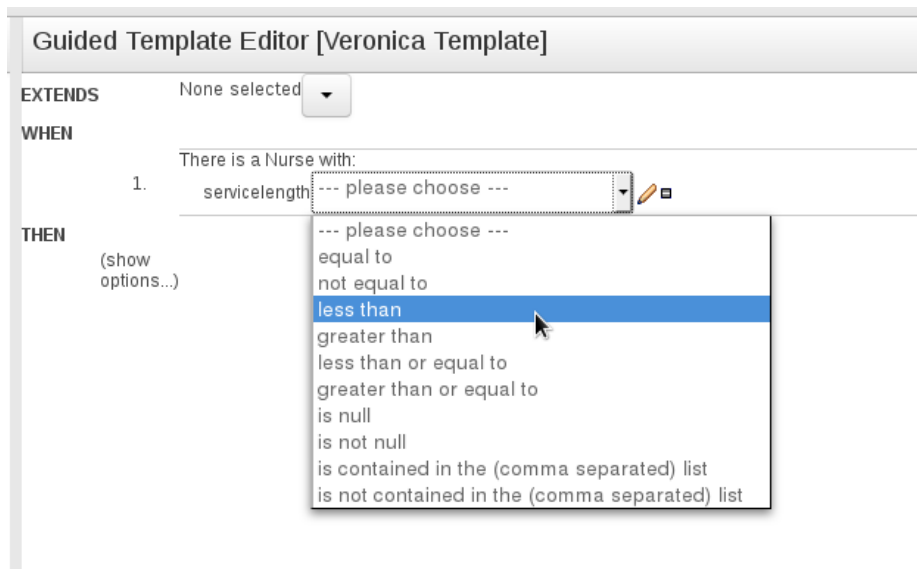



Figure 5.29. Restriction drop-down menu

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.

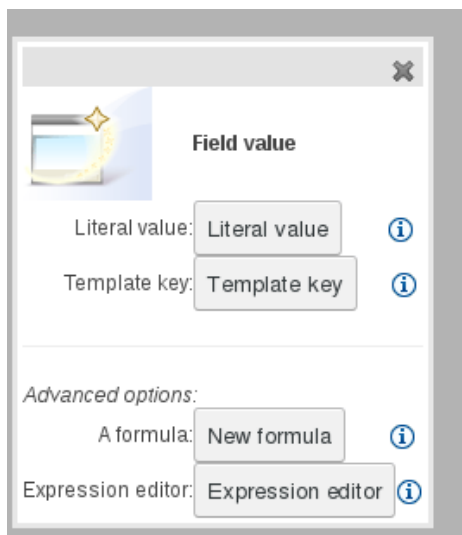


Figure 5.30. Field Value Options

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.

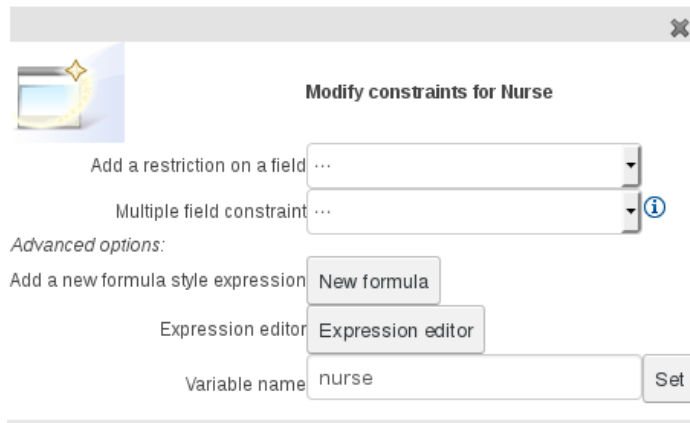


Figure 5.31. Setting a variable name

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

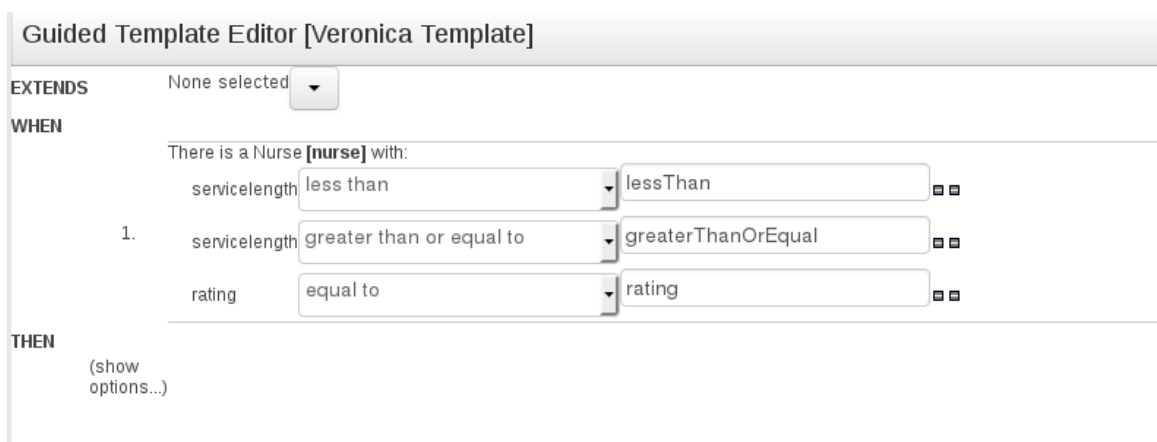



Figure 5.32. WHEN Constraints

[Report a bug](#)

5.5.3. 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 5.7. Using the Guided Template Editor with THEN actions

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

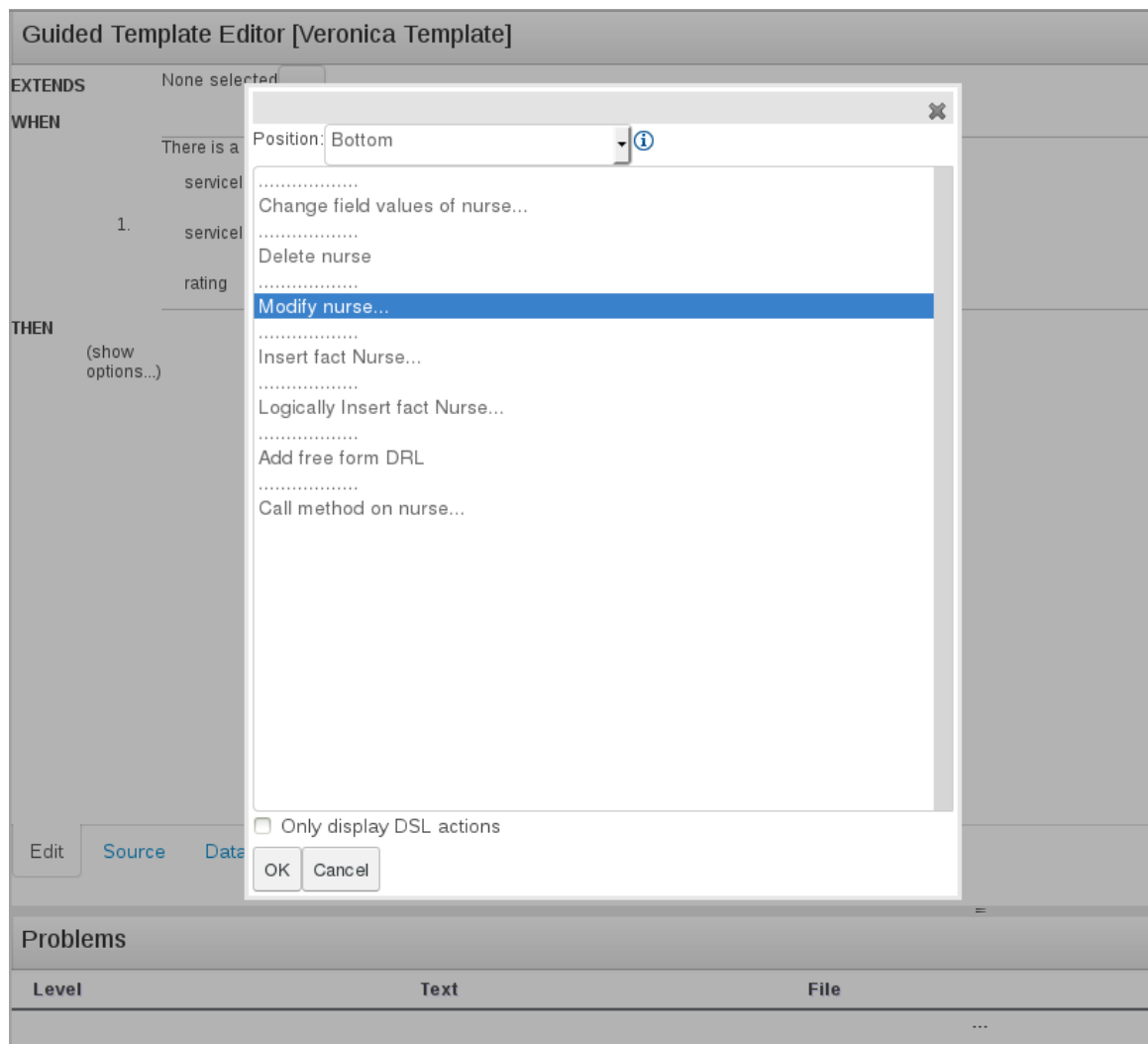



Figure 5.33. Nurse Roster THEN Dialog Window

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.

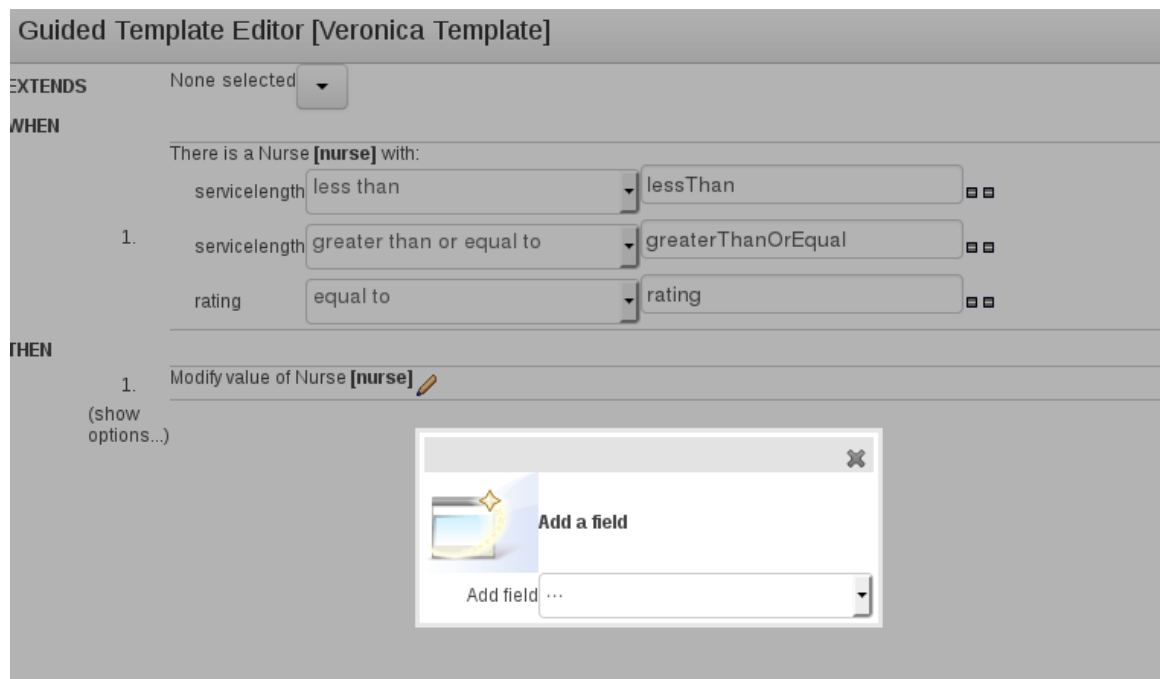


Figure 5.34. Add a field Dialog

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.

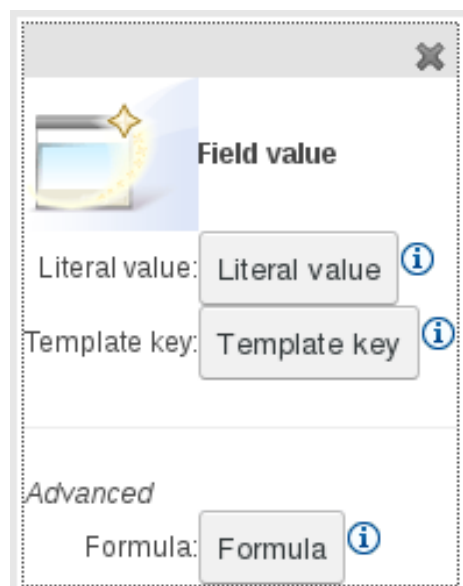


Figure 5.35. Field Value Options

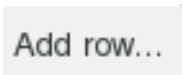
[Report a bug](#)

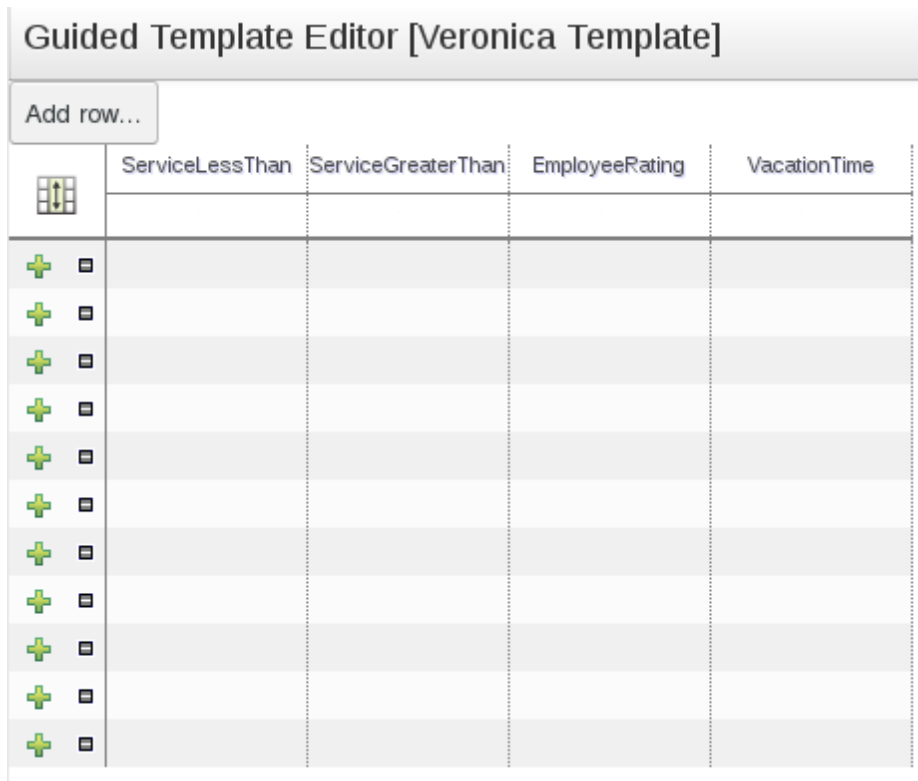
5.5.4. Data Tables in the Guided Rule Template

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 5.5.4.1. Using the Guided Template Editor with Data Tables

Procedure 5.8. 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.
























Add row...				
	ServiceLessThan	ServiceGreaterThan	EmployeeRating	VacationTime
				
 				
 				
 				
 				
 				
 				
 				
 				
 				
 				

Figure 5.36. 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.

Guided Template Editor [Veronica Template]

```

1. |package org.veronicadomain.veronicaproject;
2. |
3. |rule "Veronica Template_9"
4. |     dialect "mvel"
5. |     when
6. |         nurse : Nurse( servicelength < 5 , rating == "Average" )
7. |     then
8. |         nurse.setVacationleave( 0 );
9. |         update( nurse );
10.|end
11.|
12.|rule "Veronica Template_8"
13.|     dialect "mvel"
14.|     when
15.|         nurse : Nurse( servicelength >= 5 , rating == "Bad" )
16.|     then
17.|         nurse.setVacationleave( 39 );
18.|         update( nurse );
19.|end
20.|
21.|rule "Veronica Template_7"
22.|     dialect "mvel"
23.|     when
24.|         nurse : Nurse( servicelength < 5 , rating == "Average" )
25.|     then
26.|         nurse.setVacationleave( 7 );

```

Edit
Source
Data
Config
Metadata

Figure 5.37. Source Code for Nurse Example

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

[Report a bug](#)

5.6. 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.

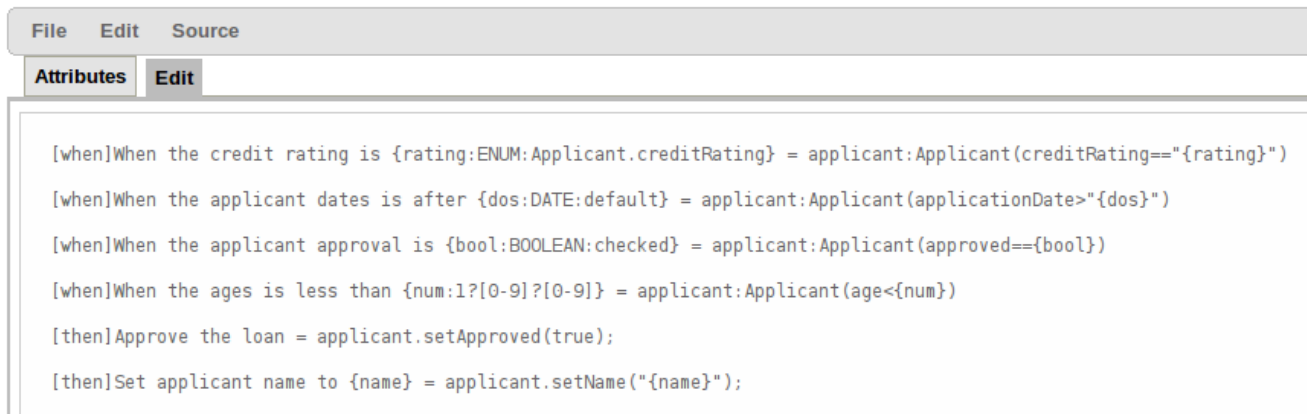


Figure 5.38. DSL Editor

[Report a bug](#)

5.7. DATA ENUMERATIONS

5.7.1. 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 5.2. An Example Enumeration Configuration

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

[Report a bug](#)

5.7.2. 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']
```

[Report a bug](#)

5.7.3. 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.

[Report a bug](#)

5.8. SCORECARDS

5.8.1. 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 5.39. Scorecard Example

[Report a bug](#)

5.8.2. Creating a Scorecard

Procedure 5.9. 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.

[Report a bug](#)

CHAPTER 6. 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:

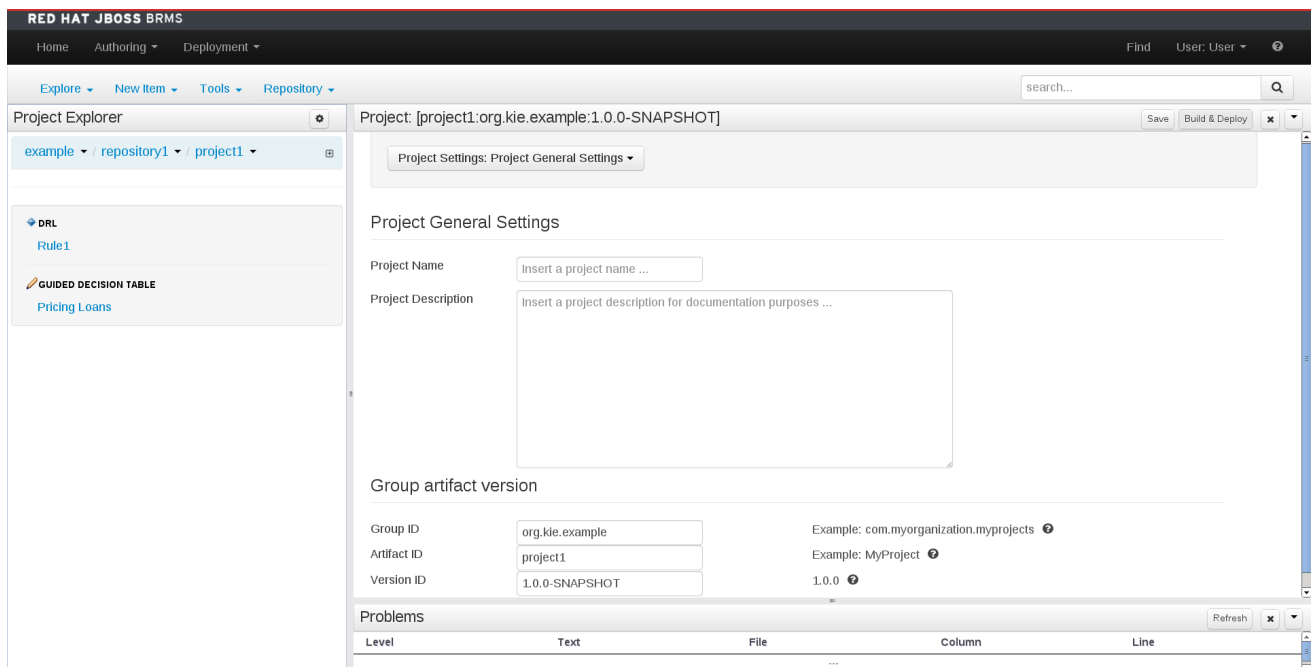


Figure 6.1. Build & Deploy Option

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.

[Report a bug](#)

CHAPTER 7. MANAGING ASSETS

7.1. CATEGORIES

Rules can be assigned to one or more categories before or after the rule is created. Categories are useful for organizing rules into meaningful groups, and as such, should have meaningful names that relate to some aspect of the business or the rule's life-cycle. For instance, having **Draft** and **Review** categories make it possible to tag a rule so that it is clear exactly where the rule is in its life-cycle.

[Report a bug](#)

7.2. 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.

[Report a bug](#)

7.3. DISCUSSION

The asset editor includes a discussion area where comments can be left regarding any changes that have been made to assets. Each comment is recorded along with the identity of the user making the comment and the date and time of the comment. Administrators can clear all comments on an asset, but other users can only append comments.

The screenshot shows a web interface for 'Project Settings: Metadata'. It features a list of expandable sections: 'Metadata', 'Other meta data', 'Version history', 'Description', and 'Discussion'. The 'Discussion' section is expanded, revealing a comment area. A comment is displayed: 'Comment by User on Fri Aug 23 15:41:54 GMT+530 2013: This will need to be updated next year when we expand into new areas as different regions have different laws about minimum age.' Below the comment are two buttons: 'Add a discussion comment' and 'Erase all comments'.

Figure 7.1. Discussion

[Report a bug](#)

CHAPTER 8. TESTING

8.1. 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.

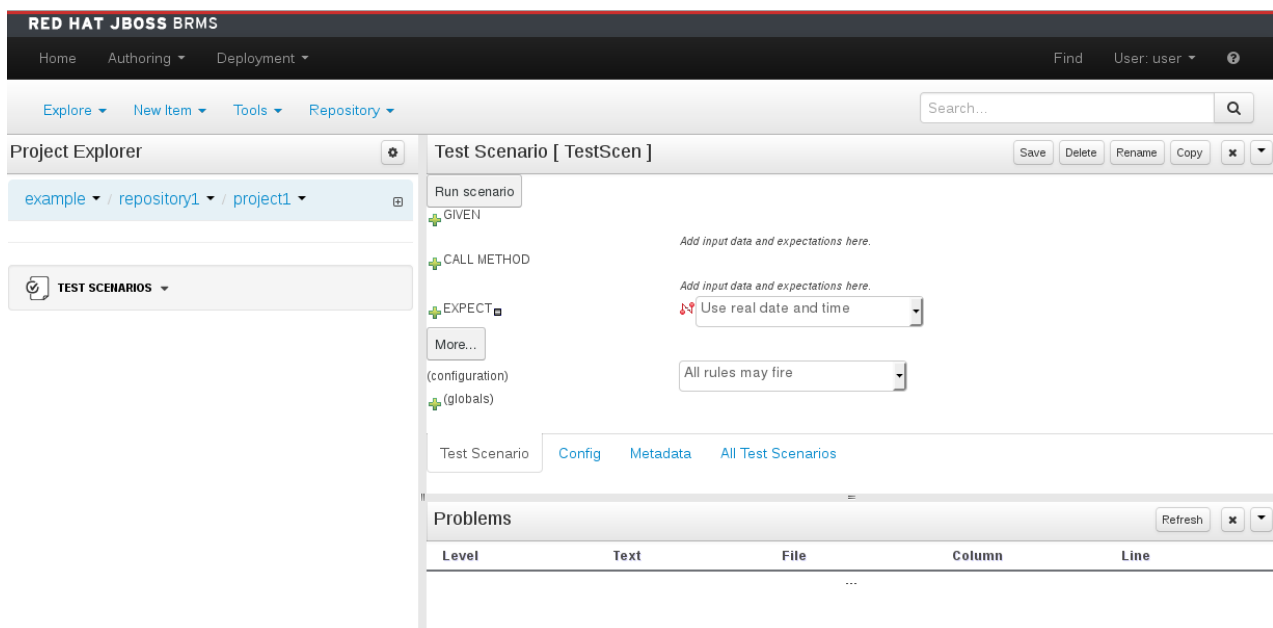


Figure 8.1. Test Scenario Screen

[Report a bug](#)

8.2. CREATING A TEST SCENARIO

Creating a Test Scenario requires you to provide a 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 8.1. Creating a new Test Scenario

1. Open the **Projects** view from the **Authoring** menu.
2. Select the project where the test scenario is to be created.
3. From the **New** 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 8.2. 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. Facts/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 fact/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 do this, you will need to import the model objects required for your Scenario by clicking 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 8.3. Providing Test Scenario conditions

1. After you have imported the model 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**
 - o **GIVEN**: What are the input facts for this Test Scenario?
 - o **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 8.4. Providing Given Facts

1. To add a new condition, 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 project models that you have imported in the **Config** screen.

You can select a particular fact/data object from the model, 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 both 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.

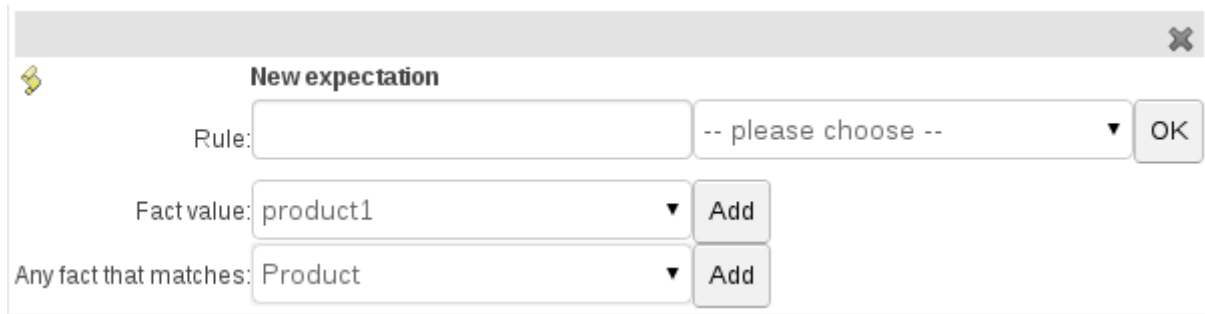
2. Depending upon the model object that you select, you will be able to provide values to its editable properties as part of your GIVEN fact. For example, if your model was a **Product**, you might have properties like **itemID**, **productName** and **price**. You get to these properties by clicking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

Procedure 8.5. Providing Expected Rules

1. Once you are satisfied with the Given fact conditions, you can provide the expected rules that will be fired if the Given fact conditions are met when the Test Scenario is run.
2. To do so, click on the green + button next to the **Expect** label. A **New expectation** dialog box will come up.



3. You can provide one of three expectations given the set of data that was created in the **Given** section. You can:
 - Either type in the name of a rule that is expected to be fired or select it from the list of rules and then click the **OK** button.
 - Expect a particular instance of a model object (and one or more of its properties) to have a certain value by selecting that instance from the drop down in the **Fact Value** field. For example, **product1** shown in the figure, which is an instance of the fictitious **Product** model created in the **Given** section. You specify the property values by first adding that instance by clicking the **Add** button and then clicking a green arrow  to bring up the fields to add. Once you have selected the field to add, you can provide a literal value for that field.
 - Expect instances of a fact model to match your **Given** conditions by selecting it from the **Any fact that matches** drop down. In the figure shown above, you can mandate that instances of **Product**'s one or more properties match the **Given** conditions. For example, You have a fact 'A' with a String field 'fieldA'. In the test scenario, you set two given facts: 'A' bound to variable 'a1' and 'A' bound to variable 'a2'. Fact 'a1' will have 'fieldA' with value "something", fact 'a2' will have 'fieldA' with value "else". You will add an expectation - any fact that matches A. Add 'fieldA' with value "something" to this expectation. Save the test scenario and run it. It will pass (correctly), even though only 'a1' matches the expectation.

Procedure 8.6. Reviewing, Saving, and Running a Scenario

1. Once you are satisfied with your Test Scenario's conditions, you can save it by clicking the **Save** button in the upper right corner. Make sure to regularly save and review your scenarios.
2. You can now run your Test Scenario by clicking the **Run Scenario** button at the top of the Test Scenario screen. The results are displayed at the bottom of this screen in a new panel called **Reporting**.



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

[Report a bug](#)

CHAPTER 9. 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 BRMS provides 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.

All REST API calls to Red Hat JBoss BRMS 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.

[Report a bug](#)

9.1. 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. The job ID is returned by most of the Knowledge Store REST calls, and this is used to request the job status and verify whether the job finished successfully. Other operations return objects like repository lists and organizational units.

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

[Report a bug](#)

9.1.1. 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 9.1. 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,"detailedResult":null}"
```

[DELETE] /jobs/{jobID}

removes the job - [DELETE]

[Report a bug](#)

9.1.2. 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 9.2. 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"}]
```

[DELETE] /repositories/{repositoryName}

This removes the repository from the Knowledge Store - [DELETE]

[POST] /repositories/

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

Example 9.3. JSON entity with repository details of a repository to be cloned

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

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

This creates a project in the repository - [POST]

Example 9.4. 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 9.5. Request body that defines the project to be deleted

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

[Report a bug](#)

9.1.3. 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].

[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 9.6. Organizational unit in JSON

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

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

This adds the repository to the organizational unit - [POST]. It also returns a **AddRepositoryToOrganizationalUnitRequest** instance.

**NOTE**

Deleting an organizational unit is not supported via the REST API. The removal of an organizational unit is only possible through the Business Central.

[Report a bug](#)

9.1.4. 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.

[Report a bug](#)

9.2. REST SUMMARY

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

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

Table 9.1. Knowledge Store REST calls

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

URL Template	Type	Description
/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

[Report a bug](#)

APPENDIX A. REVISION HISTORY

Revision 1.0.0-21

Fri Nov 21 2014

Vikram Goyal

Built from Content Specification: 22698, Revision: 727049 by vigoyal