

Guvnor User Guide

For users and administrators of Guvnor

Version 5.4.0.CR1

by *The JBoss Drools team* [<http://www.jboss.org/drools/team.html>]

1. Introduction	1
1.1. What is a Business Rules Manager?	1
1.1.1. When to use Guvnor	1
1.1.2. Who uses Guvnor	2
1.2. Features outline	2
I. User Guide	3
2. Quick Start Guide	5
2.1. Quick start guide	5
2.1.1. Supported browser platforms	7
2.1.2. Initial configuration	7
2.1.3. Writing some rules	8
2.1.4. Finding stuff	8
2.1.5. Deployment	8
3. Concepts	11
3.1. Rules are assets	11
3.2. Packages are assets	11
3.3. Perspectives	11
3.4. The business user perspective	11
3.4.1. Creating a business user view	12
4. Authoring Assets	13
4.1. Version management	13
4.2. The Asset Editor	13
4.3. Creating a package	16
4.3.1. Empty package	17
4.3.2. Importing DRL packages	17
4.4. Spring Contexts	18
4.5. Working Sets	19
4.5.1. Activating and Using Working Sets	22
4.6. Business rules with the guided editor	24
4.6.1. User driven drop down lists	28
4.6.2. Augmenting with DSL sentences	28
4.6.3. A more complex example:	29
4.7. DSL rules	32
4.8. Technical rules (DRL)	33
4.9. Spreadsheet decision tables	34
4.10. Guided decision tables (web based)	34
4.10.1. Types of decision table	34
4.10.2. Main components\concepts	36
4.10.3. Defining a web based decision table	39
4.10.4. Rule definition	49
4.11. Templates of assets/rules	50
4.11.1. Creating a rule template	50
4.11.2. Define the template	52
4.11.3. Defining the template data	53

4.11.4. Generated DRL	57
4.12. The Fact Model	58
4.12.1. Ways to define a Fact Model	58
4.12.2. Creating a JAR Model	59
4.12.3. Declarative model	61
4.13. Functions	67
4.14. DSL editor	67
4.15. Rule flows	68
4.16. BPMN2 Process	68
4.17. Work Item Definition	68
4.18. Data enumerations (drop down list configurations)	69
4.18.1. Advanced enumeration concepts	70
4.19. Test Scenario	71
4.19.1. Given Section	73
4.19.2. Expect Section	74
4.19.3. Global Section	74
4.19.4. New Input Section	75
4.19.5. Audit Log	75
4.20. File	75
4.21. Change-Set	78
4.21.1. Change-Set Editor UI	79
4.21.2. Adding Packages using a Wizard	79
4.21.3. Adding Assets using a Wizard	80
4.22. Services	81
4.22.1. Configuring Multiple Knowledge Bases	82
4.22.2. Knowledge Base Configuration Panel	83
4.22.3. Downloading War & Managing Dependencies	88
5. Managing Assets	93
5.1. Navigating and finding rules	93
5.1.1. Package explorer	93
5.1.2. Category view	96
5.1.3. Inbox and comments	97
5.2. Feed	98
6. Quality Assurance	99
6.1. Test scenarios	99
6.2. Package analysis	100
6.2.1. Items that the analyser is searching for	100
7. Packaging	103
7.1. Packaging	103
7.2. Imports and Globals	104
7.2.1. Basic View	105
7.2.2. Advanced View	106
7.3. Category rules	107
7.4. Building	107

7.5. Selectors	107
7.5.1. Built-in Selector	108
7.5.2. Custom Selector	108
7.6. Snapshots	108
7.7. Advanced configuration	109
8. Administrative Functions	111
8.1. Categories	111
8.2. Status management	113
8.3. Archived items	114
8.4. Event Log	114
8.5. User permissions	115
8.6. Import and Export	116
8.7. Repository Configuration	117
II. Developer Guide	119
9. Integrating rules with your applications	121
9.1. The Knowledge Agent	121
9.2. REST API	123
9.2.1. REST	123
9.2.2. Guvnor REST API	124
9.2.3. Source code Example	133
9.3. WebDAV and HTTP	136
9.3.1. WebDAV	136
9.3.2. URLs	136
9.4. Eclipse Guvnor integration	137
9.4.1. Source Code and Plug-in Details	137
9.4.2. Functionality Overview	137
9.4.3. Guvnor Connection Wizard	139
9.4.4. Guvnor Repository Explorer	142
9.4.5. Local Copies of Guvnor Files	144
9.4.6. Actions for Local Guvnor Resources	146
9.4.7. Importing Guvnor Repository Resources	151
9.4.8. Guvnor plugin Preferences	155
10. Embedding Guvnor In Your Application	157
10.1. Getting Started	157
10.2. Embedded Editor Entry-Point: StandaloneEditorServlet	157
10.3. Edition Modes	158
10.3.1. BRL Edition Mode	158
10.3.2. Edition of Existing Assets Mode	160
10.3.3. New Asset Mode	162
10.4. Extra HTTP parameters	163
10.4.1. Rule's Sections Visibility Parameters	163
10.4.2. Constraining Fact Types	164
10.4.3. Use existing Working-Sets	164
10.5. Interacting with the Editor	165

III. Administration Guide	167
11. Installation	169
11.1. Installation step by step	169
11.2. Supported and recommended platforms	169
12. Database configuration	171
12.1. Changing the location of the data store	171
12.2. Configuring Guvnor to use an external RDBMS	172
12.3. Searching and indexing, Version storage	174
13. Switch from JackRabbit to ModeShape	175
14. Security - Authentication and basic access	179
14.1. Using your containers security and LDAP	180
15. Fine grained permissions and security	183
15.1. Enabling fine grained authorization	185
16. Data management	187
16.1. Backups	187
16.2. Repository Data Migration	187
16.3. Adding your own logos or styles to Guvnor web GUI	188
16.4. Import and Export	189
17. Architecture	191
17.1. Building from source	192
17.1.1. Modules	192
17.1.2. Working with Maven 2	192
17.1.3. Working with GWT	192
17.1.4. Debugging, Editing and running with Eclipse	192
17.2. Re-usable components	193
17.3. Versioning and Storage	193
17.4. Contributing	194

Chapter 1. Introduction

This section introduces the Guvnor. See the other relevant sections for installation, usage and administration.

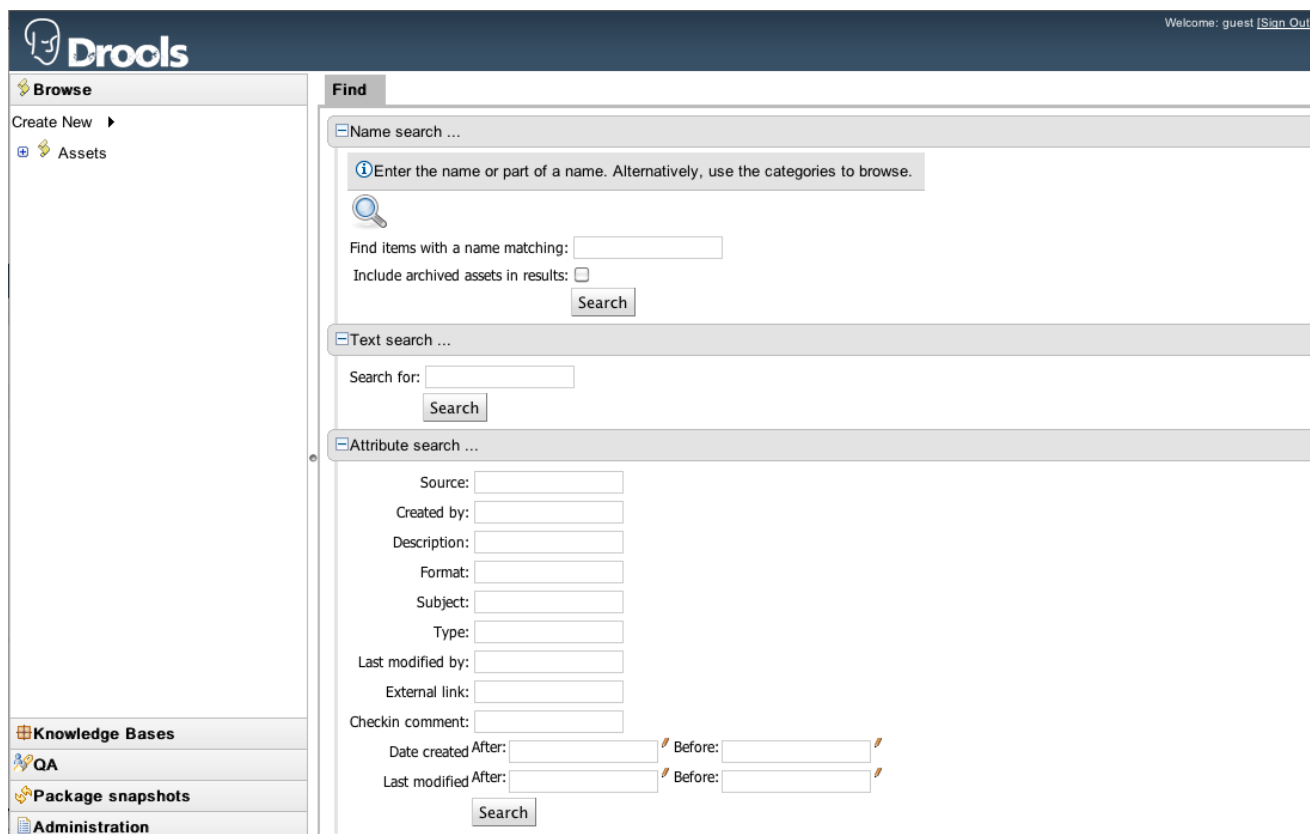


Figure 1.1. The Guvnor main screen

1.1. What is a Business Rules Manager?

A business rules manager allows people to manage rules in a multi user environment, it is a single point of truth for your business rules, allowing change in a controlled fashion, with user friendly interfaces.

Guvnor is the name of the web and network related components for managing rules with drools. This combined with the core drools engine and other tools forms the business rules manager.

1.1.1. When to use Guvnor

You should consider Guvnor if any of the following apply to you: You need to manage versions/ deployment of rules, you need to let multiple users of different skill levels access and edit rules, you don't have any existing infrastructure to manage rules, you have lots of "business" rules (as opposed to technical rules as part of an application).

Guvnor can be used on its own, or with the IDE tooling (often both together).

Guvnor can be "branded" and made part of your application, or it can be a central rule repository.

1.1.1.1. When to not use Guvnor

In some situations applications may exist which have the rules in a database (for instance as part of an existing application), and no new application is needed to manage the rules.

In this case, the drools-template library is worth a look - you can define templates for rules to be generated from any tabular data source.

Otherwise, perhaps an existing rule management system and user interface already exists (and is tailored to your environment already) - in this case migrating to Guvnor may not be necessary.

If you are using rules to solve complex algorithmic problems, and the rules are essentially an integral part of the application (and don't need to be managed separately to the code).

1.1.2. Who uses Guvnor

The main roles of people who would use Guvnor are: Business Analyst, Rule expert, Developer, Administrators (rule administrators etc).

Guvnor is designed in such a way as these different roles can be accommodated, it can be controlled how much is exposed to different users in a safe fashion.

1.2. Features outline

- Multiple types of rule editors (GUI, text)
- Version control (historical assets)
- Categorization
- Build and deploy
- Store multiple rule "assets" together as a package

Part I. User Guide

This part covers Guvnor for end-users.

Chapter 2. Quick Start Guide

2.1. Quick start guide

If you are reading this, you must be the impatient type who wants to kick the tires (and light the fires) and have a look around as soon as possible. This section will provide a quick end to end tour of the steps involved (but does not go through the concepts in detail). This assumes you have installed the repository correctly, and are able to access the main login screen.

You can also consult the [wiki](http://wiki.jboss.org/wiki/Wiki.jsp?page=RulesRepository) [http://wiki.jboss.org/wiki/Wiki.jsp?page=RulesRepository] for some tutorials and user tips (it IS a wiki, so you can even contribute your own tips and examples and even upload files if you desire !).

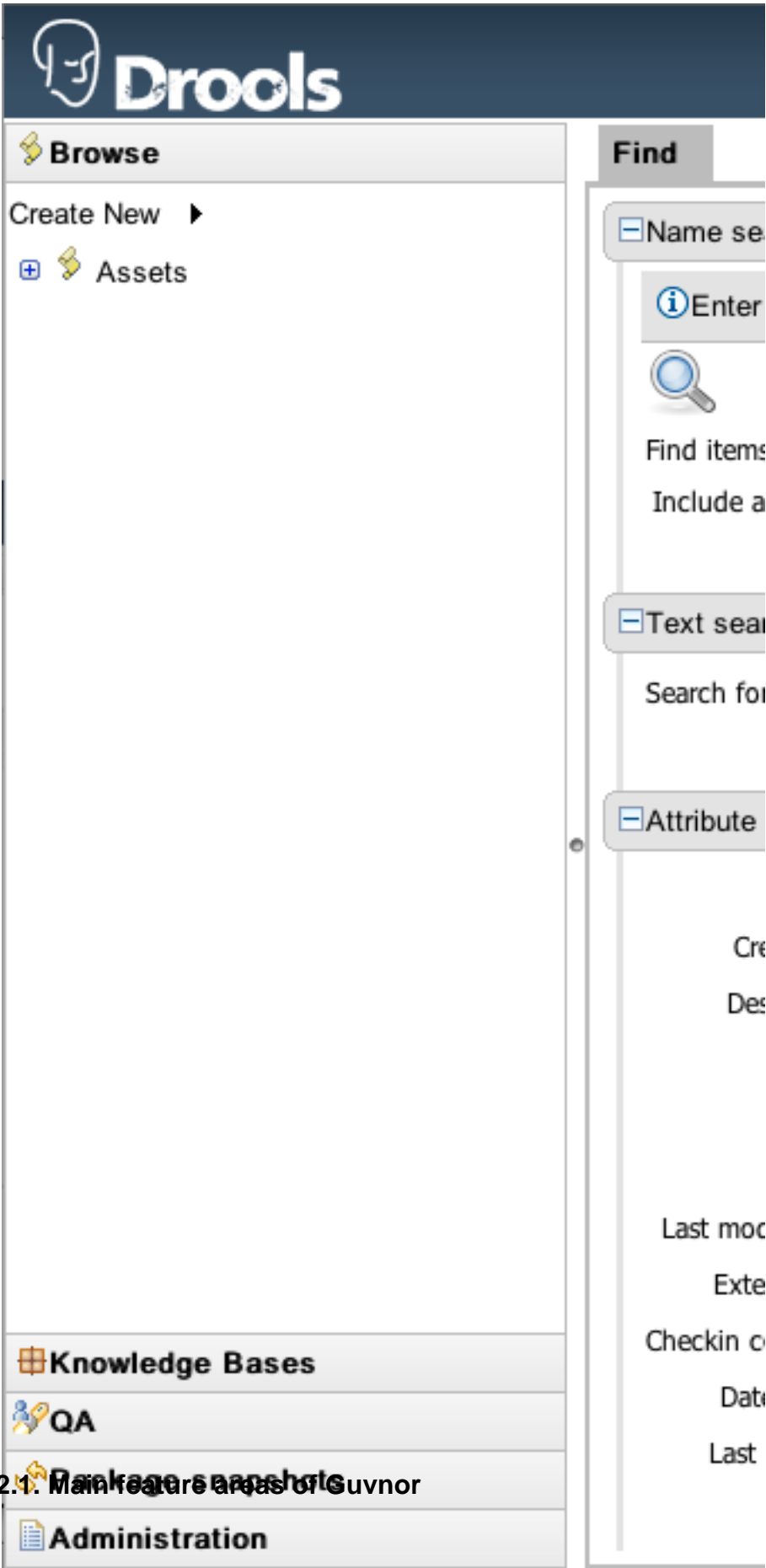


Figure 2.4. Main feature areas of Suvnor

The above picture shows the main feature areas of Guvnor.

- Info: This is the initial screen, with links to resources.
- Rules: This is the category and business user perspective.
- Package: This is where packages are configured and managed.
- Deployment: this is where deployment snapshots are managed.
- Admin: Administrative functions (categories, statuses, import and export)

2.1.1. Supported browser platforms

The supported server side platforms are mentioned in the installation guide. For browsers - the major ones are supported, this includes Firefox (1.5 and up), IE7 and up, Opera, Safari, Google Chrome etc. The preferred browser for most platforms is firefox, it is widely available and free, if you have any choice at all, Firefox is the preferred platform, followed by safari on mac. IE6 users can experience some poor performance, and as this is a dangerously insecure browser IE7 or a superior browser (such as Google Chrome, Firefox, Safari) is recommended.

2.1.2. Initial configuration

Some initial setup is required the first time. The first time the server starts up, it will create an empty repository, then take the following steps:

- Once deployed, go to <http://localhost:8080/guvnor-webapp/> This will show the initial info screen or login screen depending on the configuration.
- If it is a brand new repository, you will want to go to "Admin", and choose "Manage Categories"

(Add a few categories of your choosing, categories are only for classification, not for execution or anything else.)
- Rules need a fact model (object model) to work off, so next you will want to go to the Package management feature. From here you can click on the icon to create a new package (give it a meaningful name, with no spaces).
- To upload a model, use a JAR which has the fact model (API) that you will be using in your rules and your code (go and make one now if you need to !). When you are in the model editor screen, you can upload a JAR file, choose the package name from the list that you created in the previous step.
- Now edit your package configuration (you just created) to import the fact types you just uploaded (add import statements), and save the changes.
- At this point, the package is configured and ready to go (you generally won't have to go through that step very often).

(Note that you can also import an existing DRL package - it will store the rules in the repository as individual assets).

2.1.3. Writing some rules

- Once you have at least one category and one package setup, you can author rules.
- There are multiple rule "formats", but from the Guvnor point of view, they are all "assets".
- You create a rule by clicking the icon with the rules logo (the head), and from that you enter a name.
- You will also have to choose one category. Categories provide a way of viewing rules that is separate to packages (and you can make rules appear in multiple packages) - think of it like tagging.
- Chose the "Business rule (guided editor)" formats.
- This will open a rule modeler, which is a guided editor. You can add and edit conditions and actions based on the model that is in use in the current package. Also, any DSL sentence templates setup for the package will be available.
- When you are done with rule editing, you can check in the changes (save), or you can validate or "view source" (for the effective source).
- You can also add/remove categories from the rule editor, and other attributes such as documentation (if you aren't sure what to do, write a document in natural language describing the rule, and check it in, that can also serve as a template later)

2.1.4. Finding stuff

In terms of navigating, you can either use the Rules feature, which shows things grouped by categories, or you can use the Package feature, and view by package (and rule type). If you know the name or part of the name of an asset, you can also use the "Quick find", start typing a rule name and it will return a list of matches as you type (so if you have a sensible naming scheme, it will make it very quick to find stuff).

2.1.5. Deployment

- After you have edited some rules in a package, you can click on the package feature, open the package that you wish, and build the whole package.
- If that succeeds, then you will be able to download a binary package file which can be deployed into a runtime system.
- You can also take a "snapshot" of a package for deployment. This freezes the package at that point in time, so any concurrent changes to not effect the package. It also makes the package

available on a URL of the form: "http://<your server>/guvnor-webapp/org.drools.guvnor.Guvnor/packages/<packageName>/<snapshotName>" (where you can use that URL and downloads will be covered in the section on deployment).

Chapter 3. Concepts

3.1. Rules are assets

As the Guvnor can manage many different types of rules (and more), they are all classed as "assets". An asset is anything that can be stored as a version in the repository. This includes decision tables, models, DSLs and more. Sometimes the word "rule" will be used to really mean "asset" (i.e. the things you can do also apply to the other asset types). You can think of asset as a lot like a file in a folder. Assets are grouped together for viewing, or to make a package for deployment etc.

3.2. Packages are assets

A package is a logical container for assets.

Packages form a deployable unit containing assets and once a package has been built it itself becomes an asset.

The Global Area is a special type of package that contains assets that can be shared in other packages. Shared assets are linked to those in the Global Area.

3.3. Perspectives

TODO

3.4. The business user perspective

You can see from this manual, that some expertise and practice is required to use Guvnor. In fact any software system in some sense requires that people be "technical" even if it has a nice looking GUI. Having said that, in the right hands Guvnor can be setup to provide a suitable environment for non technical users.

The most appropriate rule formats for this use are using the Guided editor, Decision tables and DSL rules. You can use some DSL expressions also in the guided editor (so it provides "forms" for people to enter values).

You can use categories to isolate rules and assets from non technical users. Only assets which have a category assigned will appear in the "categories" view.

The initial setup of Guvnor will need to be done by a developer/technical person who will set the foundations for all the rules. They may also create "templates" which are rules which may be copied (they would typically live in a "dummy" package, and have a category of "template" - this can also help ease the way).

Deployment should also not be done by non technical users (as mentioned previously this happens from the "Package" feature).

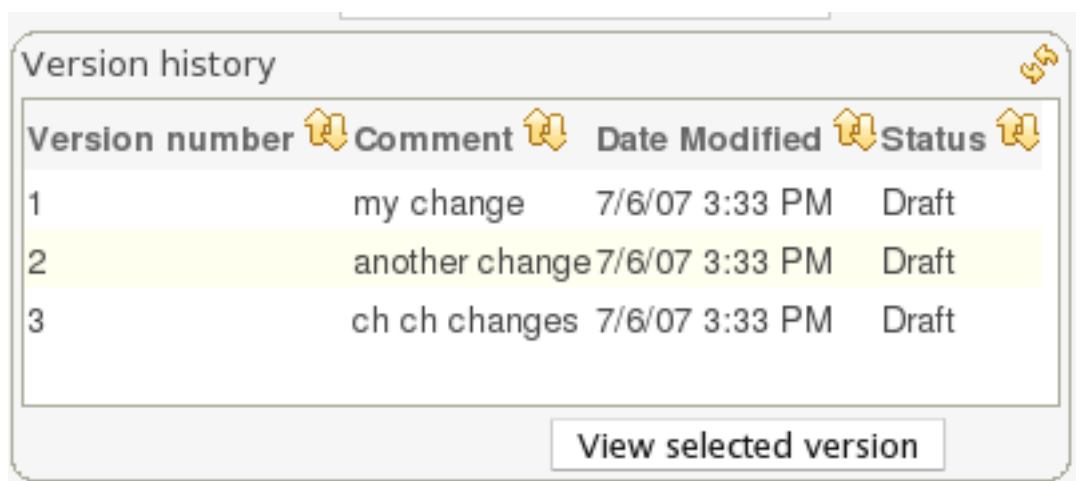
3.4.1. Creating a business user view

In most cases not all users will want to see all the functionality described here. You could have a subset of users who you only want to let view or edit certain sets of rules, without getting confused by all the other stuff. In this case you can use fine grained authorization (see the Admin Guide on how to initialize this). By setting permissions on a per category basis, users that only have category permissions will see a limited subset of functionality, and only items that are tagged with those categories.

Chapter 4. Authoring Assets

4.1. Version management

Both assets and whole packages of assets are "versioned" in the Guvnor, but the mechanism is slightly different. Individual assets are saved a bit like a version of a file in a source control system. However, packages of assets are versioned "on demand" by taking a snapshot (typically which is used for deployment). The next section talks about deployment management and snapshots.



Version number	Comment	Date Modified	Status
1	my change	7/6/07 3:33 PM	Draft
2	another change	7/6/07 3:33 PM	Draft
3	ch ch changes	7/6/07 3:33 PM	Draft

View selected version

Figure 4.1. Asset versions

Each time you make a change to an asset, it creates a new item in the version history. This is a bit like having an unlimited undo. You can look back through the history of an individual asset like the list above, and view it (and restore it) from that point in time.

4.2. The Asset Editor

The Asset Editor is the principle component of Guvnor's User-Interface. It consists of two tabs:-

- Attributes
 - A : Meta data (from the "Dublin Core" standard):-
 - "Last modified:" The last modified date.
 - "By:" Who made the last change.
 - "Note:" A comment made when the Asset was last updated (i.e. why a change was made)
 - "Created on:" The date and time the Asset was created.
 - "Created by:" Who initially authored the Asset.
 - "Format:" The short format name of the type of Asset.

"Package:" The package to which the Asset belongs.

"Is Disabled:" Whether the Asset has been disabled from inclusion in a binary package.

"UUID:" A unique identifier for the Asset version.

- B : Other miscellaneous meta data for the Asset.
- C : Version history of the Asset.
- D : Free-format documentation\description for the Asset. It is encouraged, but not mandatory, to record a description of the Asset before editing.
- E : Discussions regarding development of the Asset can be recored here.

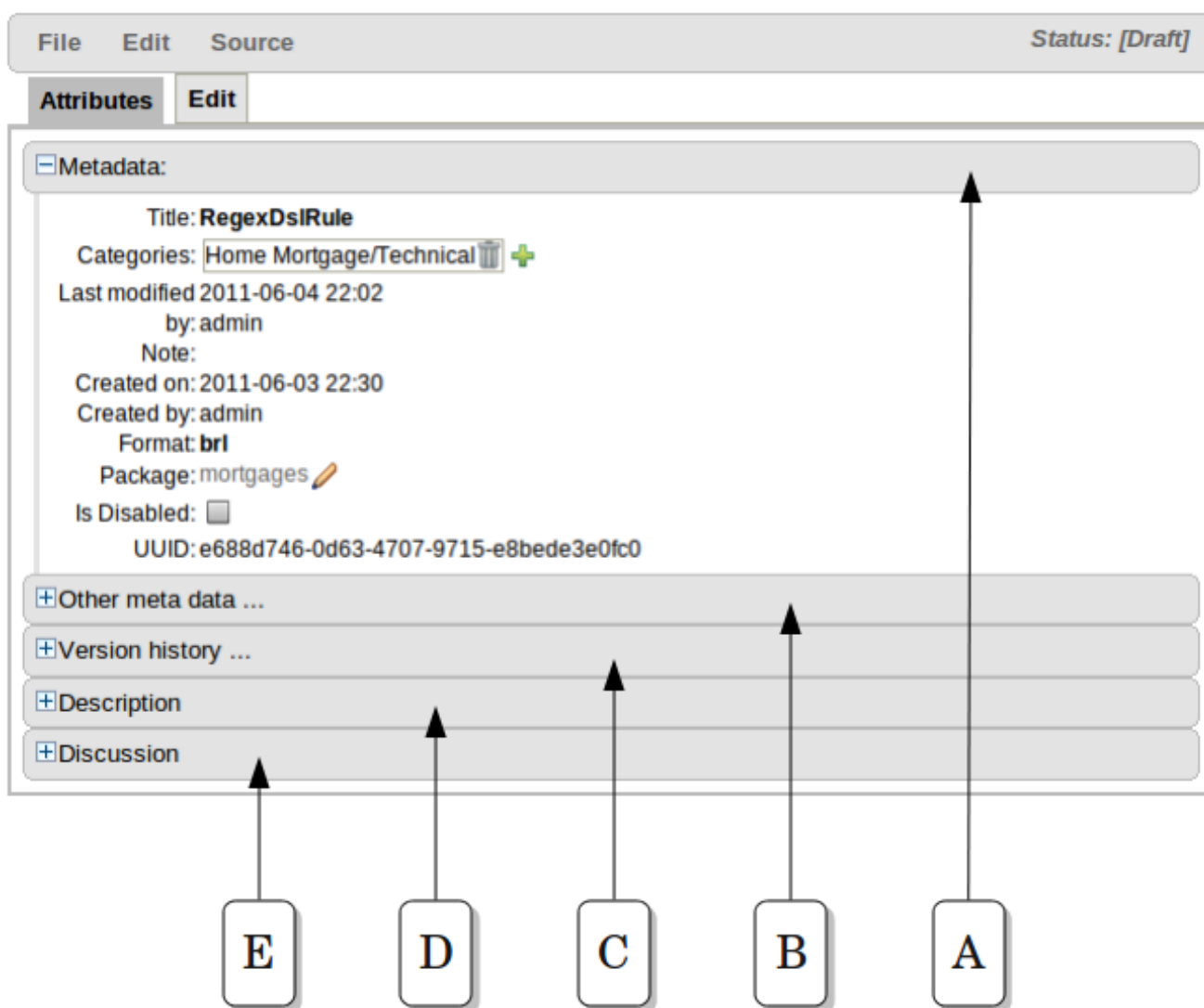


Figure 4.2. The Asset Editor - Attributes tab



[-] Other meta data ...

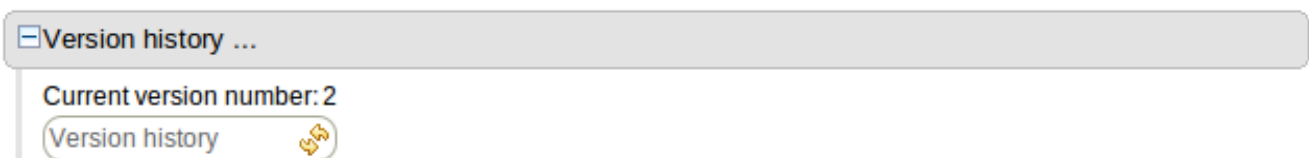
Subject:

Type:

External link:

Source:

Figure 4.3. The Asset Editor - Other meta data



[-] Version history ...

Current version number: 2


Version history 

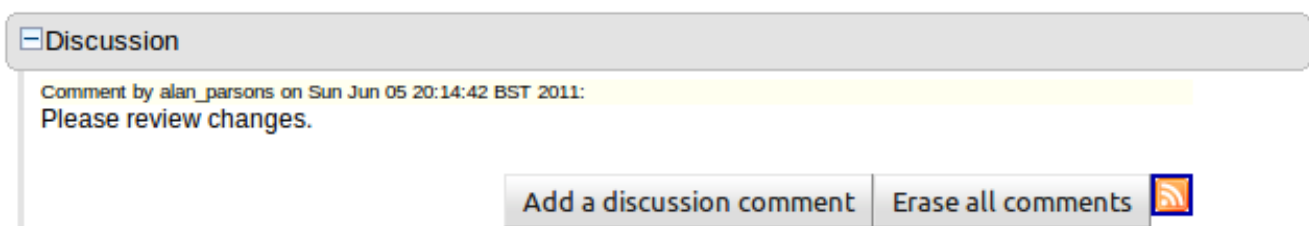
Figure 4.4. The Asset Editor - Version history



[-] Description

<documentation>

Figure 4.5. The Asset Editor - Description



[-] Discussion

Comment by alan_parsons on Sun Jun 05 20:14:42 BST 2011:
Please review changes.


Add a discussion comment Erase all comments 

Figure 4.6. The Asset Editor - Discussion

- Edit
 - A : The Asset editor is where the "editor widget" lives - exactly what form the editor takes depends on the Asset type.
 - B : These are menus contains various actions for the Asset; such as Saving, Archiving, changing Status etc.

- C : The current status of the Asset.

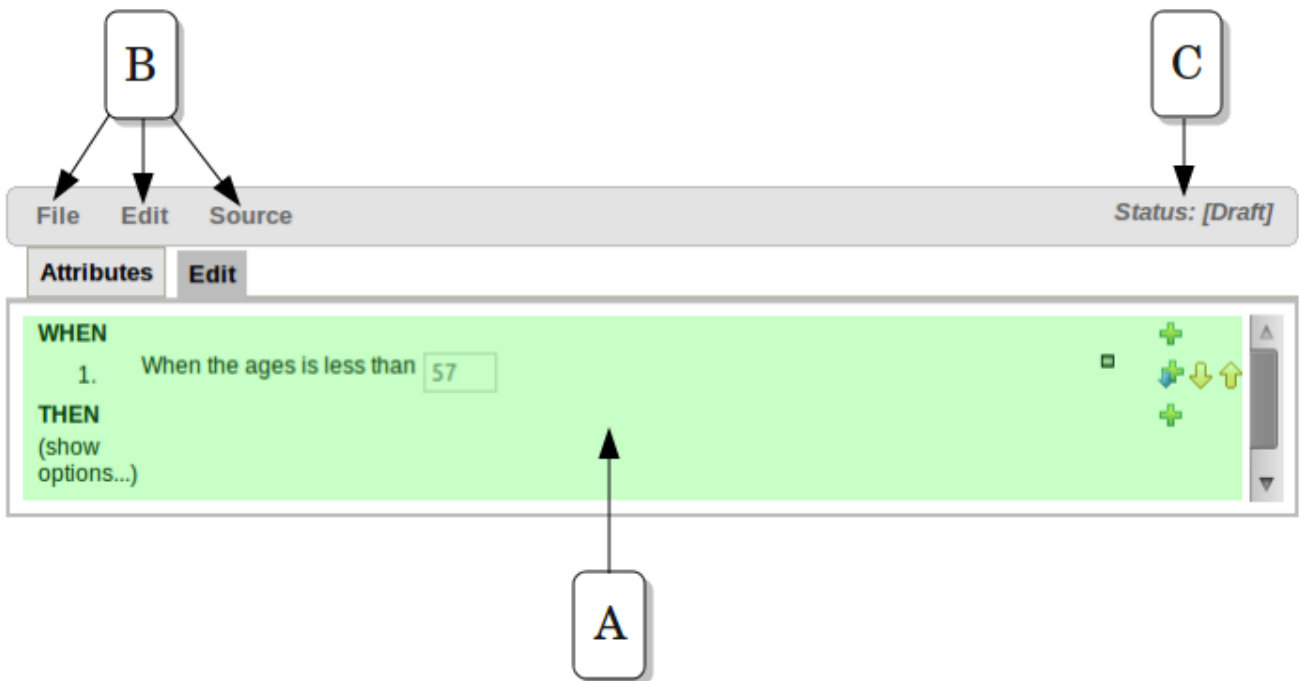


Figure 4.7. The Asset Editor - Edit tab

4.3. Creating a package

Configuring packages is generally something that is done once, and by someone with some experience with rules/models. Generally speaking, very few people will need to configure packages, and once they are setup, they can be copied over and over if needed. Package configuration is most definitely a technical task that requires the appropriate expertise.

All assets live in "packages" in the Guvnor - a package is like a folder (it also serves as a "namespace"). A home folder for rule assets to live in. Rules in particular need to know what the fact model is, what the namespace is etc.

So while rules (and assets in general) can appear in any number of categories, they only live in one package. If you think of the Guvnor as a file system, then each package is a folder, and the assets live in that folder - as one big happy list of files. When you create a deployment snapshot of a package, you are effectively copying all the assets in that "folder" into another special "folder".

To create an empty package select "New Package" from the "Create new" (asset) menu.

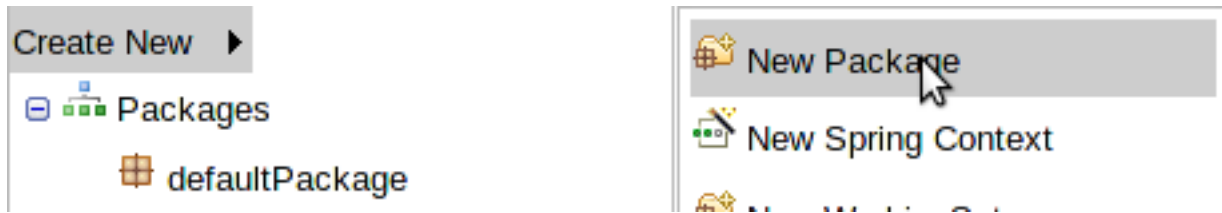


Figure 4.8. New Package

4.3.1. Empty package

An empty package can be created by simply specifying a name and optional description.

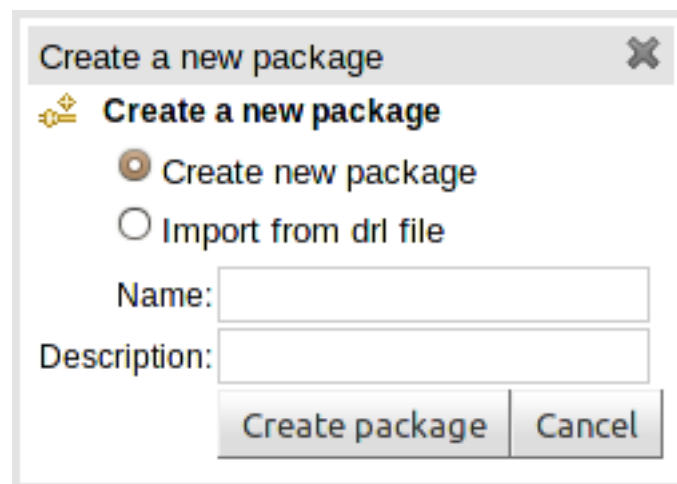


Figure 4.9. New empty Package

4.3.2. Importing DRL packages

It is also possible to create a package by importing an existing DRL file. When you choose to create a new package, you can choose an option to upload a `.drl` file. The Guvnor will then attempt to understand that DRL, break create a package for you. The rules in it will be stored as individual assets (but still as DRL text content). Note that to actually build the package, you will need to upload an appropriate model (as a JAR) to validate against, as a separate step.

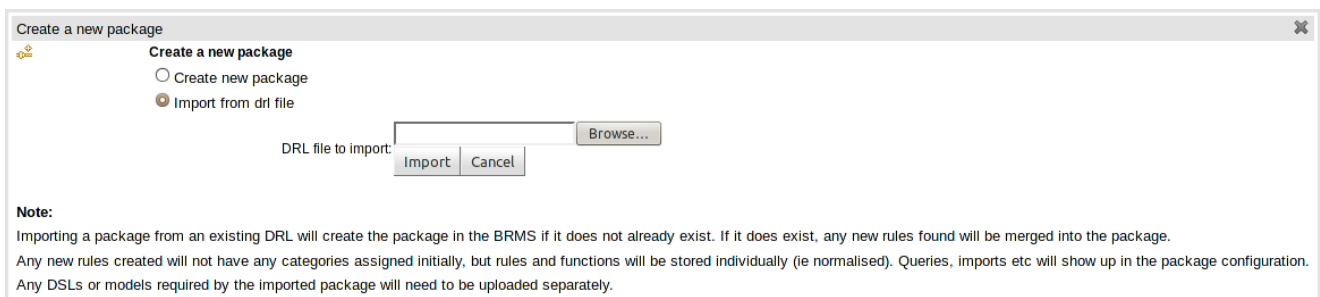


Figure 4.10. Importing DRL packages

4.4. Spring Contexts

This textual editor allows you to define Drools (and potentially any) Spring context file. These files are later accessible through HTTP.



Figure 4.11. Spring Context - Editor

The editor comes with a basic palette that you can use to paste predefined Spring Beans templates like kbases, ksessions and so on.

The palette also has a Package tree that can be used to add resources to the Spring Context file being edited.

The Beans are inserted in the caret position of the editor

The elements in the palette can be customized editing the file `$GUVNOR_HOME/WEB-INF/Classes/springContextElements.properties`

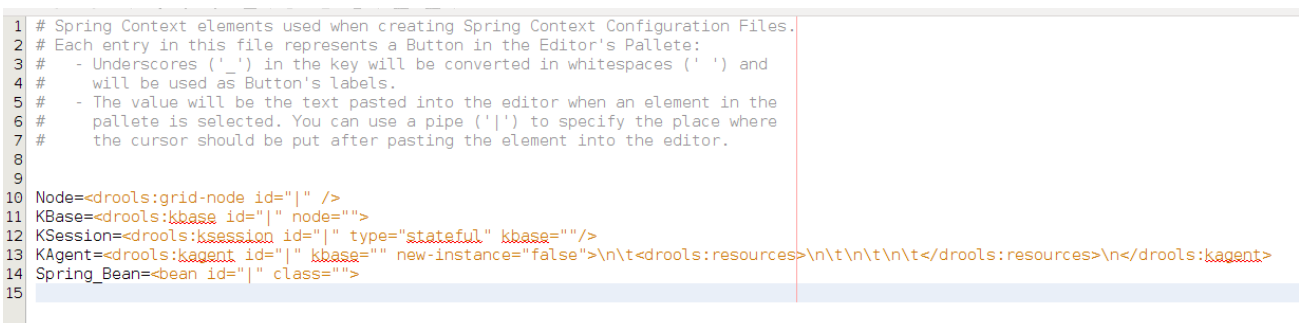


Figure 4.12. Spring Context - Palette Configuration

Each Spring Context has its own URL that applications can use to access it. These URLs are shown in the Package Edit Screen

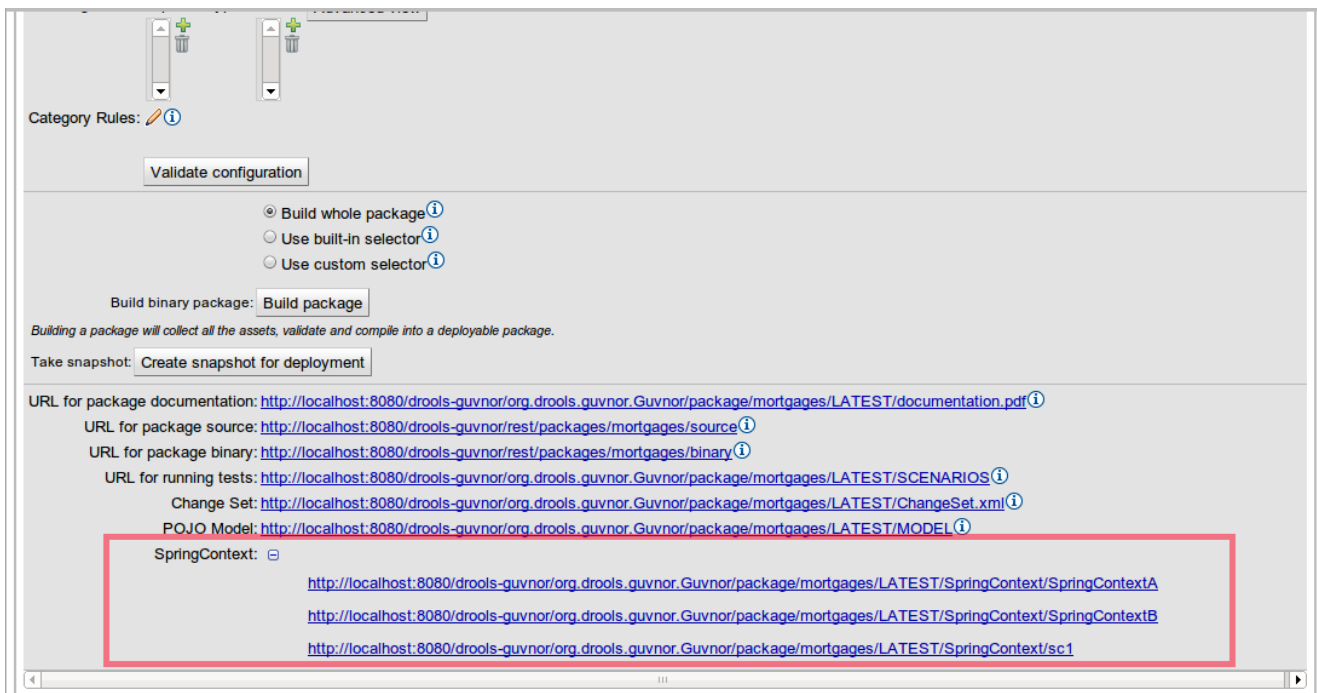


Figure 4.13. Spring Context - Public URLs

4.5. Working Sets

Working Sets are a mean for grouping Facts and then defining constraints on them. You can create groups of Facts and only those Facts will be visible when authoring rules using the Guided Editor.

Right now, Working Sets must be activated manually from the Guided Editor window (using the "Select Working Set" button placed in the toolbar). In the future, different Working Sets could be assigned to different users to reduce the scope and complexity when authoring rules.

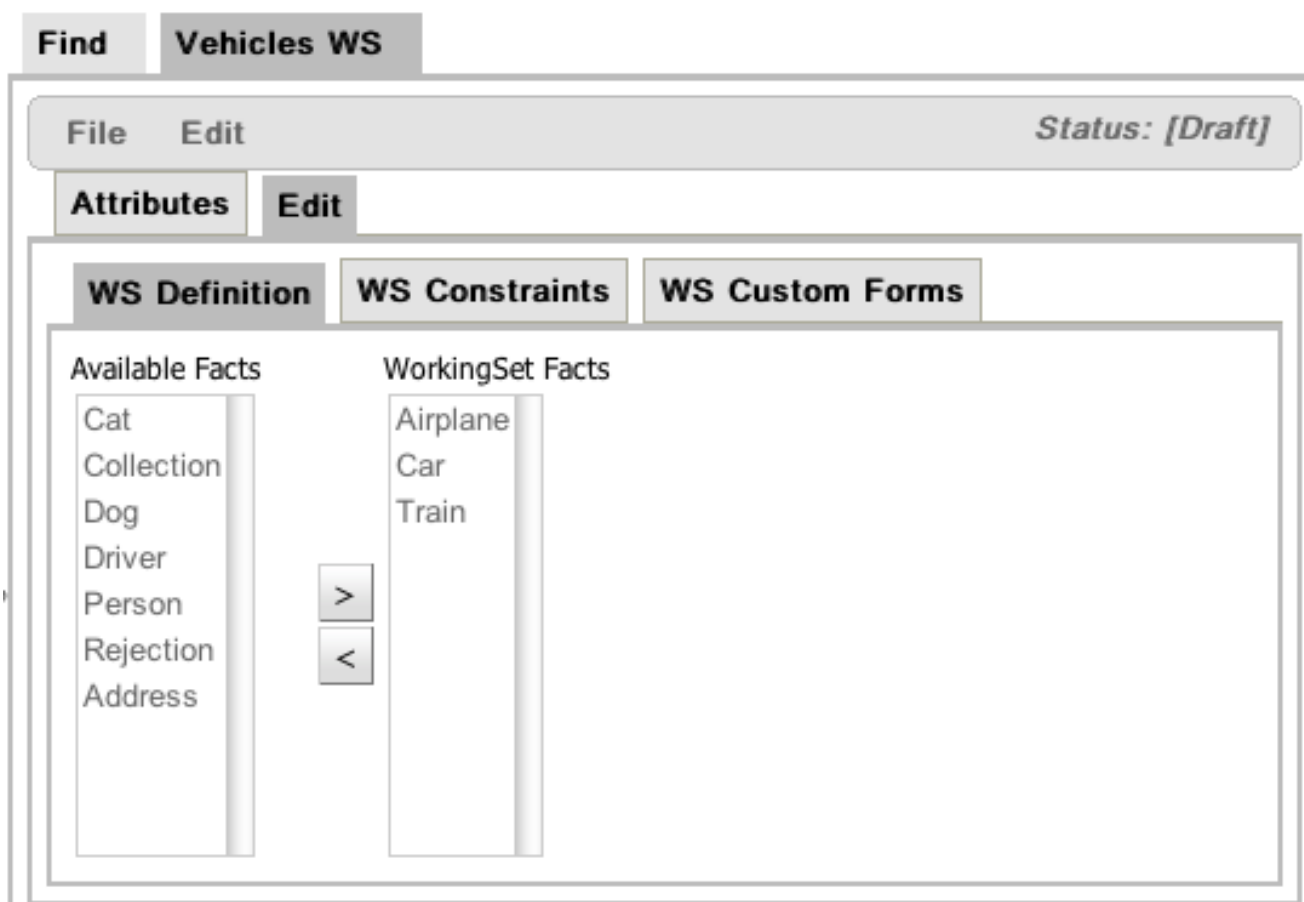


Figure 4.14. Creating a new Working Set

The figure above shows the window used to create or modify Working Sets. In this window you will find 2 lists. The list on the left side contains the possible Fact Types that can be added to the Working Set. These facts are those defined/imported in the package's model. The list on the right side contains the allowed Fact Types of this Working Set. When this Working Sets is active, only those Fact Types could be used while authoring rules using the Guided BRL Editor

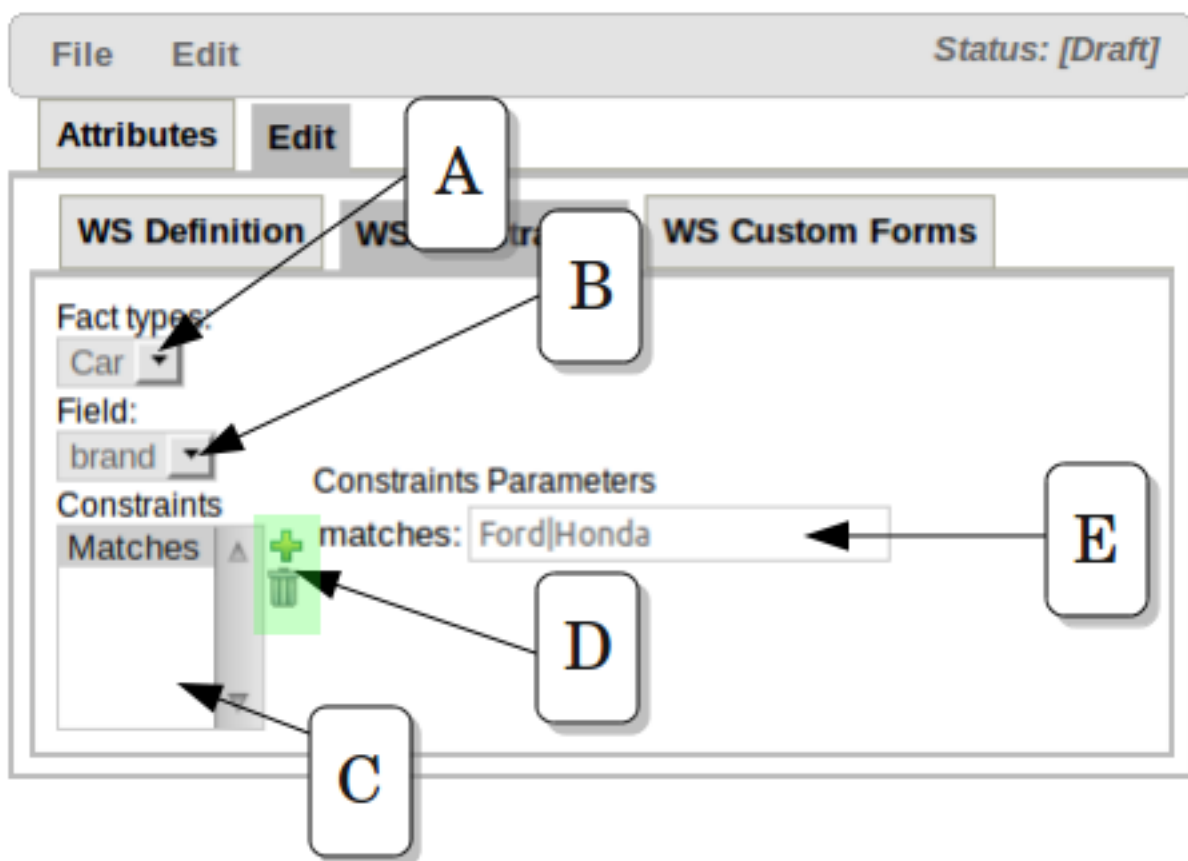


Figure 4.15. Defining Field Constraints inside a Working Set

Once you have selected the valid Fact Types for a Working Set, you can add Constraints to the fields of those Facts Types. The image above shows how the Field Constraint tab looks like. In this configuration screen you will find:

A.- Fact Types dropdown: Here you will find a list containing the Working Set's Fact Types

B.- Field dropdown: Once you have selected a Fact Type, this dropdown will contain its fields.

C.- Constraints List: This lists shows all the Constraints applied to the selected Field

D.- Action Buttons: Using these buttons you will be able to add or remove Constraints to the selected Field. Right now, Guvnor provides a built-in collection of Constraints. The idea for next releases is to let users to plug their custom Constraints too.

E.- Constraint's Attributes: In this section you will find all the attributes of the current Constraint that could be parametrized by the user.

In the example above, a Matches Constraint is created for Car.brand field. This means that when rule authors use this field in a Rule condition, they should use a value valid according to this constraint, otherwise they will receive an error or warning.

4.5.1. Activating and Using Working Sets

Working Sets are not active by default in Guvnor. Because this is an experimental feature, you must enable them manually in the Guided Editor panel if you want to use them. In the future, Working Sets will be associated to each user's profile.

A new button was added in Guided Editor's Toolbar: "Select Working Sets". This button will open a popup with the list of the package's Working Sets. Using this popup you can activate one or more Working Sets.

When Working Sets are activated, only the Fact Types allowed by them could be used when inserting new Patterns or Actions. The Patterns and Actions already present in the rule that contain prohibited Fact Types are marked as read only. Take a look at the next screen shots comparing the Guided Editor panel with and without Working Sets

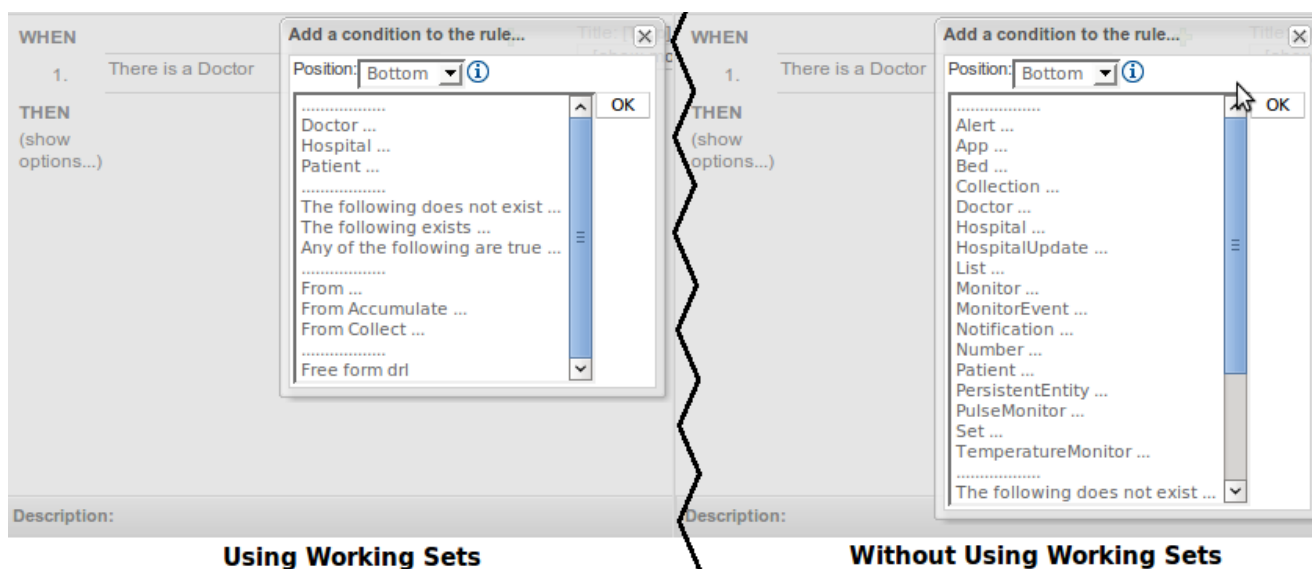


Figure 4.16. Comparison of "Add new Pattern" window using Working Set and without using them

In the image you can see how Working Sets could help rule's authors by reducing the amount of available Fact Types

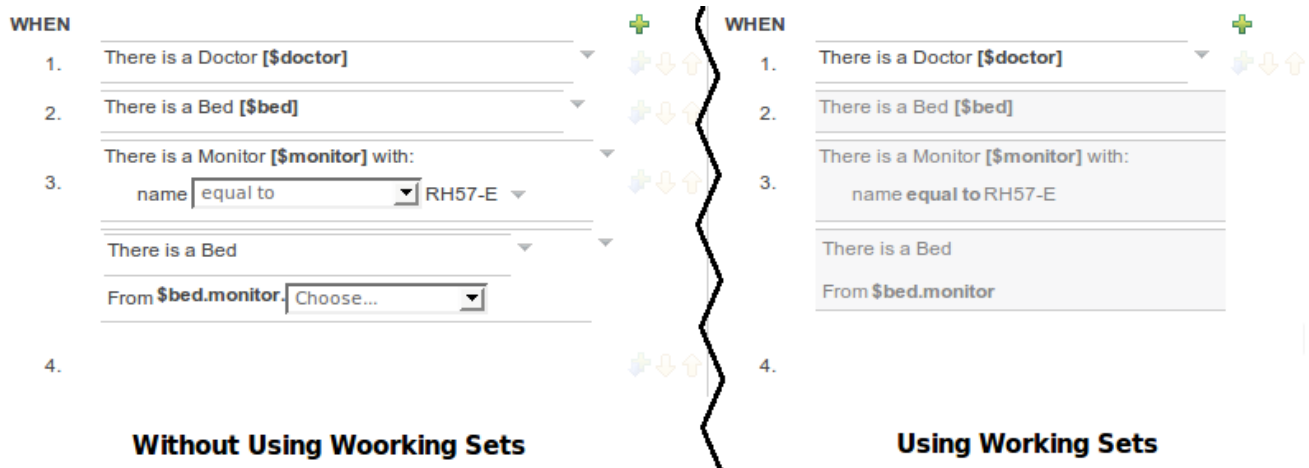


Figure 4.17. Comparison of "Add new Pattern" window using Working Set and without using them

Here you can see how Patterns containing prohibited Fact Types are switched to read only mode after Working Sets are activated.

4.5.1.1. Using Field Constraint

Up to now we have only cover how Facts are filtered using Working Sets. Another important feature of Working Sets is Field Constraints. We have already saw how to configure them, now we are going to explain how to use them.

Because Field Constraints are defined inside a Working Set, we need to activate one or more Working Set to start working with them. Once a Working Set defining Field's Constraints is active we have two ways to use them: on demand validation and real-time validation.

On demand validation is performed when you press the "Verify" button present in Guided Editor's toolbar. This button will fire a rule verification and will end up showing a report with the results. Any violated constraint will be shown as an error or warning according to its relevance



Figure 4.18. On demand Field Constraints validation

The image above shows the report that appears when a Working Set defines a Range Constraint on Driver.age. The age should be between 18 and 80.

Real-Time validation is an experimental feature (yes, inside another experimental feature like Working Sets) that checks for Field's Constraints violations in real time and mark the lines where the violations are using error or warning icons. This feature is disabled by default because sometimes it could be expensive. If you want to try it out, you should enable it in Administration -> Rules Verification. This configuration is not yet persisted, so you need to enable it every time you start Guvnor.



Figure 4.19. Real-Time Field Constraints validation

This Image shows the result of real-time validation. There you can see the same result as on demand validation, but you don't need to click any button, and the errors/warnings are shown in a more fashionable way!



Warning

The problem with real-time validation is that right now only support "top level" Patterns.

4.6. Business rules with the guided editor

Guided editor style "Business rules": (also known as "BRL format"). These rules use the guided GUI which controls and prompts user input based on knowledge of the object model. This can also be augmented with DSL sentences.



Note

To use the BRL guided editor, someone will need to have you package configured before hand.

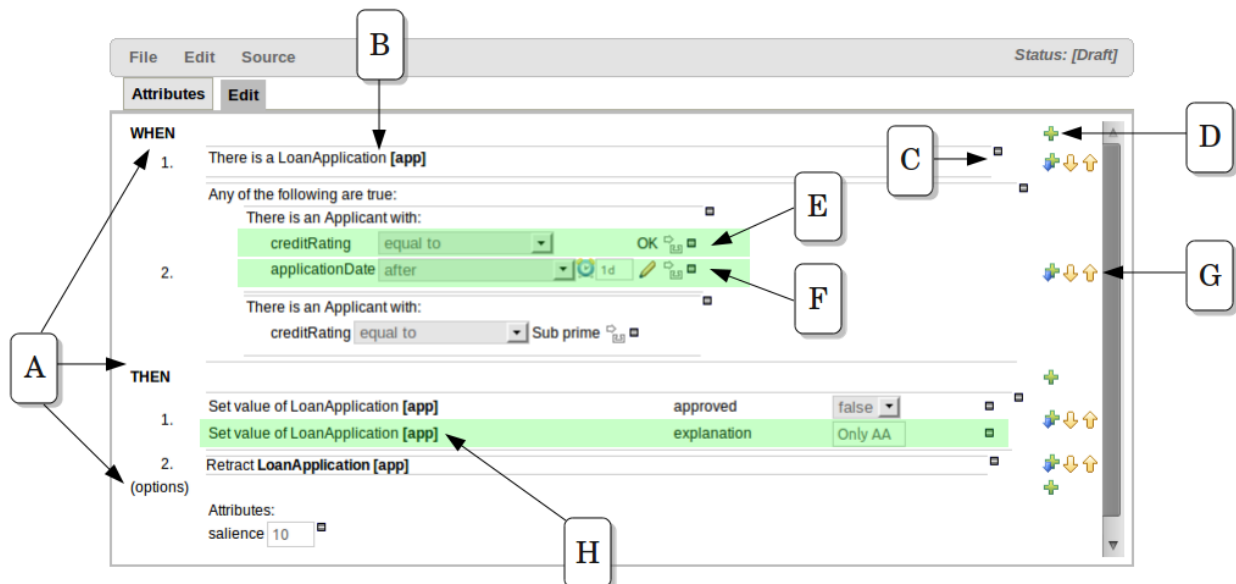


Figure 4.20. The guided BRL editor

The above diagram shows the editor in action. The following descriptions apply to the lettered boxes in the diagram:-

A : The different parts of a rule:-

- The "WHEN" part, or conditions, of the rule.
- The "THEN" action part of the rule.
- Optional attributes that may effect the operation of the rule.

B : This shows a pattern which is declaring that the rule is looking for a "LoanApplication" fact (the fields are listed below, in this case none). Another pattern, "Applicant", is listed below "LoanApplication". Fields "creditRating" and "applicationDate" are listed. Clicking on the fact name ("LoanApplication") will pop-up a list of options to add to the fact declaration:-

- Add more fields (e.g. their "location").
- Assign a variable name to the fact (which you can use later on if needs be)

- Add "multiple field" constraints - i.e. constraints that span across fields (e.g. age > 42 or risk > 2).

C : The "minus" icon ("[-]") indicates you can remove something. In this case it would remove the whole "LoanApplication" fact declaration. Depending upon the placement of the icon different components of the rule declaration can be removed, for example a Fact Pattern, Field Constraint, other Conditional Element ("exists", "not exists", "from" etc) or an Action.

D : The "plus" icon ("+") allows you to add more patterns to the condition or the action part of the rule, or more attributes. In all cases, a popup option box is provided. For the "WHEN" part of the rule, you can choose from a list of Conditional Elements to add:

- A Constraint on a Fact: it will give you a list of facts.
- "The following does not exist": the fact plus constraints must not exist.
- "The following exists": at least one match should exist (but there only needs to be one - it will not trigger for each match).
- "Any of the following are true": any of the patterns can match (you then add patterns to these higher level patterns).
- "From": this will insert a new From Conditional Element to the rule.
- "From Accumulate": this will insert a new Accumulate Conditional Element to the rule.
- "From Collect": this will insert a new Collect Conditional Element to the rule.
- "From Entry-point": this allows you to define an Entry Point for the pattern.
- "Free Form DRL": this will let you insert a free chunk of DRL.

If you just put a fact (like is shown above) then all the patterns are combined together so they are all true ("and").

E : This shows the constraint for the "creditRating" field. Looking from left to right you find:-

- The field name: "creditRating". Clicking on it you can assign a variable name to it, or access nested properties of it.
- A list of constraint operations ("equal to" being selected): The content of this list changes depending on the field's data type.
- The value field: It could be one of the following:-
 1. A literal value: depending on the field's data type different components will be displayed:
 - String or Number -> TextBox
 - Date -> Calendar

- Enumeration -> Combobox
- Boolean -> Checkbox

2. A "formula": this is an expression which is calculated (this is for advanced users only)

3. An Expression - this will let you use an Expression Builder to build up a full mvel expression. (At the moment only basic expressions are supported)

F : This shows the constraint for the "applicationDate" field. Looking from left to right you find:

- The field name: "applicationDate".
- A list of constraint operations: "after" being selected.
- A "clock" icon. Since the "applicationDate" is a Date data-type the list of available operators includes those relating to Complex Event Processing (CEP). When a CEP operator is used this additional icon is displayed to allow you to enter additional CEP operator parameters. Clicking the "clock" will cycle the available combinations of CEP operator parameters.



Note

Complex Event Processing operators are also available when the Fact has been declared as an event. Refer to the "Fact Model" chapter of this user-guide for details on how to add annotations to your Fact model. Events have access to the full range of CEP operators; Date field-types are restricted to "after", "before" and "coincides".



Note

Facts annotated as Events can also have CEP sliding windows defined.

G : Pattern/Action toolbar. Next to each Pattern or Action you will find a toolbar containing 3 buttons. The first button lets you insert a new Pattern/Action below the one you selected, the other two buttons will move the current Pattern/Action up or down.

H : This shows an "action" of the rule, the Right Hand Side of a rule consists in a list of actions. In this case, we are updating the "explanation" field of the "LoanApplication" fact. There are quite a few other types of actions you can use:-

- Insert a completely new Fact
- Logically insert a completely new Fact (see "Truth Maintenance" in the Expert documentation).

- Modify an existing fact (which tells the engine the fact has changed).
- Set a field on a fact (in which case the engine doesn't know about the change - normally because you are setting a result).
- Retract a fact.
- Add Facts to existing global lists.
- Call a method on a variable.
- Write a chunk of free form code.

In most cases you can click on the Fact name to get a list of its attributes or to bind it to a variable name.

4.6.1. User driven drop down lists

The screenshot shows a rule editor interface. At the top, it says "There is a LoanApplication [app]". Below that, it says "Any of the following are true:". Under this, there is a section "There is an Applicant with:". This section contains two rows of conditions. The first row is "creditRating" followed by a drop-down menu showing "equal to". The second row is "applicationDate" followed by a drop-down menu showing "after". To the right of these conditions are icons for "OK", a clock, and a pencil. A mouse cursor is clicking on the "OK" button in the drop-down menu for "applicationDate". The drop-down menu is open, showing a list of values: "AA", "OK", and "Sub prime". There are up and down arrow buttons on the right side of the list, and an "OK" button to the right of the list.

Figure 4.21. Data enumeration showing as a drop down list

Note that it is possible to limit field values to items in a pre configured list. This list is configured as part of the package (using a data enumeration to provide values for the drop down list). These values can be a fixed list, or (for example) loaded from a database. This is useful for codes, and other fields where there are set values. It is also possible to have what is displayed on screen, in a drop down, be different to the value (or code) used in a rule. See the section on data enumerations for how these are configured.

4.6.2. Augmenting with DSL sentences

If the package the rule is part of has a DSL configuration, when when you add conditions or actions, then it will provide a list of "DSL Sentences" which you can choose from - when you choose one, it will add a row to the rule - where the DSL specifies values come from a user, then a edit box (text) will be shown (so it ends up looking a bit like a form). This is optional, and there is another DSL editor. Please note that the DSL capabilities in this editor are slightly less then the full set of DSL features (basically you can do [when] and [then] sections of the DSL only - which is no different to drools 3 in effect).

The following diagram shows the DSL sentences in action in the guided editor:

WHEN	A template captures values in a form style of input
THEN	Action sentence template
(options)	

Figure 4.22. DSL in guided editor

4.6.3. A more complex example:

WHEN		+
There is a Person [\$p] with:		
1.	birthDate less than 19-Dec-1982	↓ ↑
	(*)= car.brand == "Ford" && salary > (2500 * 4.1)	
There is an Address with:		
2.	street equal to Elm St.	↓ ↑
	From \$p.addresses. Choose...	
The following does not exist:		
3.	There is a Person with:	↓ ↑
	salary equal to (*)= \$p.salary * 2	
There is a Number [\$totalAddresses]		
From Accumulate		
There is an Address [\$a] with:		
4.	zipCode equal to 43240	↓ ↑
	From \$p.addresses. Choose...	
	Custom Code Function	
	Function: count(\$a)	
THEN		+
Insert Person:		
1.	name \$p.name	↓ ↑
(show options...)		

Figure 4.23. A more complex BRL example

In the above example, you can see how to use a mixture of Conditional Elements, literal values, and formulas. The rule has 4 "top level" Patterns and 1 Action. The "top level" Patterns are:

1. A Fact Pattern on Person. This Pattern contains two field constraints: one for birthdate field and the other one is a formula. Note that the value of the birthdate restriction is selected from a calendar. Another thing to note is that you can make calculations and use nested fields in the formula restriction (i.e. car.brand). Finally, we are setting a variable name (\$p) to the Person Fact Type. You can then use this variable in other Patterns.



Note

The generated DRL from this Pattern will be:

```
$p : Person( birthDate < "19-Dec-1982" , eval( car.brand == "Ford"
    && salary > (2500 * 4.1) ) )
```

2. A From Pattern. This condition will create a match for every Address whose street name is "Elm St." from the Person's list of addresses. The left side of the from is a regular Fact Pattern and the right side is an Expression Builder that let us inspect variable's fields.



Note

The generated DRL from this Pattern will be: `Address(street == "Elm St.")`
`from $p.addresses`

3. A "Not Exist" Conditional Element. This condition will match when its content doesn't create a match. In this case, its content is a regular Fact Pattern (on Person). In this Fact Pattern you can see how variables (\$p) could be used inside a formula value.



Note

The generated DRL from this Pattern will be: `not Person(salary ==`
`($p.salary * 2))`

4. A "From Accumulate" Conditional Element. This is maybe one of the most complex Patterns you can use. It consist in a Left Pattern (It must be a Fact Pattern. In this case is a Number Pattern. The Number is named \$totalAddresses), a Source Pattern (Which could be a Fact Pattern, From, Collect or Accumulate conditional elements. In this case is an Address Pattern Restriction with a field restriction in its zip field) and a Formula Section where you can use any built-in or custom Accumulate Function (in this example a count() function is used). Basically, this Conditional Element will count the addresses having a zip code of 43240 from the Person's list of addresses.

**Note**

The generated DRL from this Pattern will be: `$totalAddresses : Number()
from accumulate ($a : Address(zipCode == " 43240") from
$p.addresses, count($a))`

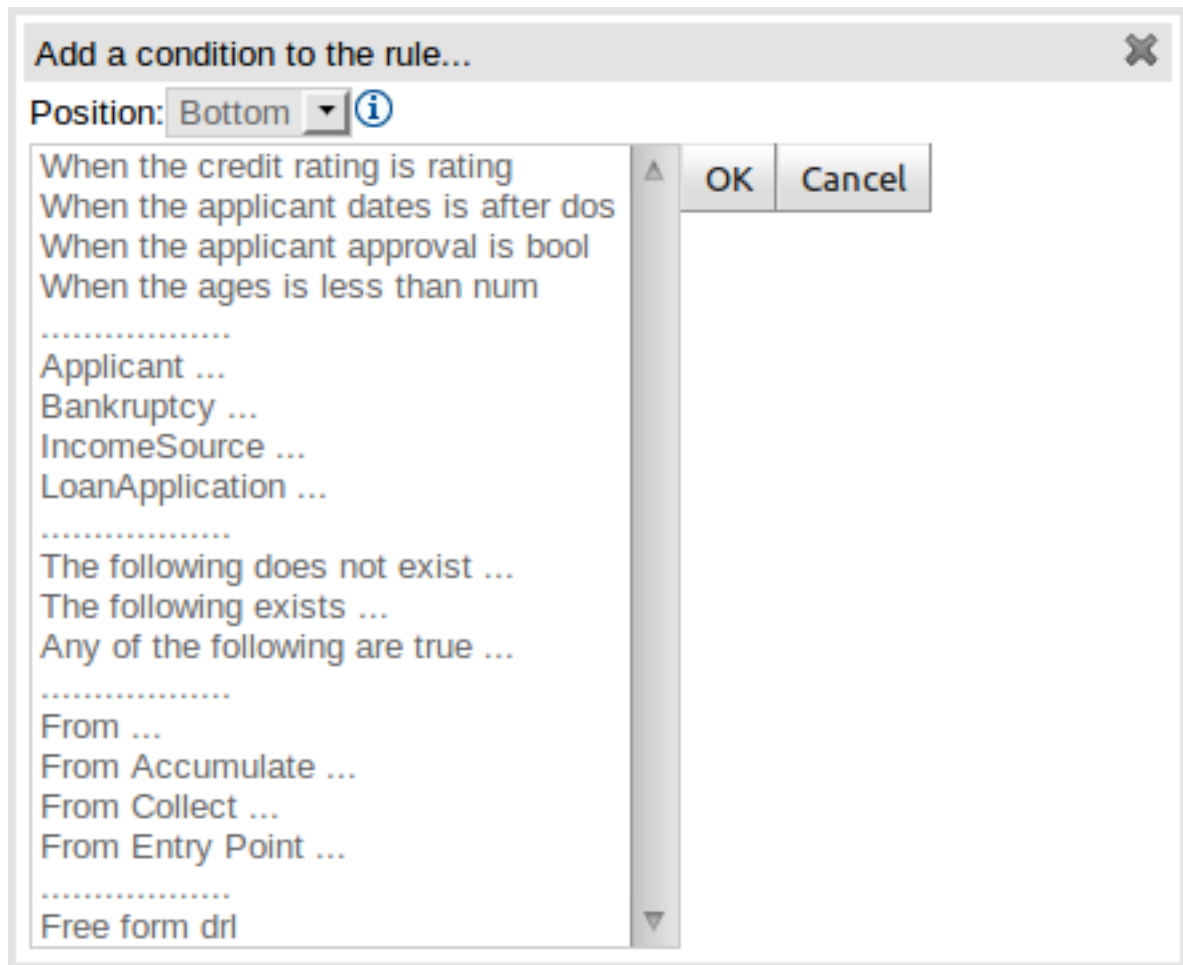


Figure 4.24. Adding Patterns

When clicking on the + button of the WHEN section, a new popup will appear letting you to add a new Pattern to the Rule. The popup will look similar to the image above. In this popup you could select the type of Pattern to add by selecting one of the list items. In the list you will have an entry for each defined Fact Type, in addition to the already mentioned Conditional Elements like "exists", "doesn't exist", "from", "collect", "accumulate", "from entry-point" and "free form DRL". Once you have selected one of these elements, you can add a new Pattern by clicking on the "Ok" button. The new pattern will be added at the bottom of the rule's left hand side. If you want to choose a different position, you can use the combobox placed at the top of the popup.

You can also open this popup by clicking in the [+] button from a Pattern's action toolbar. If that is the case, the pop-up that appears wouldn't contain the position combobox, because the new Pattern will be added just after the Pattern where you clicked.

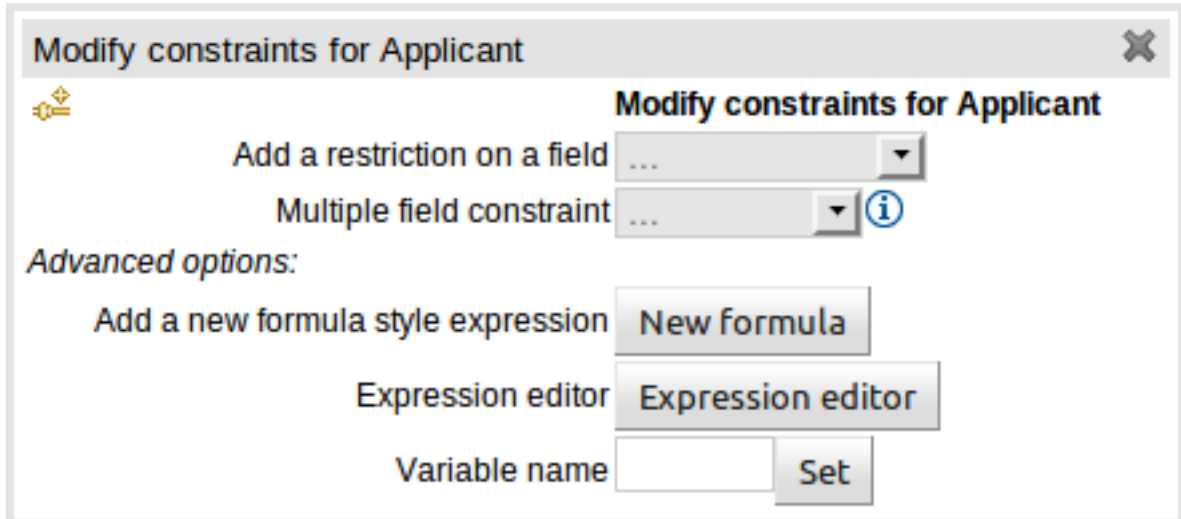


Figure 4.25. Adding constraints

The above dialog is what you will get when you want to add constraints to a fact. In the top half are the simple options: you can either add a field constraint straight away (a list of fields of the applicable fact will be shown), or you can add a "Multiple field constraint" using AND or OR operands. In the bottom half of the window you have the Advanced options: you can add a formula (which resolves to True or False - this is like in the example above: "... salary > (2500 * 4.1)". You can also assign a Variable name to the fact (which means you can then access that variable on the action part of the rule, to set a value etc).

4.7. DSL rules

DSL rules are textual rules, that use a language configuration asset to control how they appear.



Figure 4.26. DSL rule

A dsl rule is a single rule. Referring to the picture above, you can a text editor. You can use the icons to the right to provide lists of conditions and actions to choose from (or else press Control + Space at the same time to pop up a list).

4.8. Technical rules (DRL)

Technical (DRL) rules are stored as text - they can be managed in the Guvnor. A DRL can either be a whole chunk of rules, or an individual rule. if its an individual rule, no package statement or imports are required (in fact, you can skip the "rule" statement altogether, just use "when" and "then" to mark the condition and action sections respectively). Normally you would use the IDE to edit raw DRL files, since it has all the advanced tooling and content assistance and debugging. However, there are times when a rule may have to deal with something fairly technical in a package in Guvnor. In any typical package of rules, you generally have a need for some "technical rules" - you can mix and match all the rule types together of course.

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

Figure 4.27. DRL technical rule

4.9. Spreadsheet decision tables

Multiple rules can be stored in a spreadsheet (each row is a rule). The details of the spreadsheet are not covered in this chapter (as there is a separate chapter for them).



Figure 4.28. Spreadsheet decision table

To use a spreadsheet, you upload an XLS file (and can download the current version, as per the picture above). To create a new decision table, when you launch the rule wizard, you will get an option to create one (after that point, you can upload the XLS file).

4.10. Guided decision tables (web based)

The guided decision table feature allows decision tables to be edited in place on the web. This 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.

4.10.1. Types of decision table

There are broadly two different types of decision table, both of which are supported in Guvnor:-

- Extended Entry
- Limited Entry

4.10.1.1. Extended Entry

An Extended Entry decision table is one for which the column definitions, or stubs, 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. Guvnor supports use of Java enumerations, Guvnor enumerations or decision table "optional value lists" to restrict value entry.






+ Decision table					
	#	Description	Age	Make	Premium
			Applicant [\$a]	Vehicle [\$v]	
			age [<]	make [==]	
 	1		35	BMW	1000
 	2		35	Audi	1000

Figure 4.29. Extended Entry Decision table

4.10.1.2. 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.

+

Decision table

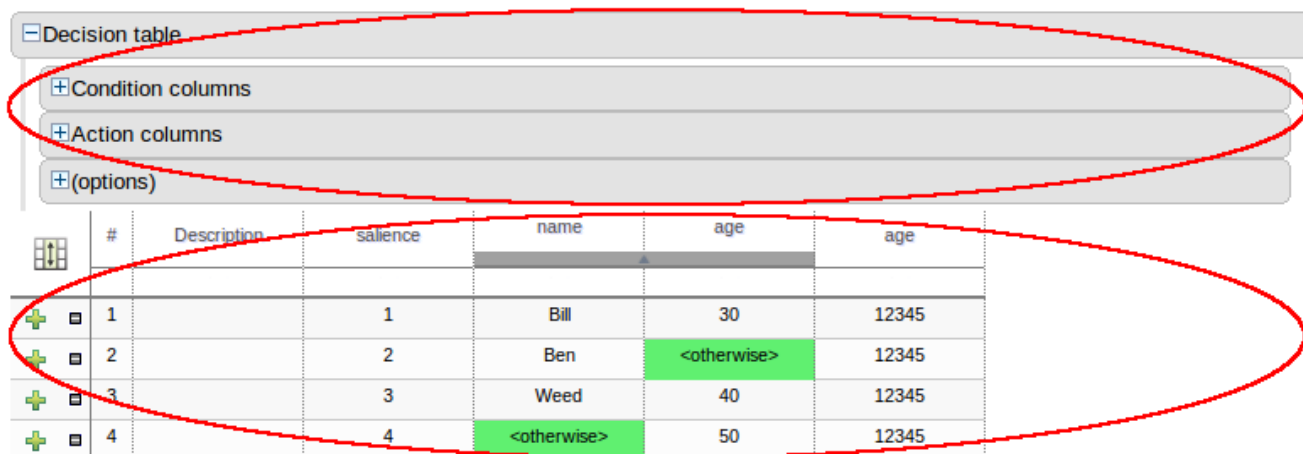
<div> <div></div> <div></div> </div>	#	Description	Age < 35	BMW	Audi	Premium 1000	
			Applicant [Sa]	Vehicle [Sv]			
			age [<35]	make [==BMW]	make [==Audi]		
<div>+</div> <div></div>	1		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
<div>+</div> <div></div>	2		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
<div>+</div> <div></div>	3		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
<div>+</div> <div></div>	4		<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	
<div>+</div> <div></div>	5		<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
<div>+</div> <div></div>	6		<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
<div>+</div> <div></div>	7		<input checked="" type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	
<div>+</div> <div></div>	8		<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>	

Figure 4.30. Limited Entry Decision table

4.10.2. Main components\concepts

The guided decision table is split into two main sections:-

- The upper section allows table columns to be defined representing rule attributes, meta-data, conditions and actions.
- The lower section contains the actual table itself; where individual rows define separate rules.



The screenshot shows a 'Decision table' configuration interface. A red oval highlights the configuration section (Condition columns, Action columns, and options) and the table itself. The table has columns for #, Description, salience, name, age, and age. The data rows are as follows:

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

Figure 4.31. Main components

4.10.2.1. Navigation

Cells can be selected in a variety of ways:-

- Firstly individual cells can be double-clicked and a pop-up editor corresponding to the underlying data-type will appear. Groups of cells in the same column can be selected by either clicking in the first and dragging the mouse pointer or clicking in the first and clicking the extent of the required range with the shift key pressed.
- Secondly the keyboard cursor keys can be used to navigate around the table. Pressing the enter key will pop-up the correspondig editor. Ranges can be selected by pressing the shift key whilst extending the range with the cursor keys.

Columns can be resized by hovering over the corresponding divider in the table header. The mouse cursor will change and then the column width dragged either narrower or wider.

4.10.2.2. 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 4.32. Cell merging

4.10.2.3. 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 4.33. Cell grouping

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

4.10.2.4. Operation of "otherwise"

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
```

4.10.2.5. Re-arranging columns

Whole patterns and individual conditions can be re-arranged by dragging and dropping them in the configuration section of the screen. This allows constraints to be re-ordered to maximise performance of the resulting rules, by placing generalised constraints before more specific. Action columns can also be re-arranged by dragging and dropping them.

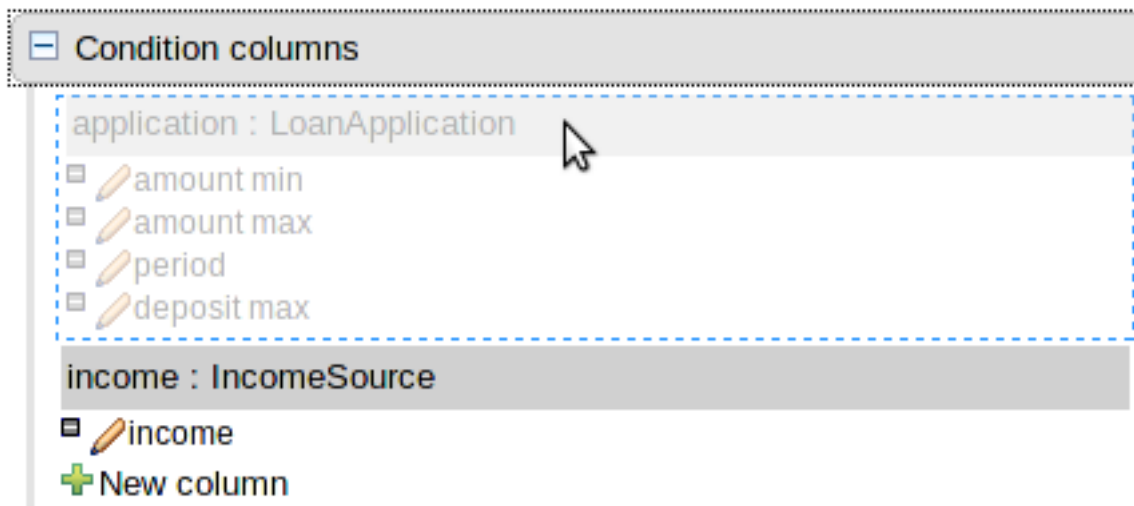


Figure 4.34. Re-arranging Condition patterns

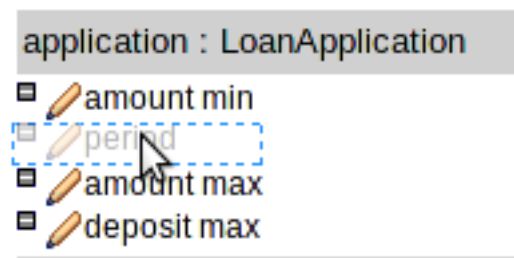


Figure 4.35. Re-arranging columns

4.10.3. Defining a web based decision table

4.10.3.1. Manual creation

When a new empty decision table has been created you need to define columns for Facts, their constraints and corresponding actions.

4.10.3.1.1. Column configuration

Expand the "Decision table" element and you will see three further sections for "Conditions", "Actions" and "Options". Expanding either the "Conditions" or "Actions" sections reveals the "New column" icon. This can be used to add new column definitions to the corresponding section. Existing columns can be removed by clicking the "-" icon beside each column name, or edited by clicking the "pencil" icon also beside each column name. The "Options" section functions slightly differently however the principle is the same: clicking the "Add Attribute/Metadata" icon allows columns for table attributes to be defined (such as "salience", "no-loop" etc) or metadata added.

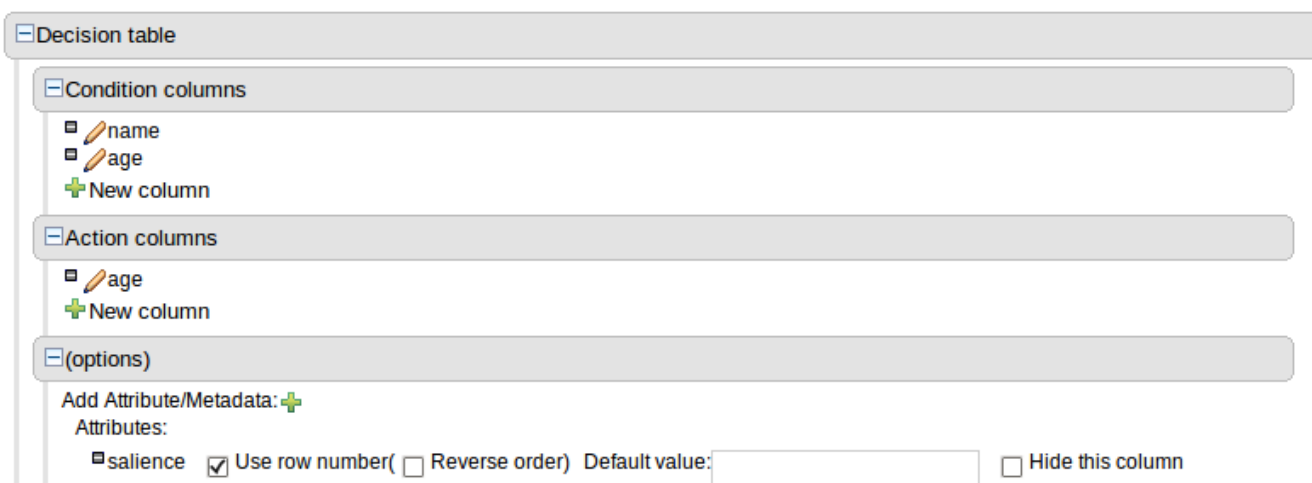


Figure 4.36. Column configuration

4.10.3.1.1.1. Utility columns

Two columns containing rule number and description are provided by default.

4.10.3.1.1.2. 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
```

4.10.3.1.1.3. Meta-data columns

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

4.10.3.1.1.4. 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.

4.10.3.1.1.5. Action columns

Action columns can be defined to perform the following simple operations:-

- Set the value of fields on previously bound facts, with the option to notify the Rule Engine of the modified values which could lead to other rules being re-activated.
- Insert new Facts into the Rule Engine Working Memory and set the values of the new Facts' fields. You can choose to have the new Fact "logically inserted" meaning it will be automatically retracted should the conditions leading to the action being invoked cease to be true. Please refer to the Drools Expert documentation for details on Truth Maintenance and Logical insertions.
- Retract a bound fact.

4.10.3.2. Using a Wizard

A Wizard can also be used to assist with defining the decision table columns.

The wizard can be chosen when first electing to create a new rule. The wizard provides a number of pages to define the table:-

- Summary
- Add Fact Patterns
- Add Constraints
- Add Actions to update facts
- Add Actions to insert facts
- Columns to expand

4.10.3.2.1. Selecting the wizard

The "New Wizard" dialog shows a "Use wizard" checkbox when the asset type is set to "Decision Table (Web - guided editor)".

New Rule

☒ Create new:

☐ Import asset from global area:

Name:

Initial category: ▼

- ☒ Home Mortgage
- ☒ Commercial Mortgage

Type (format) of rule: ▼

☒ Use Wizard

Options: ☐ Extended entry, values defined in table body

☒ Limited entry, values defined in columns

☒ Create in Package: ▼

☐ Create in Global area

Initial description:

Figure 4.37. Selecting the wizard

4.10.3.2.2. Summary page

the summary page shows a few basic details about the decision table and allows the asset name to be changed.

The screenshot shows a window titled "Guided Decision Table Wizard" with a close button in the top right corner. On the left side, there is a list of steps, each preceded by a green checkmark: "Summary", "Add Fact Patterns", "Add Constraints", "Add Actions to update Facts", "Add Actions to insert Facts", and "Columns to expand". The "Summary" step is currently selected. The main area of the wizard is titled "Summary of fields for the decision table." and contains three input fields: "Name:" with the value "example" and a red asterisk indicating a required field, "Initial description:" which is empty, and "Create in Package:" with the value "cep". At the bottom of the window, there are four buttons: "< Previous", "Next >", "Cancel", and "Finish".

Figure 4.38. Summary page

4.10.3.2.3. Add Fact Patterns page

This page allows Fact types to be defined that will form the "When" columns of the rules. Fact types that are available in your model will be shown in the left-hand listbox. Select a Fact type and use the ">>" button to add it to your list of chosen facts on the right-hand listbox. Removal is a similar process: the Fact that is no longer required can be selected in the right-hand listbox and the "<<" button used to remove it. All Fact types need to be bound to a variable. Incomplete Fact types will be highlighted and a warning message displayed. You will be unable to finish your definition until all warnings have been resolved.

The screenshot shows the 'Add Fact Patterns' page of the 'Guided Decision Table Wizard'. On the left, a sidebar lists steps: Summary, Add Fact Patterns (highlighted), Add Constraints, Add Actions to update Facts, Add Actions to insert Facts, and Columns to expand. The main area is titled 'Define Facts\Patterns on which constraints can be defined.' It contains two lists: 'Available patterns' (ArrayList, Cheese, Collection, List, TelephoneCall) and 'Chosen patterns' (tc : TelephoneCall). Between the lists are '>>' and '<<' buttons. Below the lists are input fields for 'Binding: tc', 'From Entry Point:', and 'Over sliding window: --- None ---'. At the bottom are buttons for '<- Previous', 'Next ->', 'Cancel', and 'Finish'.

Figure 4.39. Add Fact Patterns page

This screenshot shows the same 'Add Fact Patterns' page but in an error state. A red banner at the top reads 'Fact definitions incomplete'. The 'Chosen patterns' list now shows 'TelephoneCall' in red text. The 'Binding' field is also highlighted in red. A green callout box with an arrow pointing to the 'Add Fact Patterns' step in the sidebar contains the text: 'The page has errors and is therefore marked as incomplete. The wizard cannot be finished.' The 'Finish' button at the bottom is also highlighted in green.

Figure 4.40. Example of an incomplete Fact definition

4.10.3.2.4. Add Constraints page

This page allows field constraints on the Fact types you have chosen to use in the decision table to be defined. Fact types chosen on the previous Wizard page are listed in the right-hand listbox. Selecting a Fact type by clicking on it will result in a list of available fields being shown in the middle listbox together with an option to create a predicate that do not require a specific field. Fields can be added to the pattern's constraints by clicking on the field and then the ">>" button. Fields can be removed from the pattern definition by clicking on the Condition in the right-hand listbox and then the "<<" button. All fields need to have a column header and operator. Incomplete fields will be highlighted and a warning message displayed. You will be unable to finish your definition until all warnings have been resolved.

Figure 4.41. Add Constraints page

4.10.3.2.5. Add Actions to update facts page

Fact types that have been defined can be updated in the consequence, or action, part of a rule. This page allows such actions to be defined. Fact types added to the decision table definition are listed in the left-hand listbox. Selecting a Fact type by clicking on it will result in a list of available fields being shown in the middle listbox. Fields that need to be updated by the rule can be added by selecting an available field and pressing the ">>" button. Fields can be removed similarly by clicking on a chosen field and then the "<<" button. All actions require a column header. Any incomplete actions will be highlighted and a warning message displayed. You will be unable to finish your definition until all warnings have been resolved.

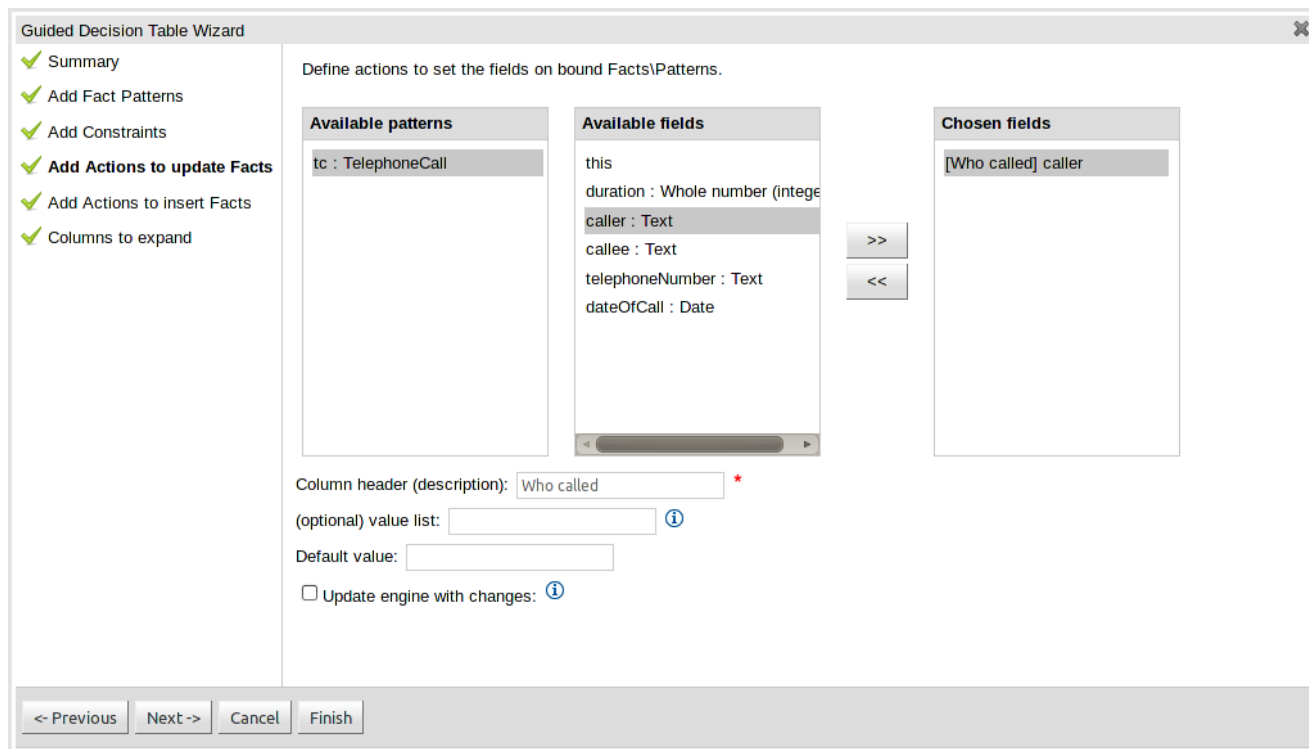


Figure 4.42. Add Actions to update facts page

4.10.3.2.6. Add Actions to insert facts page

Actions can also be defined to insert new Facts into the Rule Engine. A list of Fact types available in your model are listed in the left-hand listbox. Select those you wish to include in your decision table definition by clicking on them and pressing the ">>" button between the left most listbox and that titled "Chosen patterns". Removal is a similar process whereby a chosen pattern can be selected and removed by pressing the "<<" button. Selection of a chosen pattern presents the user with a list of available fields. Fields that need to have values set by the action can be added by selecting them and pressing the ">>" button between the "Available fields" and "Chosen fields" listbox. Removal is a similar process as already described. New Facts need to be bound to a variable and have a column heading specified. Incomplete Facts and/or fields will be highlighted and a warning message displayed. You will be unable to finish your definition until all warnings have been resolved.

Guided Decision Table Wizard

- ✓ Summary
- ✓ Add Fact Patterns
- ✓ Add Constraints
- ✓ Add Actions to update Facts
- ✓ **Add Actions to insert Facts**
- ✓ Columns to expand

Define actions to insert new Facts\Patterns.

Available patterns	Chosen patterns	Available fields	Chosen fields
ArrayList	c : Cheese	this	[Cheese] f1
Cheese		f1 : Text	
Collection		f2 : Text	
List		f3 : Text	
TelephoneCall			

Binding: c

☐ Logically assert a fact - the fact will be retracted when the supporting evidence is removed.

Column header (description): Cheese

(optional) value list:

Default value:

<- Previous Next -> Cancel Finish

Figure 4.43. Add Actions to insert facts page

4.10.3.2.7. Columns to expand page

This page controls how the decision table, based upon Conditions defined on the prior pages, will be created. Condition columns defined with an optional list of permitted values can be used to create rows in the decision table. Where a number of Condition columns have been defined with lists of permitted values the resulting table will contain a row for every combination of values; i.e. the decision table will be in expanded form. By default all Condition columns defined with value lists will be included in the expansion however you are able to select a sub-set of columns if so required. This can be accomplished by unticking the "Fully expand" checkbox and adding columns to the right-hand listbox. If no expansion is required untick the "Fully expand" checkbox and ensure zero columns are added to the right-hand listbox.

The screenshot shows the 'Guided Decision Table Wizard' window. On the left, a sidebar lists steps: Summary, Add Fact Patterns, Add Constraints, Add Actions to update Facts, Add Actions to insert Facts, and Columns to expand (which is highlighted with a green checkmark). The main area is titled 'Define the columns from which the generated table will be expanded.' It contains a checkbox 'Fully expand the table, including all columns.' Below this are two panels: 'Available columns' and 'Chosen columns'. The 'Available columns' panel contains the text '[Who called] caller'. Between the panels are '>>' and '<<' buttons. At the bottom are buttons for '<- Previous', 'Next ->', 'Cancel', and 'Finish'.

Figure 4.44. Columns to expand page

The screenshot shows the 'Guided Decision Table Wizard' window at the 'Define constraints on the Facts\Patterns fields.' step. The sidebar on the left shows 'Add Constraints' highlighted with a green checkmark. The main area has three panels: 'Available patterns' (containing 'tc : TelephoneCall'), 'Available fields' (containing 'this', 'duration : Whole number (integer)', 'caller : Text', 'callee : Text', 'telephoneNumber : Text', 'dateOfCall : Date', and '[New Predicate]'), and 'Conditions' (containing '[Date of call] dateOfCall' and '[Who called] caller'). Between the panels are '>>' and '<<' buttons. To the right of the 'Conditions' panel are up and down arrow buttons. Below the panels, there are fields for 'Calculation type:' (with radio buttons for 'Literal value' and 'Formula'), 'Column header (description):' (with text 'Who called'), 'Operator:' (with a dropdown menu showing 'equal to'), '(optional) value list:' (with text 'Rod,Jane,Freddie' and a red circle around it), and 'Default value:'. At the bottom are buttons for '<- Previous', 'Next ->', 'Cancel', and 'Finish'.

Figure 4.45. Example of a Condition column with optional values defined

Decision table

Condition columns

Date of call

Who called

New column

Action columns

(options)




	#	Description	Date of call	Who called
			TelephoneCall [tc]	
			dateOfCall [==]	caller [==]
	1			Rod
	2			Jane
	3			Freddie

Figure 4.46. Example of a decision table generated with expanded columns

4.10.4. Rule definition

This section allows individual rules to be defined using the columns defined earlier.

Rows can be appended to the end of the table by selecting the "Add Row" button. Rows can also be inserted by clicking the "+" icon beside an existing row. The "-" icon can be used to delete rows.

				name	age
	#	Description	sallience	Person [Sp]	
				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 4.47. Rule definition

4.11. Templates of assets/rules

The guided rule editor is great when you need to define a single rule, however if you need to define multiple rules following the same structure but with different values in field constraints or action sections a "Rule Template" is a valuable asset. Rule templates allow the user to define a rule structure with place-holders for values that are to be interpolated from a table of data. Literal values, formulae and expressions can also continue to be used.

Rule Templates can often be used as an alternative for Decision Tables in Drools Guvnor.

4.11.1. Creating a rule template

To create a template for a rule simply select the "New Rule Template" from the Knowledge Bases "Create New" popup menu. The create "New Rule Template" asset popup window will appear from which the normal asset attributes can be defined; such as name, category and description.



Figure 4.48. Create a template

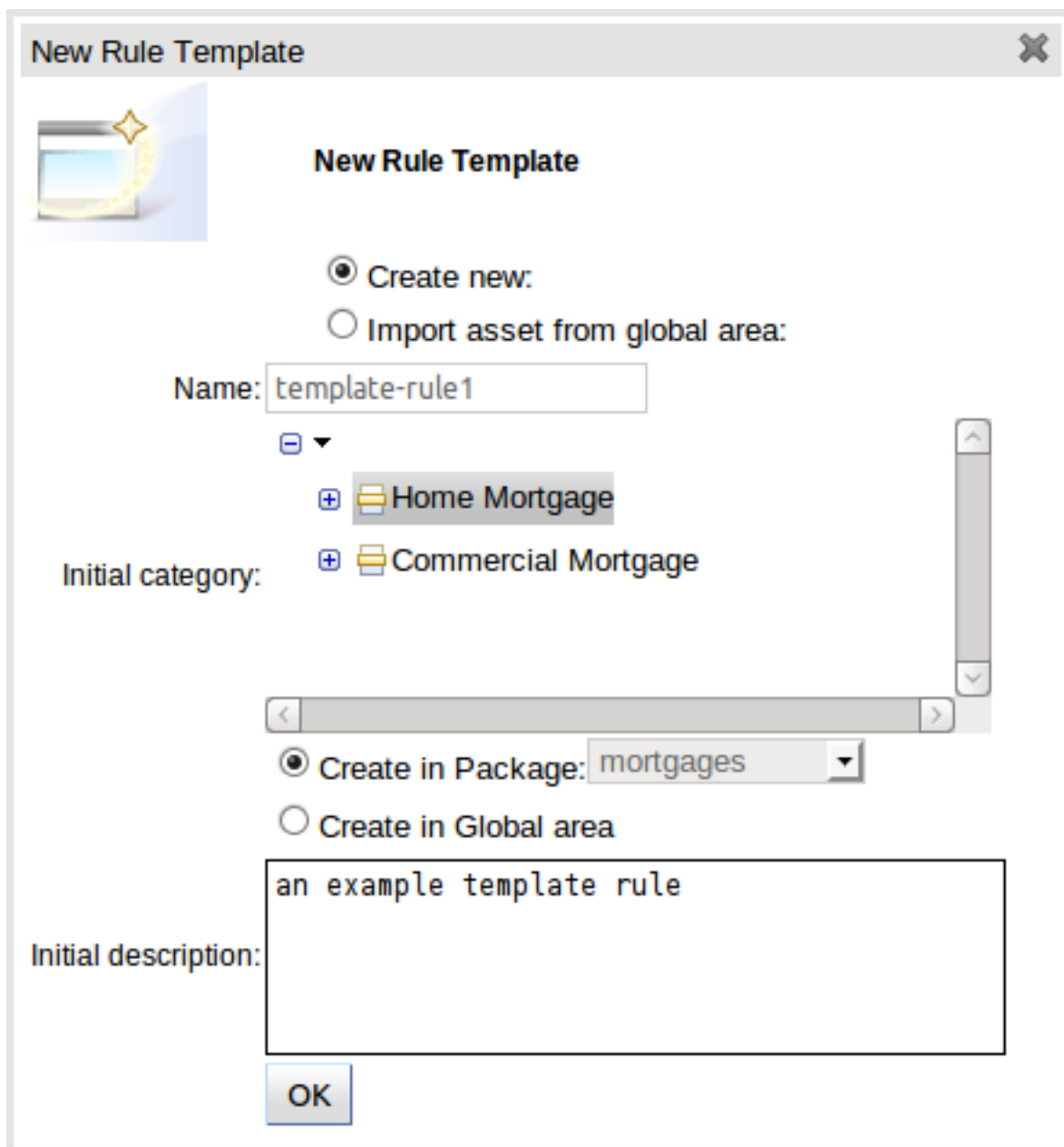


Figure 4.49. Create "New Rule Template" popup

4.11.2. Define the template

Once a rule template has been created the editor is displayed. The editor takes the form of the standard guided editor explained in more detail under the "Rule Authoring" section. As the rule is constructed you are given the ability to insert "Template Keys" as place-holders within your field constraints and action sections. Literal values, formulae and expressions can continue to be used as in the standard guided editor.

Field value

Literal value: ⓘ

Template key: ⓘ

Advanced options:

A formula: ⓘ

Expression editor: ⓘ

Figure 4.50. Template Key popup

The following screenshot illustrates a simple rule that has been defined with a "Template Key" for the applicants' maximum age, minimum age and credit rating. The template keys have been defined as "\$max_age", "\$min_age" and "\$cr" respectively.

File Edit Source Status: [Draft]

Attributes Edit

Load Template Data

WHEN

There is an Applicant with:

1. age less than \$max_age

age greater than or equal to \$min_age

creditRating equal to \$cr

2. There is a LoanApplication [\$a]

THEN

1. Modify value of LoanApplication [\$a] approved false

(show options...)

Figure 4.51. Rule template in the guided editor

4.11.3. Defining the template data

When you have completed the definition of your rule template you need to enter the data that will be used to interpolate the "Template Key" place-holders. Drools Guvnor provides the facility to

enter data in a flexible grid within the guided editor screen. The grid editor can be launched by pressing the "Load Template Data" button on the guided editor screen.

The rule template data grid is very flexible; with different pop-up editors for the underlying fields' data-types. Columns can be resized and sorted; and cells can be merged and grouped to facilitate rapid data entry.

One row of data interpolates the "Template Key" place-holders for a single rule; thus one row becomes one rule.






















Note

If any cells for a row are left blank a rule for the applicable row is not generated.

Template Data

Template Data

	\$max_age	\$min_age	\$scr
			
 	25	20	AA
 	25	20	OK
 	25	20	Sub prime
 	35	25	AA
 	35	25	OK
 	35	25	Sub prime
 	45	35	AA
 	45	35	OK
 	45	35	Sub prime

Save and close

Add row...

Figure 4.52. Template data grid

4.11.3.1. Cell merging

The icon in the top left of the grid 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.

Template Data

Template Data

	\$max_age	\$min_age	\$scr
	25	20	AA
			OK
			Sub prime
	35	25	AA
			OK
			Sub prime
	45	35	AA
			OK
			Sub prime

Save and close

Add row...




















Figure 4.53. Cell merging

4.11.3.2. 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.

Template Data

Template Data

	\$max_age	\$min_age	\$scr
 	 25	 20	AA
 			OK
 			Sub prime
 	35	 25	AA
 	 45	 35	AA
 			OK
 			Sub prime

Save and close

Add row...

Figure 4.54. Cell grouping

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

4.11.4. Generated DRL

Whilst not necessary, rule authors can view the DRL that will be generated for a "Rule Template" and associated data. This feature and its operation is no different to that for other assets. Select the "Source" -> "View Source" menu item from the Asset Editor screen. The DRL for all rules will be displayed.

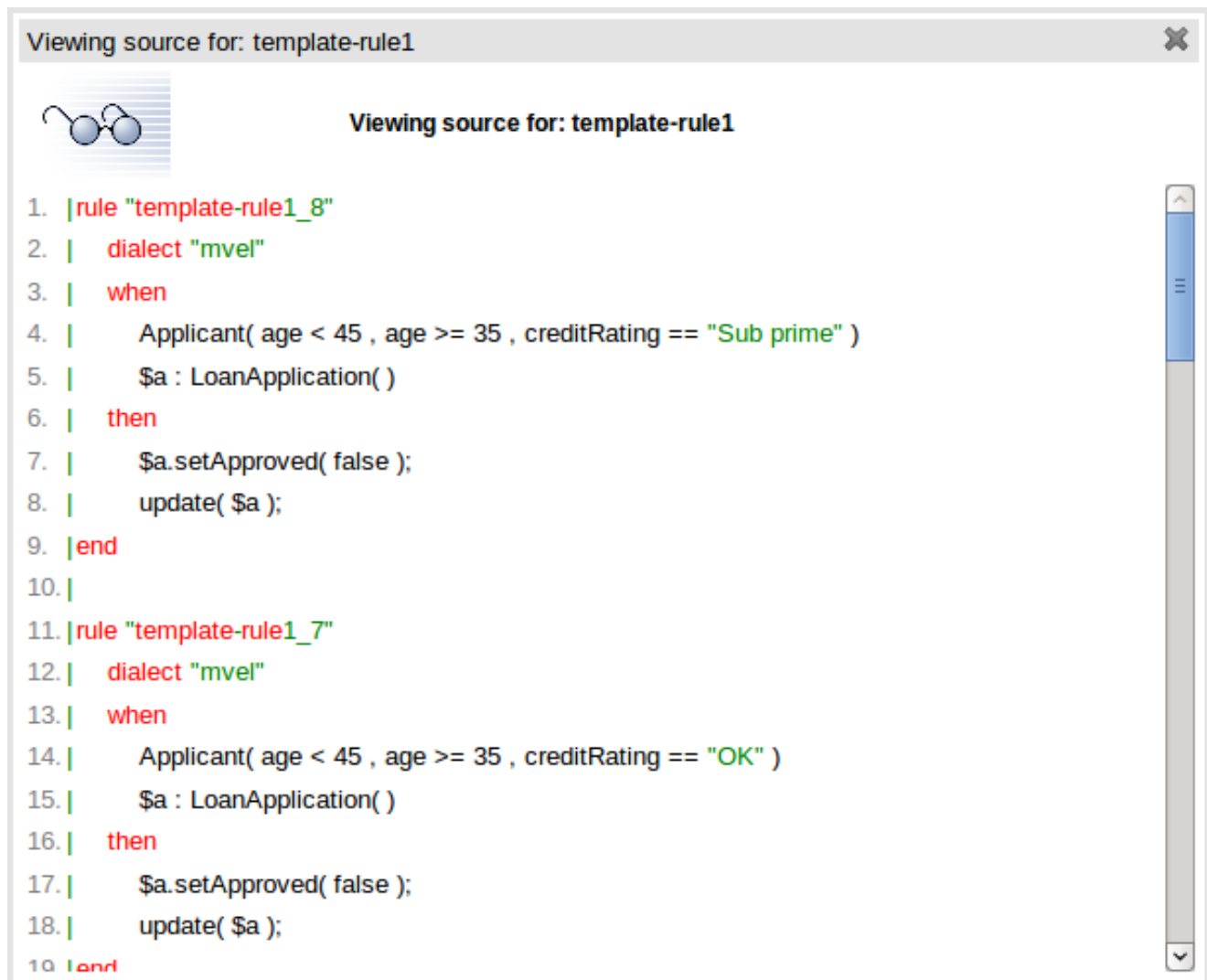


Figure 4.55. Generated DRL

4.12. The Fact Model

For any rule base application, a fact model is needed to drive the rules. The fact model typically overlaps with the applications domain model, but in general it will be decoupled from it (as it makes the rules easier to manage over time). There are no technical limitations on using your domain model as your fact model, however this introduces tighter coupling between your business domain (domain model) and your knowledge domain (fact model). Consequentially if your domain model were to change you would need to, at the very least, revisit your rule definitions.

4.12.1. Ways to define a Fact Model

There are two ways to do define your fact model; each of which will be discussed in more detail in the following sections.

- Upload a JAR file containing Java Classes used by both your application and rules.

- Declare a model within Guvnor; that can be exported as a KnowledgeBase and used within your Java code.

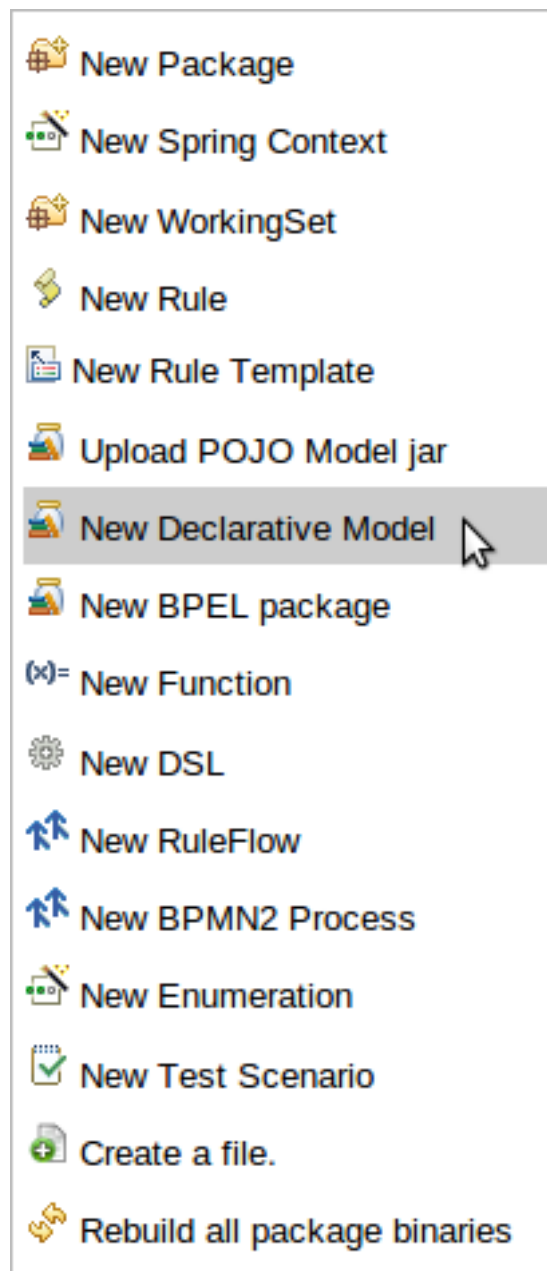


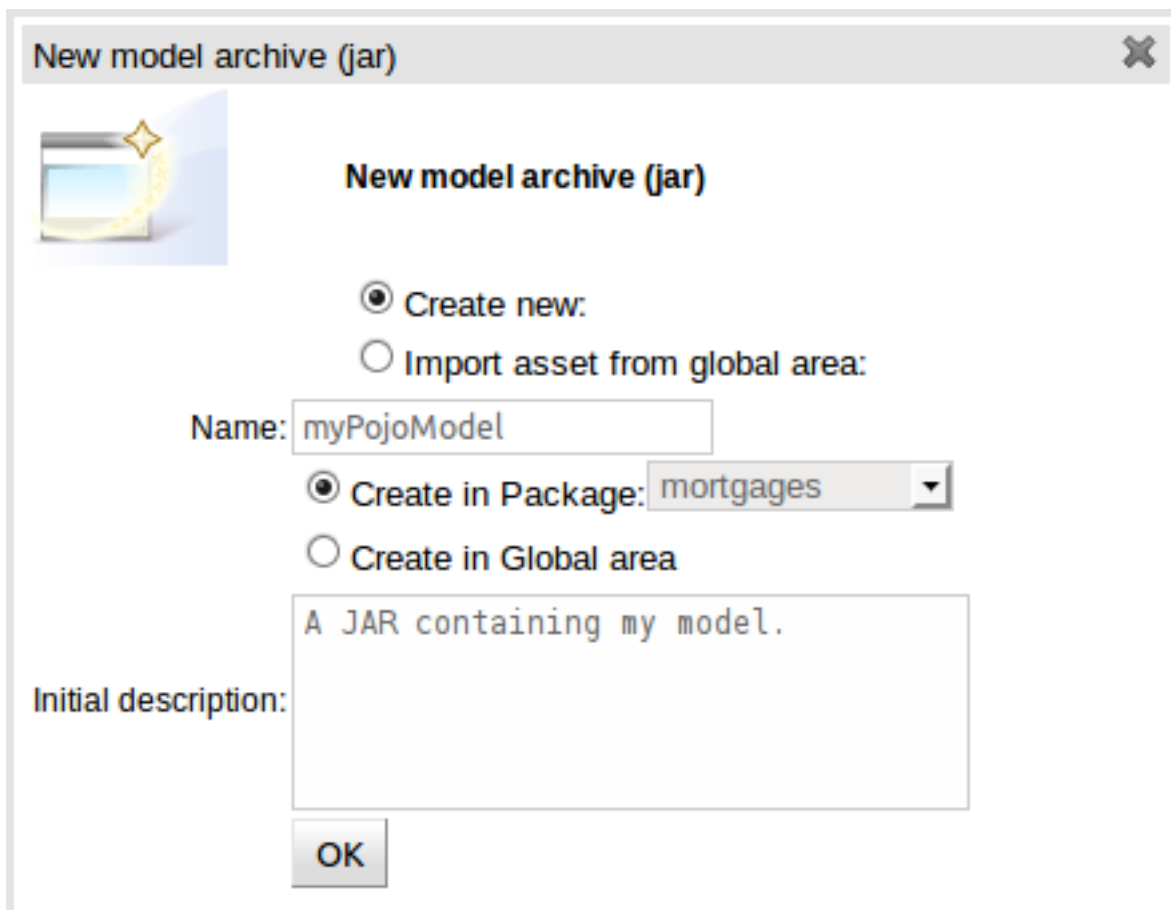
Figure 4.56. Choosing a model type

4.12.2. Creating a JAR Model

Creating and uploading a JAR model file is a two step process.

4.12.2.1. Create a JAR Model asset

Select "Upload POJO Model JAR" from the "Create New" popup menu from the "Knowledge Bases" section of the Explorer widget. This will launch the "New Asset" configuration screen from which the new upload can be given basic details such as name, category and a description.



New model archive (jar)

New model archive (jar)

☒ Create new:
☐ Import asset from global area:

Name: myPojoModel

☒ Create in Package: mortgages
☐ Create in Global area

Initial description:
A JAR containing my model.

OK

Figure 4.57. Creating a JAR Model asset

4.12.2.2. Upload a JAR Model into the asset

Once the POJO Model JAR asset has been created you are presented with a screen to upload the actual JAR containing the model defined as Java classes and packaged in a regular Java JAR file. Many Java IDE's are able to export classes as a JAR file.

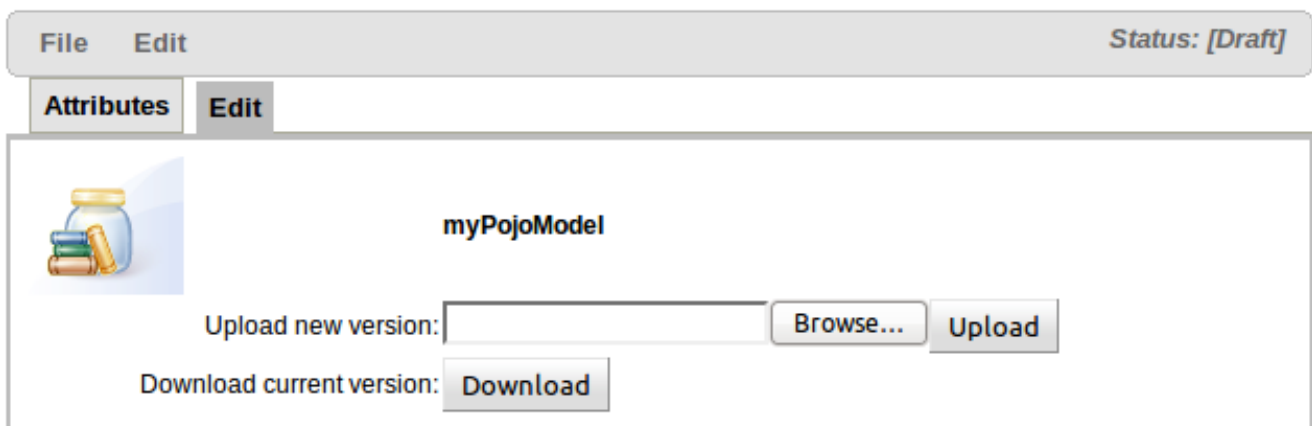


Figure 4.58. Uploading the JAR

4.12.3. Declarative model

Why would you chose declared types over JAR files: generally this reinforces the fact that the model "belongs" to the KnowledgeBase, rather than the application, and allows the model to have a lifecycle separate from the application. It also allows Java types to be enriched with Rule specific annotations. Additionally it also removes the burden of keeping JAR files synchronised between rules and the applications that use the rules.

Declarative models can be either:-

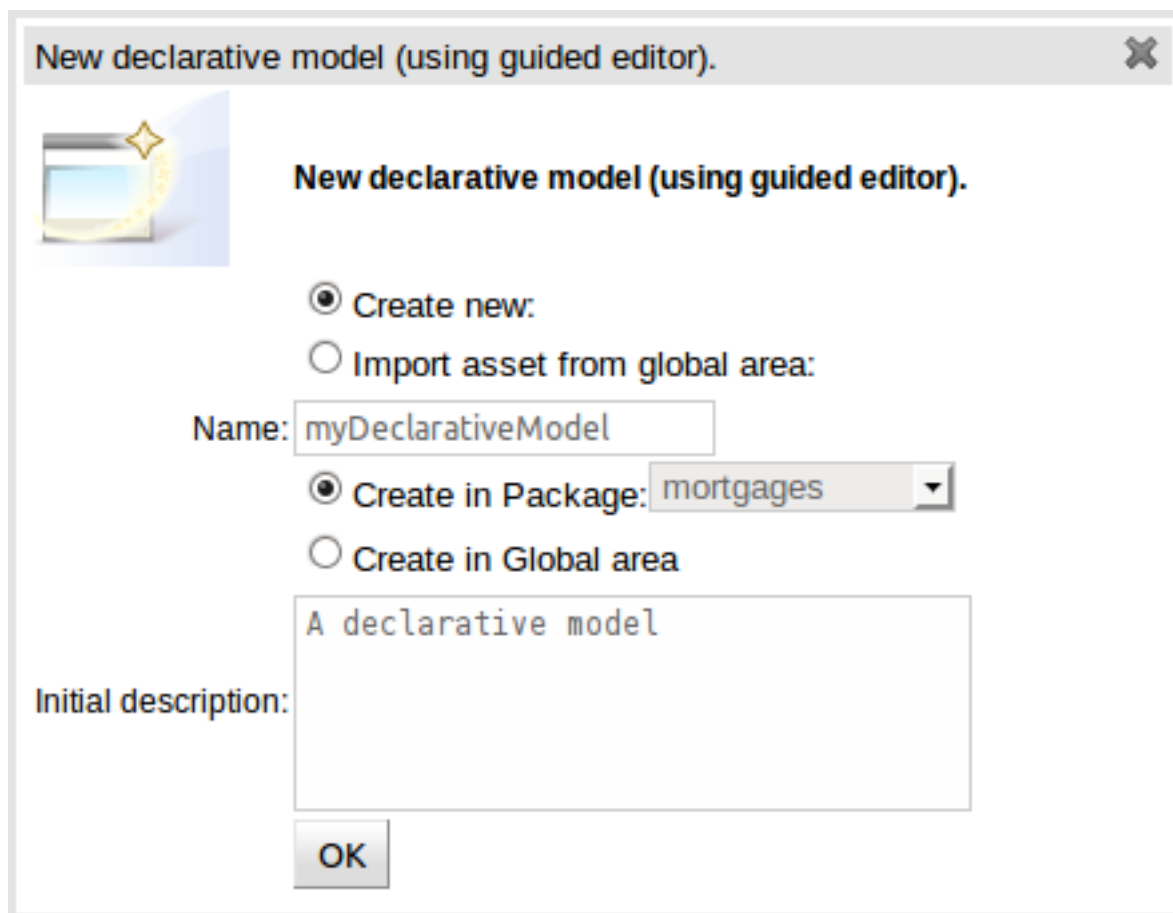
- A standalone definition of the entire Fact model used within your rules.
- Supplementary Fact defintions to support a Java POJO Model.
- Used to enrich a Java JAR model as uploaded in the previous section. Enriching JAR models allows annotations used by Drools (such as a "role" of type "event" for Facts used as events in Complex Event Processing) to be appended to classes. When enriching an existing Java JAR model the package name in Guvnor needs to be identical to the Java package name containing the class(es) you wish to enrich.

4.12.3.1. Creating a Declarative Model

Creating a Declarative Model is a two step process.

4.12.3.1.1. Create a Declarative Model asset

Select "New Declarative Model" from the "Create New" popup menu from the "Knowledge Bases" section of the Explorer widget. This will launch the "New Asset" configuration screen from which the new upload can be given basic details such as name, category and a description.



New declarative model (using guided editor).

New declarative model (using guided editor).

☒ Create new:

☐ Import asset from global area:

Name:

☒ Create in Package:

☐ Create in Global area

Initial description:

OK

Figure 4.59. Creating a Declarative Model asset

4.12.3.1.2. Defining the model

Once the Declarative Model asset has been created you are presented with the initial modelling screen; that is empty to begin.



File Edit Source Status: [Draft]

Attributes Edit

+ Add new fact type

Figure 4.60. Initial modelling screen

Facts, being semantically equivalent to Java classes, can be created by selecting the "Add new fact type" button. An existing Fact definition can be edited by clicking the "pencil" icon on the same row as the Fact name. Furthermore existing Facts can be deleted by clicking the "[-]" icon.

Figure 4.61. New fact popup

Figure 4.62. A Declarative model with one Fact defined

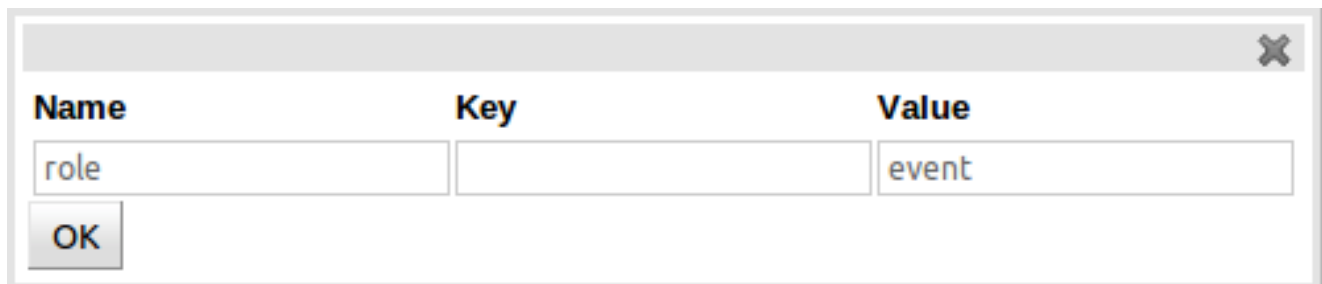
Fact Fields can be created by selecting the "Add field" button. The type of a field is suggested by a list (but this list is not exhaustive). An existing Fact Field definition can be edited by clicking the "pencil" icon on the same row as the Fact Field name. Furthermore existing Fact Fields can be deleted by clicking the "[-]" icon.

Figure 4.63. Fact Field popup

Fact annotations can be created by selecting the "Add annotation" button. Annotations are listed under the Fact title, before the fields, by convention. Annotations are prefixed with the "@" symbol. This not only makes them instantly recognisable but is also consistent with their definition in DRL.

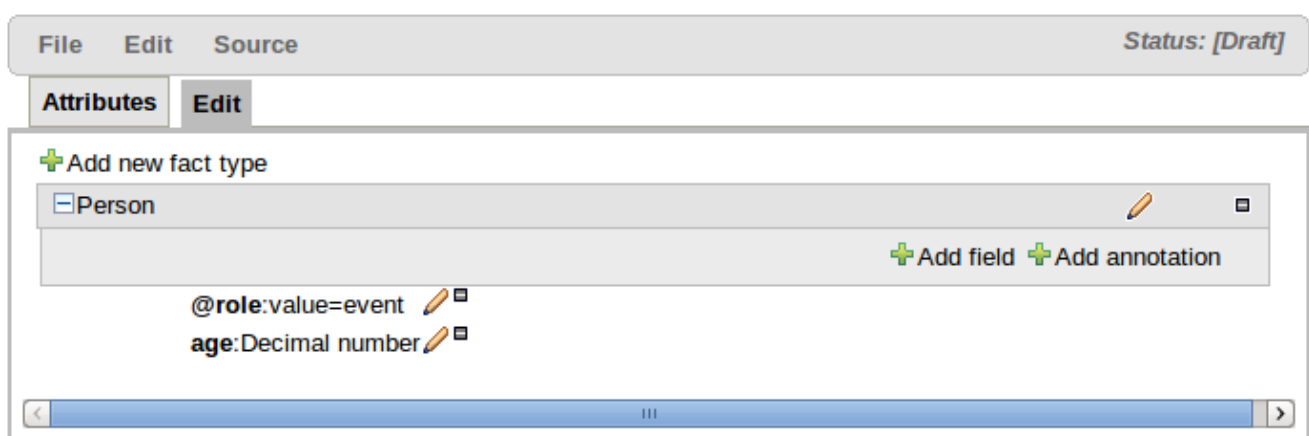
The annotation "Name" and "Value" are mandatory whereas the "Key" is optional. If a "Key" is not given a default value of "value" will be assigned. This is consistent with how annotations are held within Drools Expert.

An existing Fact Annotation can be edited by clicking the "pencil" icon on the same row as the Fact Annotation name. Furthermore existing Fact Annotations can be deleted by clicking the "[-]" icon.



A dialog box titled "Fact annotation popup" with a close button (X) in the top right corner. It contains three input fields labeled "Name", "Key", and "Value". The "Name" field contains the text "role", the "Key" field is empty, and the "Value" field contains the text "event". Below the fields is an "OK" button.

Figure 4.64. Fact annotation popup



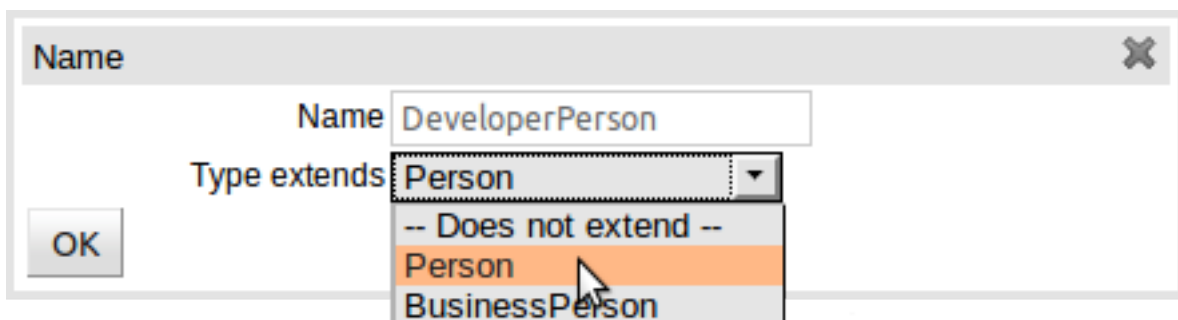
A screenshot of a software interface for defining a fact type. At the top is a menu bar with "File", "Edit", and "Source", and a status indicator "Status: [Draft]". Below the menu bar are two tabs: "Attributes" and "Edit". The "Edit" tab is active, showing a list of fact types. The first fact type is "Person", which is expanded to show its attributes: "@role:value=event" and "age:Decimal number". To the right of the "Person" fact type are two buttons: "+ Add field" and "+ Add annotation".

Figure 4.65. A completed definition

4.12.3.1.3. Extending the model

Declarative types can extend existing:-

- Java classes uploaded as part of a JAR (POJO) model
- Other declared types in the same package.



A dialog box titled "Extending an existing type" with a close button (X) in the top right corner. It contains two input fields: "Name" and "Type extends". The "Name" field contains the text "DeveloperPerson". The "Type extends" field contains a dropdown menu with the following options: "Person", "-- Does not extend --", "Person", and "BusinessPerson". The "Person" option is currently selected. Below the fields is an "OK" button.

Figure 4.66. Extending an existing type

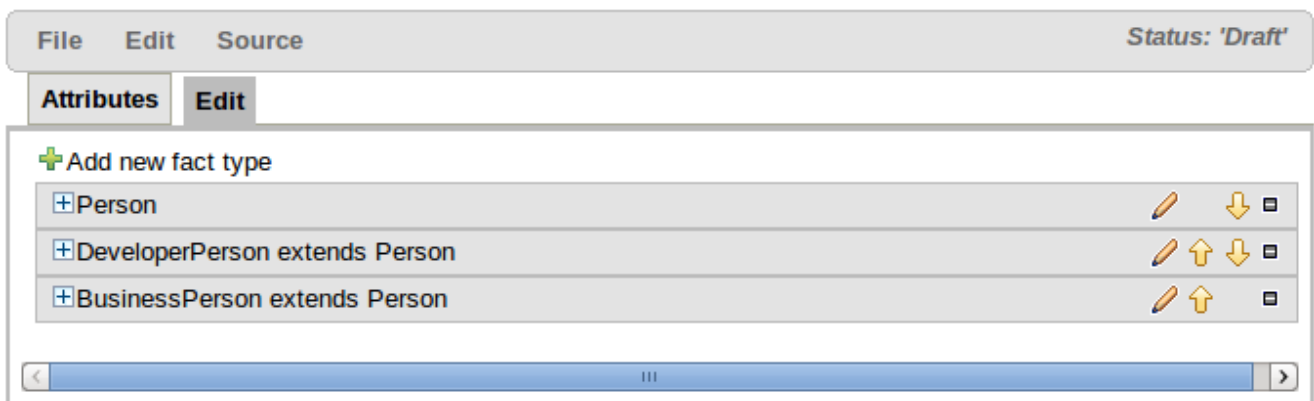


Figure 4.67. Declarative summary showing types that extend other types

4.12.3.1.3.1. Extending a Java class

To extend a Java class the following steps need to be completed:-

- Import the applicable Java JAR into Guvnor
- If the Java package name in which the class belongs is different to the Guvnor package name in to which the JAR has been imported ensure the Guvnor package imports the class from which you want to extend. This is normally completed for you automatically when you upload a JAR model however if you have multiple classes with the same name in the JAR you should check the appropriate one has been imported.
- Within the Declarative Modelling screen define an empty type (i.e. with no fields) of the same name as that you want to extend.
- Create a new declarative type as appropriate, extending the empty declaration created in the preceeding step.



Note

Only properties on Java classes that have both a "getter" and "setter" following standard Java Bean conventions are available on the declared sub-type. This is because the declarative symantics imply all properties on declared types have both accessors and mutators.

4.12.3.1.3.2. Extending existing declared types

To extend an existing declared type simply select the appropriate type from the "Type extends" dropdown when defining the sub-type.

4.12.3.1.4. Consuming a declarative model from Java

Declared types are generated at knowledge base compilation time, i.e. the application will only have access to them at application run time. Therefore, these classes are not available for direct reference from the application.

Declarative types can be used like normal fact objects, but the way you create them is different (as they are not on your applications classpath). To create these objects, they are available from the KnowledgeBase instance.

Example 4.1. Handling declared fact types through the API

```
// get a reference to a knowledge base with a declared type:
KnowledgeBase kbase = ...

// get the declared FactType
FactType personType = kbase.getFactType( "org.drools.examples",
                                          "Person" );

// handle the type as necessary:
// create instances:
Object bob = personType.newInstance();

// set attributes values
personType.set( bob,
               "name",
               "Bob" );
personType.set( bob,
               "age",
               42 );

// insert fact into a session
StatefulKnowledgeSession ksession = ...
ksession.insert( bob );
ksession.fireAllRules();

// read attributes
String name = personType.get( bob, "name" );
int age = personType.get( bob, "age" );
```



Note

The namespace of the declared type is the package namespace where it was declared (i.e. `org.drools.examples` in the above example).

4.13. Functions

Functions are another asset type. They are NOT rules, and should only be used when necessary. The function editor is a textual editor. Functions

```
function <returnType> funcName(<args here>) {  
    //code goes in here...  
}
```

Figure 4.68. Function

4.14. DSL editor

The DSL editor allows DSL Sentences to be authored. The reader should take time to explore DSL features in the Drools Expert documentation; as the syntax in Guvnor's DSL Editor is identical. The normal syntax is extended to provide "hints" to control how the DSL variable is rendered and validated within the user-interface.

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 "Guvnor enumeration" (i.e. defined as 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.

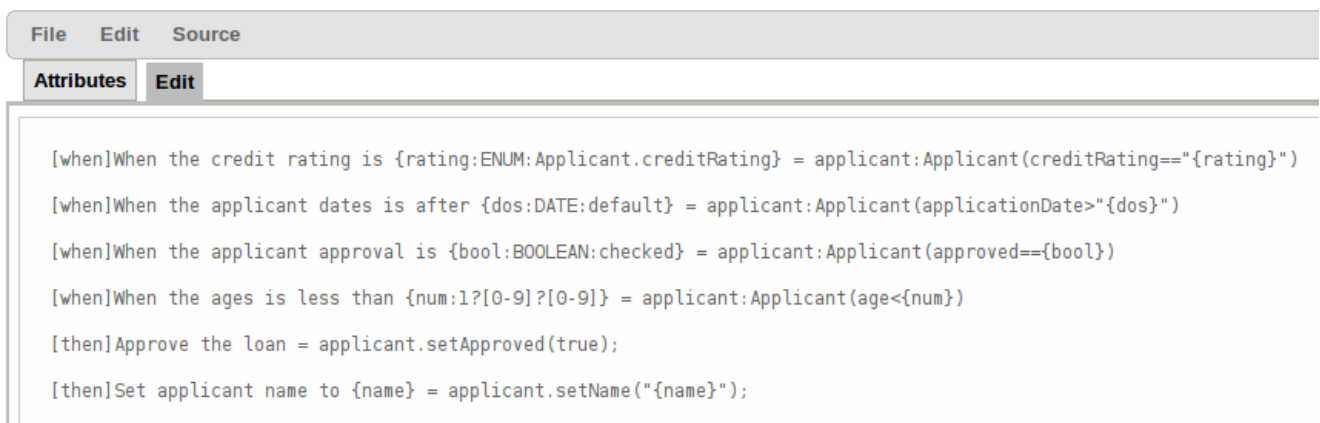
- {<varName>:BOOLEAN:<[checked | unchecked]>}

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.

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

This will render a button that will allow you to set the value of this variable using a Custom Form. In order to use this feature, a Working-Set containing a Custom Form Configuration for factType.fieldName must be active. If there is no such Working-Set, a simple text box is used (just like a regular variable).

For more information, please read more about Working-Sets and Custom Form Configurations.



```
[when]When the credit rating is {rating:ENUM:Applicant.creditRating} = applicant:Applicant(creditRating=="{rating}")
[when]When the applicant dates is after {dos:DATE:default} = applicant:Applicant(applicationDate>"{dos}")
[when]When the applicant approval is {bool:BOOLEAN:checked} = applicant:Applicant(approved=={bool})
[when]When the ages is less than {num:1?[0-9]?[0-9]} = applicant:Applicant(age<{num})
[then]Approve the loan = applicant.setApproved(true);
[then]Set applicant name to {name} = applicant.setName("{name}");
```

Figure 4.69. DSL rule

4.15. Rule flows

Rule flows: Rule flows allow you to visually describe the steps taken - so not all rules are evaluated at once, but there is a flow of logic. Rule flows are not covered in this chapter on the Guvnor, but you can use the IDE to graphically draw ruleflows, and upload the `.xrm` file to the Guvnor.

Similar to spreadsheets, you upload/download ruleflow files (the eclipse IDE has a graphical editor for them). The details of Rule Flows are not discussed here.

4.16. BPMN2 Process

Check the jBPM documentation: Chapter 10. Designer, for more information.

4.17. Work Item Definition

Check the jBPM documentation: Chapter 10. Designer, for more information.

4.18. Data enumerations (drop down list configurations)

Data enumerations are an optional asset type that technical folk can configure to provide drop down lists for the guided editor. These are stored and edited just like any other asset, and apply to the package that they belong to.

The contents of an enum config are a mapping of Fact.field to a list of values to be used in a drop down. That list can either be literal, or use a utility class (which you put on the classpath) to load a list of strings. The strings are either a value to be shown on a drop down, or a mapping from the code value (what ends up used in the rule) and a display value (see the example below, using the '=').

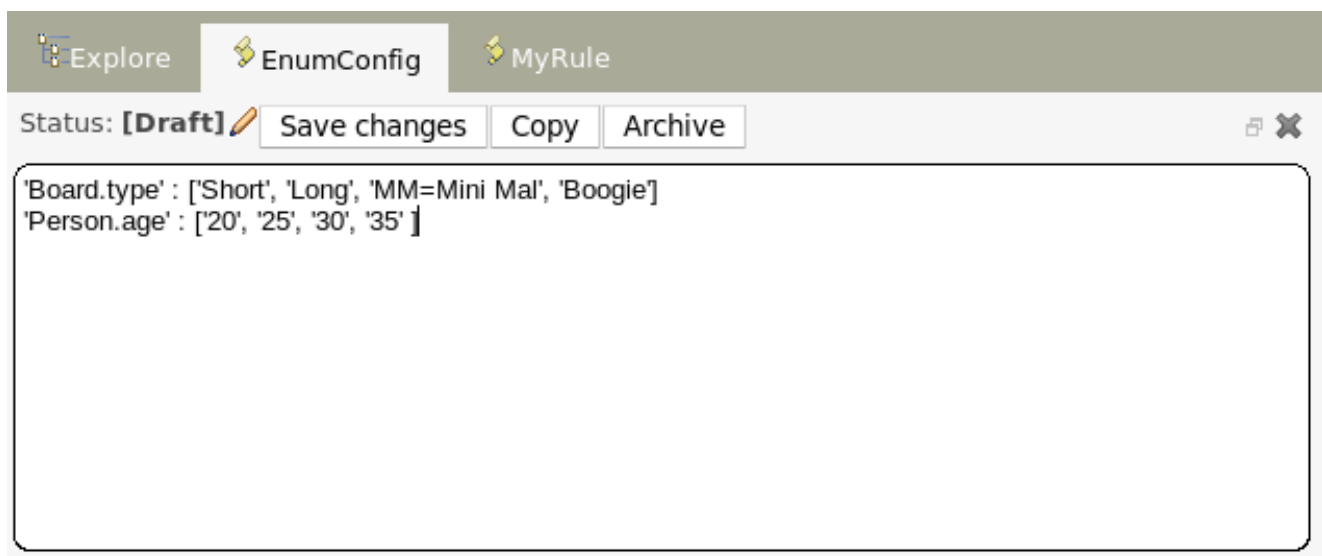


Figure 4.70. Data enumeration

In the above diagram - the "MM" indicates a value that will be used in the rule, yet "Mini Mal" will be displayed in the GUI.

Getting data lists from external data sources: It is possible to have the Guvnor call a piece of code which will load a list of Strings. To do this, you will need a bit of code that returns a `java.util.List` (of String's) to be on the classpath of the Guvnor. Instead of specifying a list of values in the Guvnor itself - the code can return the list of Strings (you can use the "=" inside the strings if you want to use a different display value to the rule value, as normal). For example, in the 'Person.age' line above, you could change it 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 (and is on the classpath). You can of course mix these "dynamic"

enumerations with fixed lists. You could for example load from a database using JDBC. The data enumerations are loaded the first time you use the guided editor in a session. If you have any guided editor sessions open - you will need to close and then open the rule to see the change. To check the enumeration is loaded - if you go to the Package configuration screen, you can "save and validate" the package - this will check it and provide any error feedback.

4.18.1. Advanced enumeration concepts

There are a few other advanced things you can do with data enumerations.

Drop down lists that depend on field values: Lets imagine a simple fact model, we have a class called Vehicle, which has 2 fields: "engineType" and "fuelType". We want to have a choice for the "engineType" of "Petrol" or "Diesel". Now, obviously the choice type for fuel must be dependent on the engine type (so for Petrol we have ULP and PULP, and for Diesel we have BIO and NORMAL). We can express this dependency in an enumeration as:

```
'Vehicle.engineType' : ['Petrol', 'Diesel']  
'Vehicle.fuelType[engineType=Petrol]' : ['ULP', 'PULP' ]  
'Vehicle.fuelType[engineType=Diesel]' : ['BIO', 'NORMAL' ]
```

This shows how it is possible to make the choices dependent on other field values. Note that once you pick the engineType, the choice list for the fuelType will be determined.

Loading enums programmatically: In some cases, people may want to load their enumeration data entirely from external data source (such as a relational database). To do this, you can implement a class that returns a Map. The key of the map is a string (which is the Fact.field name as shown above), and the value is a `java.util.List` of Strings.

```
public class SampleDataSource2 {  
  
    public Map<String>, List<String> loadData() {  
        Map data = new HashMap();  
  
        List d = new ArrayList();  
        d.add("value1");  
        d.add("value2");  
        data.put("Fact.field", d);  
  
        return data;  
    }  
}
```

And in the enumeration in the BRMS, you put:

```
=(new SampleDataSource2()).loadData()
```

The "=" tells it to load the data by executing your code.

Mode advanced enumerations: In the above cases, the values in the lists are calculated up front. This is fine for relatively static data, or small amounts of data. Imagine a scenario where you have lists of countries, each country has a list of states, each state has a list of localities, each locality has a list of streets and so on... You can see how this is a lot of data, and it can not be loaded up. The lists should be loaded dependent on what country was selected etc...

Well the above can be addressed in the following fashion:

```
'Fact.field[dependentField1, dependentField2]' : '(new  
"@{dependentField2}")'
```

Similar to above, but note that we have just specified what fields are needed, and also on the right of the ":" there are quotes around the expression. This expression will then be evaluated, only when needed, substituting the values from the fields specified. This means you can use the field values from the GUI to drive a database query, and drill down into data etc. When the drop down is loaded, or the rule loaded, it will refresh the list based on the fields. 'dependentField1' and 'dependentField2' are names of fields on the 'Fact' type - these are used to calculate the list of values which will be shown in a drop down if values for the "field".

4.19. Test Scenario

Test Scenarios are used to validate that rules and knowledge base work as expected. When the knowledge base evolves, Test Scenarios guard against regression.

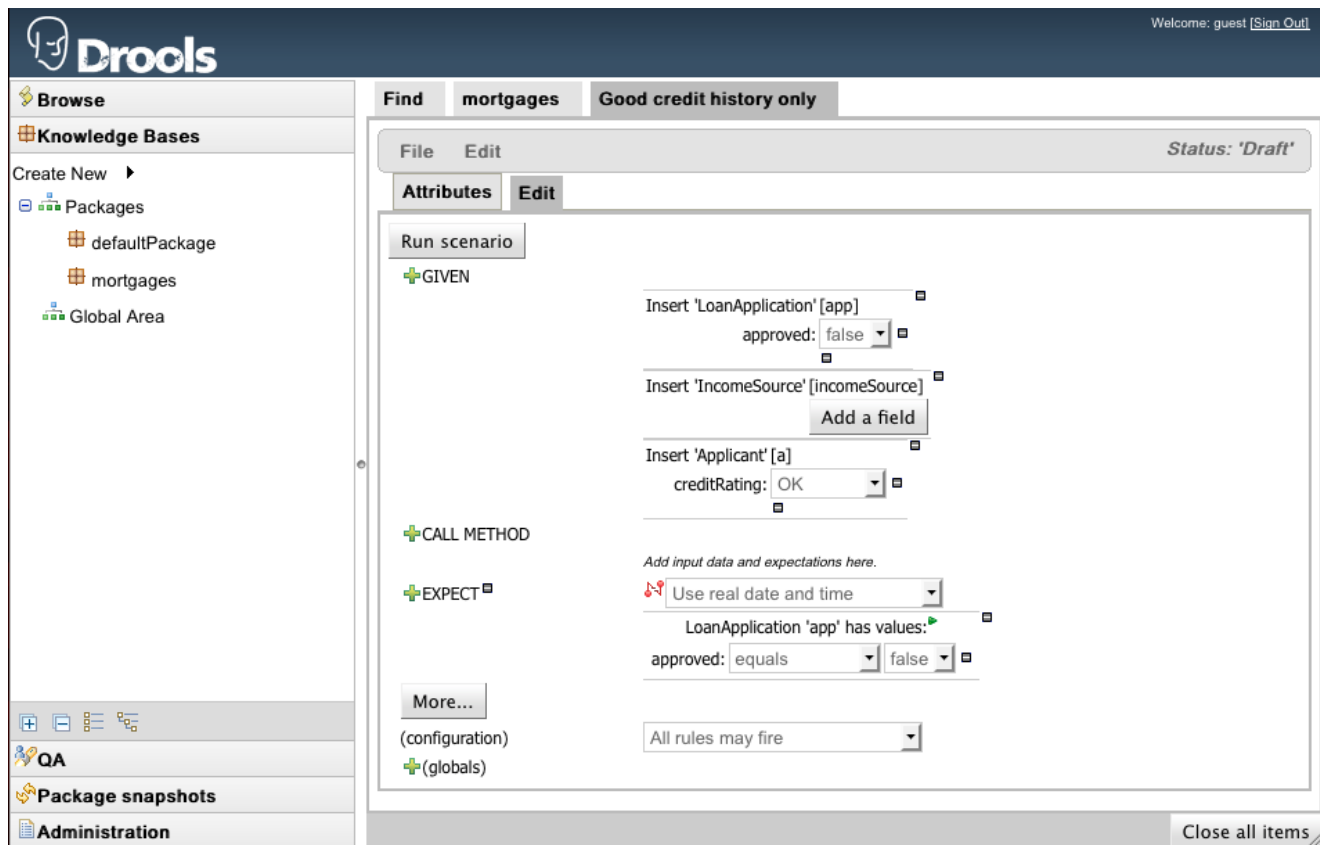


Figure 4.71. Example Test Scenario

Given section lists the facts needed for the behaviour. Expect section lists the expected changes and actions done by the behaviour. Given facts are passed for the Test Scenario before execution. During the rule execution, changes in the knowledge base are recorded. After the execution ends the recorded actions, existing facts in the knowledge base and knowledge base output is compared against the expectations.

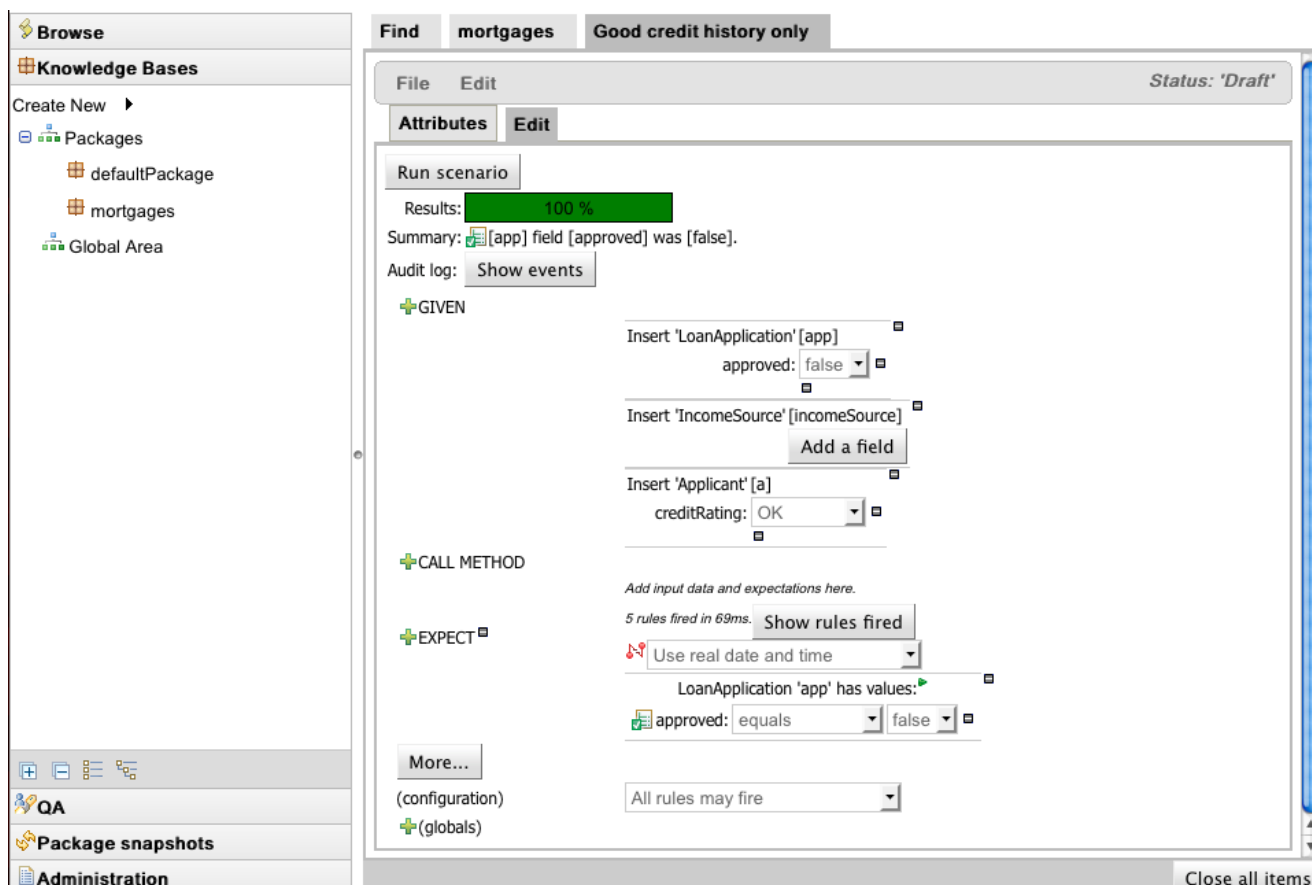


Figure 4.72. Example Test Scenario after execution

4.19.1. Given Section

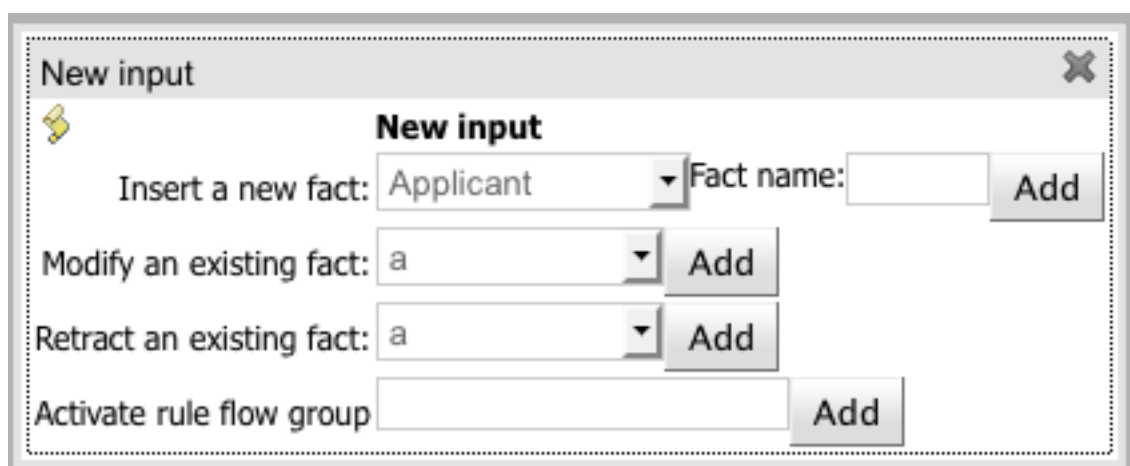


Figure 4.73. Given popup

- Insert a new fact - Adds a new fact that will be inserted into the knowledge base before execution.

- Modify an existing fact - Allows editing a fact between knowledge base executions.
- Retract an existing fact - Allows removing facts between executions.
- Activate rule flow group - Allows rules from a rule flow group to be tested, by activating the group in advance.

4.19.2. Expect Section

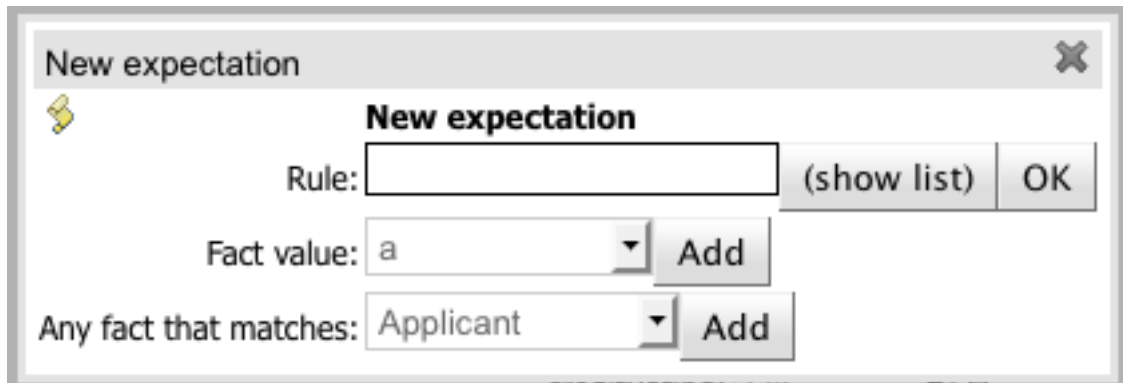


Figure 4.74. Expect popup

- Rule - Validate that a certain rule fired.
- Fact value - Validate fact values for a fact created in the Given section.
- Any fact that matches - Validate that there is at least one fact in the knowledge base with the specified field values.

4.19.3. Global Section

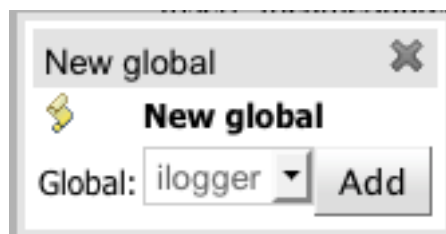


Figure 4.75. Global popup

- Global - Validate that the global field values.

4.19.4. New Input Section

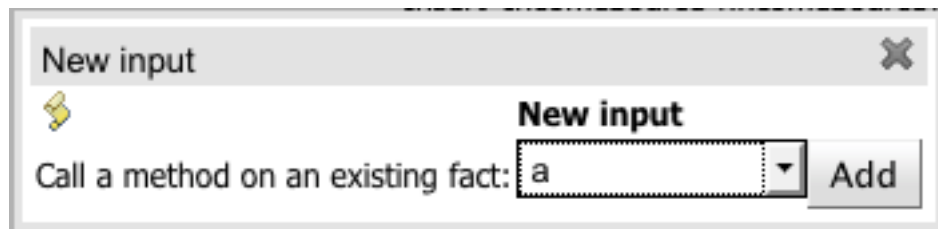


Figure 4.76. New Input popup

- Call method on an existing fact - Call a method from a fact in the beginning of the rule execution.

4.19.5. Audit Log

Run scenario

Results: 100 %

Summary: [app] field [approved] was [false].

Audit log:

OBJECT ASSERTED value:Applicant(age=0, applicationDate=null, creditRating=OK, name=null, approved=false) factId: 1

OBJECT ASSERTED value:LoanApplication(amount=0, approved=false, deposit=0, approvedRate=0, lengthYears=0, explanation=null, insuranceCost=0) factId: 2

OBJECT ASSERTED value:IncomeSource(amount=0, type=null) factId: 3

FIRING rule: [No bad credit checks] activationId:No bad credit checks [1, 2] declarations: app=LoanApplication(amount=0, approved=false, deposit=0, approvedRate=0, lengthYears=0, explanation=null, insuranceCost=0)(2)

OBJECT RETRACTED value:LoanApplication(amount=0, approved=false, deposit=0, approvedRate=0, lengthYears=0, explanation=Only AA, insuranceCost=0) factId: 2

FIRING rule: [A] activationId:A [1] declarations:

OBJECT ASSERTED value:IncomeSource(amount=1, type=null) factId: 4

FIRING rule: [RegexDslRule] activationId:RegexDslRule [1] declarations: applicant=Applicant(age=0, applicationDate=null, creditRating=OK, name=null, approved=false)(1)

FIRING rule: [CreditApproval] activationId:CreditApproval [1] declarations: applicant=Applicant(age=0, applicationDate=null, creditRating=OK, name=null, approved=false)(1)

FIRING rule: [Dummy rule] activationId:Dummy rule [0] declarations:

+GIVEN

Insert 'LoanApplication' [app]

approved: false

Figure 4.77. Expanded Audit Log

Audit log shows what happened when the knowledge base was executed.

4.20. File

Files can be stored in Guvnor as assets.

You need to provide the asset name and file extension in the "New Asset" popup dialog.

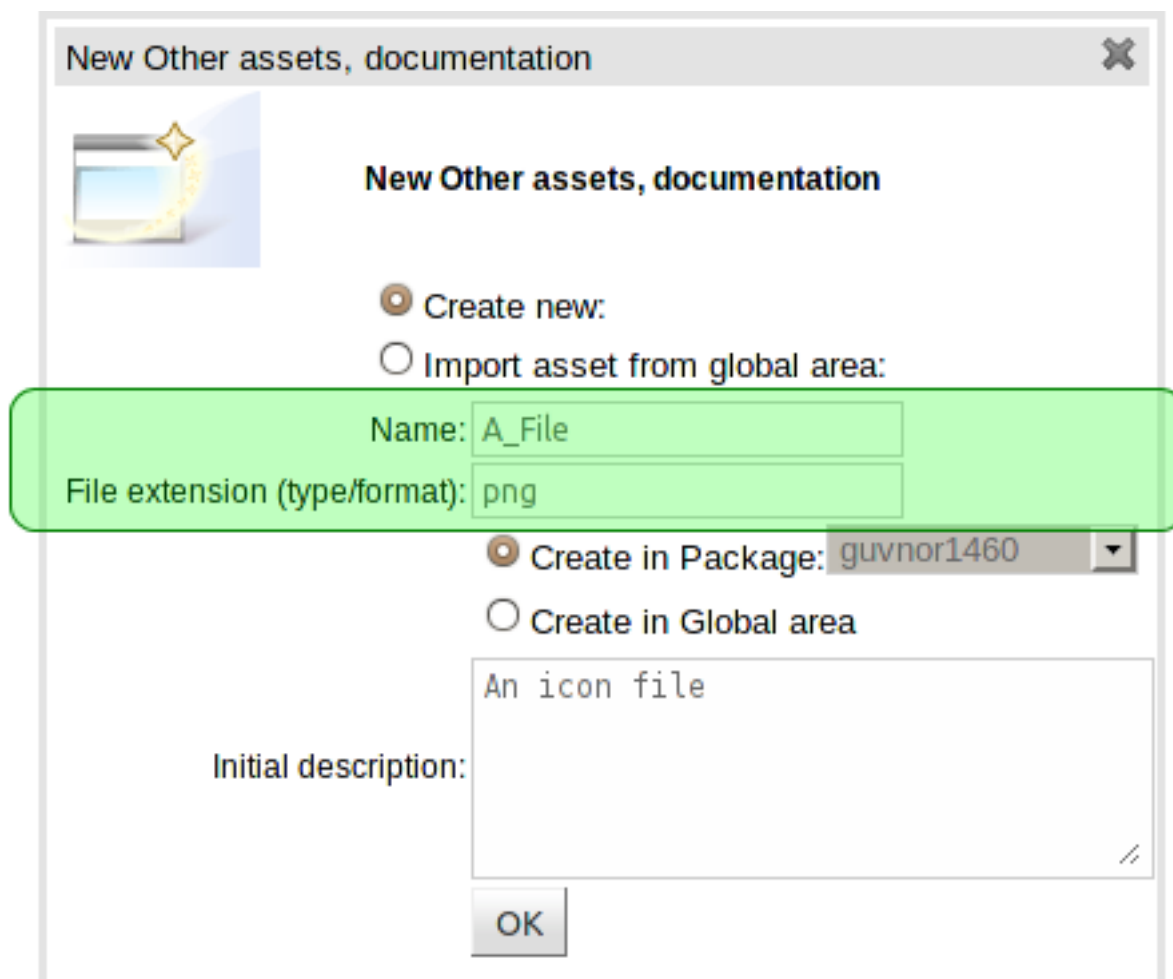


Figure 4.78. Creating a new file asset

Once you have created the asset a screen to upload (or download) the file itself is shown.

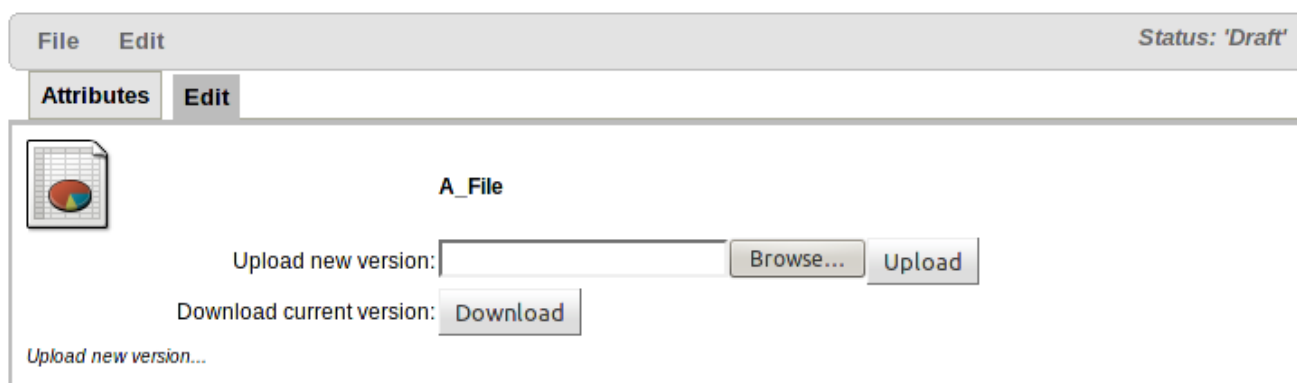


Figure 4.79. File upload/download screen

If you specify a file extension of "properties" a standard Java properties file can also be created.

The screenshot shows a dialog box titled "New Other assets, documentation". It has a close button in the top right corner. On the left is an icon of a folder with a star. The main title is "New Other assets, documentation". There are two radio buttons: "Create new:" (selected) and "Import asset from global area:". Below these are two text input fields: "Name:" with the value "A_Property_File" and "File extension (type/format):" with the value "properties". These two fields are highlighted with a green rounded rectangle. Below the extension field are two more radio buttons: "Create in Package:" (selected) and "Create in Global area:". The "Create in Package:" option has a dropdown menu showing "guvnor1460". Below these is a text area labeled "Initial description:" containing the text "A property file". At the bottom is an "OK" button.

Figure 4.80. Creating a new properties file asset

The properties name/value pairs can be maintained from within Guvnor.

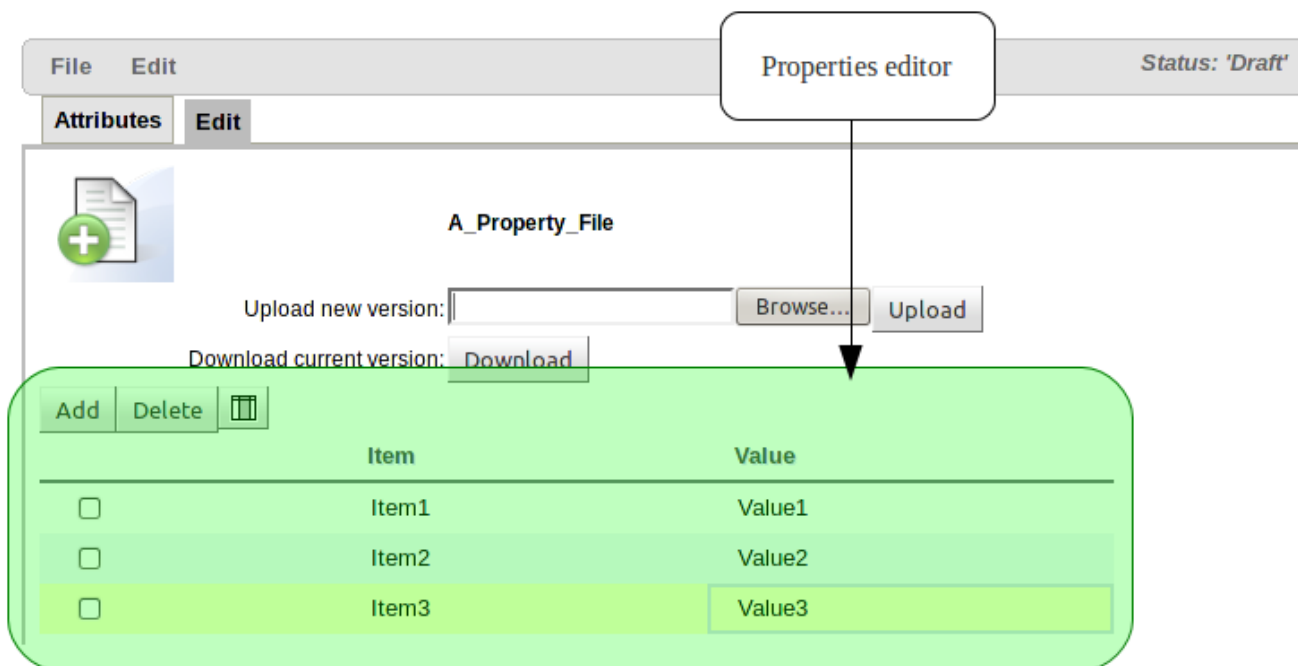


Figure 4.81. Maintaining a properties file

4.21. Change-Set

Change-set is one of the ways Drools has to group one or more Assets in a single Asset. Change-sets can then be added to a Knowledge Builder or a Knowledge Agent to process all the assets referenced by it.

By default, Guvnor provides a unique Change-Set per package containing the package binary or the drl source. The main problem with this is when you want to use just a sub-set of assets of a package (or even from different packages) in your application.

Using the Change-Set editor you can define your own change-sets inside Guvnor in any way you want. You could have a combination of different assets and packages, or even different snapshots of a package in a change-set. You can have all the change-sets per package you need, so if different applications are using the same package in different ways (different assets or assets' versions of a same package) you can make use of this feature of Guvnor.

4.21.1. Change-Set Editor UI

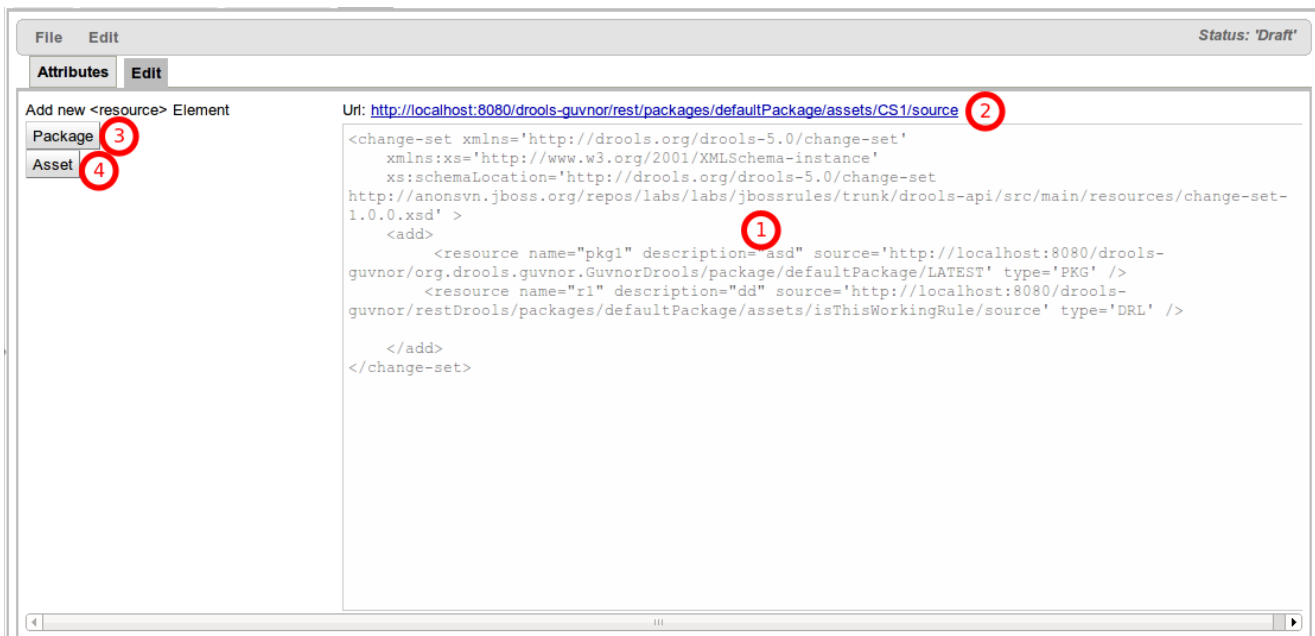


Figure 4.82. Change-Set Editor

To create a new Chang-Set you have to go to "Create New" -> "New Change Set" option in Knowledge Base section of Guvnor.

In the editor you have 4 items you can interact with:

1. Change-Set Editor canvas: Here you can edit your change-set by hand. You can add, remove or edit resources as you wish. Of course you will need to know the URL of each of the resources. You could get this information from the "Attributes" tab of each of the resources.
2. Change-Set URL: This is the URL you need to use in your applications to make reference to this change-set.
3. Package Button: This is an easy way to add packages to your change-set. You need to place the cursor where you want to add a new Package in the Change-Set Editor canvas and then click this button. A Wizard will appear to help you choose the right package. See next section for further information.
4. Asset Button: Useful button to add a new Assets to the Change-Set using a Wizard. For more information please read "Adding Assets using a Wizard" section.

4.21.2. Adding Packages using a Wizard

Clicking in the "Package" button of Change-Set Editor, a wizard will be displayed to help you choose the package you want to add as resource.

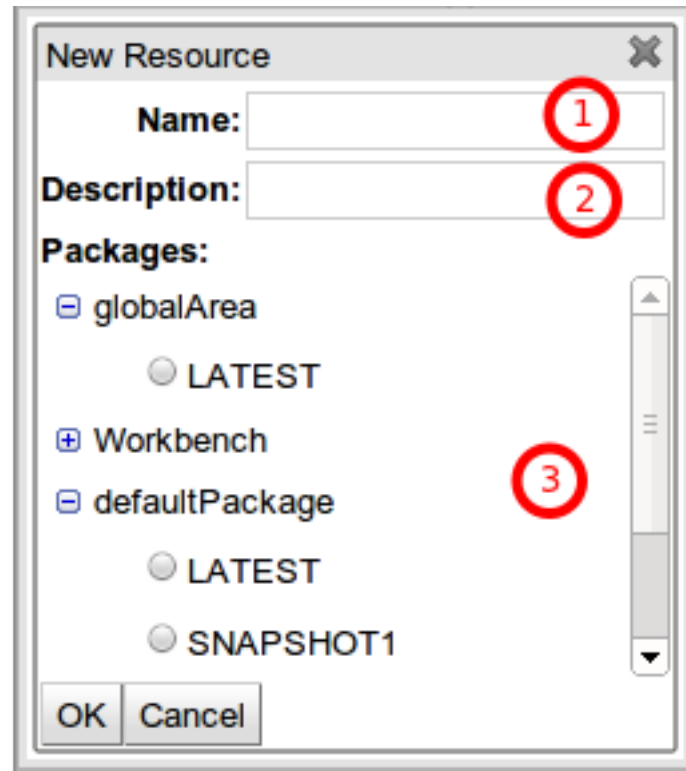


Figure 4.83. Adding a Package resource

The Package Wizard has 3 sections:

1. Name: Each resource inside a Change-Set could have a descriptive name. Use this optional field to set this property.
2. Description: Each resource inside a Change-Set could have a description. Use this optional field to set this property.
3. Packages: In this section you can select the package and the Snapshot version of a package you want to add as a resource of the Change-Set you are editing. If the Change-Set you are editing belongs to the Global Area, then you will be able to select the resource to add from all the packages defined in Guvnor and their snapshots of course. If the Change-Set you are editing belongs to a particular Package, you will only see the Snapshots of that package in this Wizard.

4.21.3. Adding Assets using a Wizard

You have 2 ways of adding assets as resources of a Change-Set: by hand using the Change-Set Editor canvas or using the Asset Wizard. This Wizard is invoked using the "Asset" button of the editor.

The 'New Resource' dialog box contains the following fields and sections:

- Name:** A text input field (highlighted with a red circle 1).
- Description:** A text input field (highlighted with a red circle 2).
- Package:** A dropdown menu showing 'package1' (highlighted with a red circle 3).
- Format:** A dropdown menu showing 'bpmn2' (highlighted with a red circle 4).
- Assets:** A table listing assets with columns: Format, Name, Status, Last modified, and Open. The table contains two rows:

Format	Name	Status	Last modified	Open
<input type="checkbox"/>	P2 asd	Draft	2011 Oct 12 16:48:58	<input type="button" value="Open"/>
<input type="checkbox"/>	Process1	Draft	2011 Oct 11 13:24:20	<input type="button" value="Open"/>

 The second row is highlighted with a red circle 5.

At the bottom of the dialog are 'OK' and 'Cancel' buttons.

Figure 4.84. Adding Assets resources

This Wizard has 5 sections:

1. **Name:** Each resource inside a Change-Set could have a descriptive name. Use this optional field to set this property.
2. **Description:** Each resource inside a Change-Set could have a description. Use this optional field to set this property.
3. **Package:** Use this field to filter the assets of a particular Package. If the Change-Set you are editing belongs to the Global Area, then you will be able to select any package, otherwise the package where your Change-Set belongs is automatically selected and you will not be able to change it.
4. **Format:** Use this field to filter the type of assets you are looking for.
5. **Assets:** The list of assets that match the selected package and format. You can select one or more assets using the check-box column and then click "OK" to add each resource to the Change-Set you are editing.

4.22. Services

Service Config is a special asset that defines an execution service configuration. This execution service is a war file (generated automatically by the editor) which you can deploy to execute KnowledgeBases remotely for any sort of client application. As this service is stateless, it is possible to have as many of these services deployed as you need to serve the client load.

This version of the execution server supports as many Knowledge Bases as you need, supporting also multiple stateless and stateful sessions in a native way.

To create a new Service you have to go to "Create New" -> "New Service Config" option in Knowledge Base section of Guvnor. In the following sections you have detailed information on how to configure your Service Config.

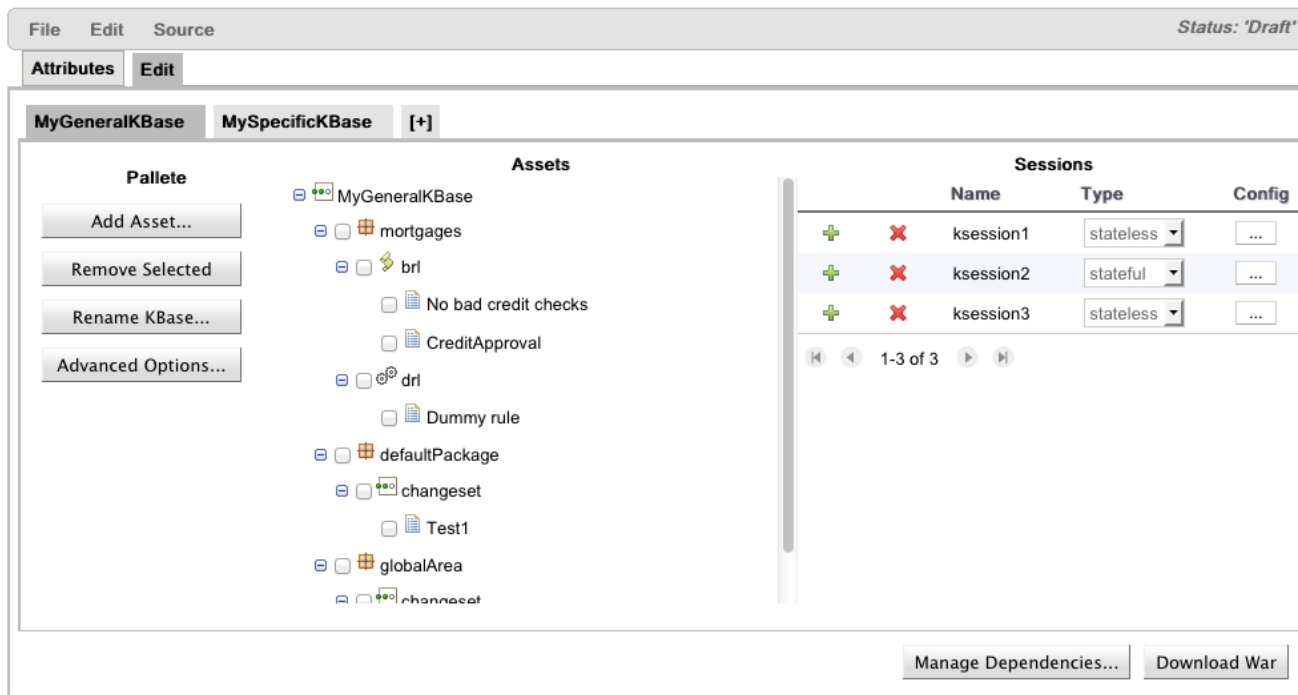


Figure 4.85. Service Configuration Editor

4.22.1. Configuring Multiple Knowledge Bases

As already mentioned the Service editor allows users configure as many knowledge bases as they need. This section describes how to use the tabbed panel to configure your Service with multiple knowledge bases.

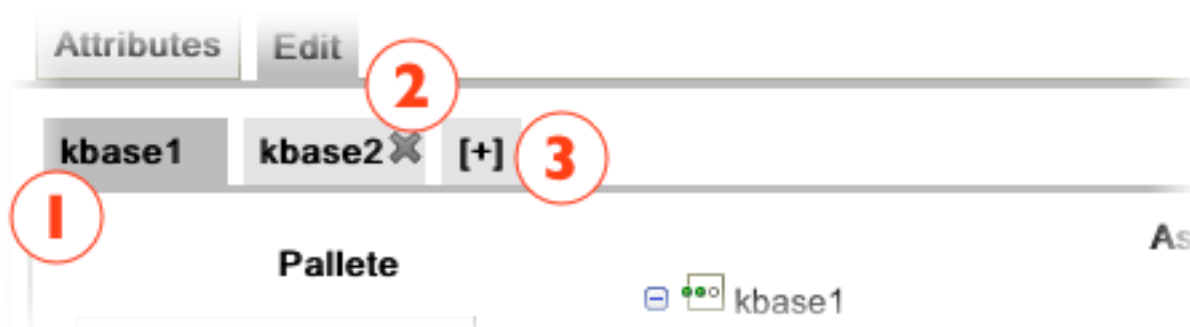


Figure 4.86. Tabbed Panel

The above panel is composed by the following:

1. Named Tab: each tab represents a knowledge base, and once selected, its configuration panel is displayed. See next section for further information.
2. Removing: in order to remove an existing tab/knowledge base, you have to click the highlighted "X" and then confirm the action. Important to note that should exist at least one knowledge base.
3. Add new: to add a new kbase, all you need is click on this special tab. The new kbase will be added on the left hand side of the plus tab.

4.22.2. Knowledge Base Configuration Panel

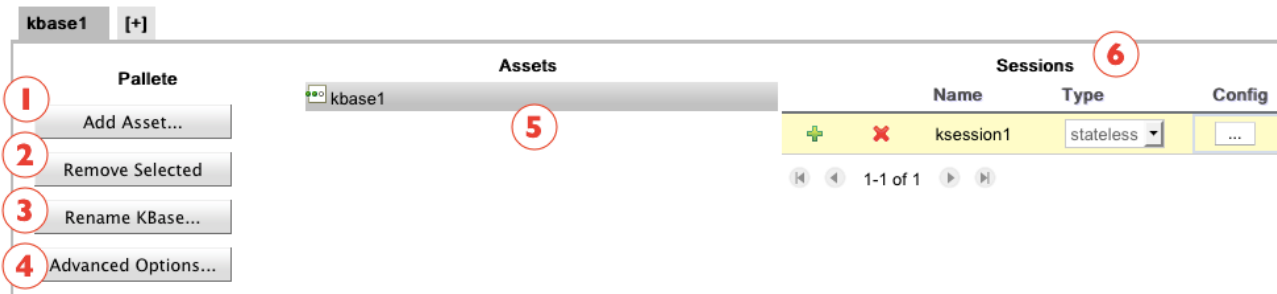
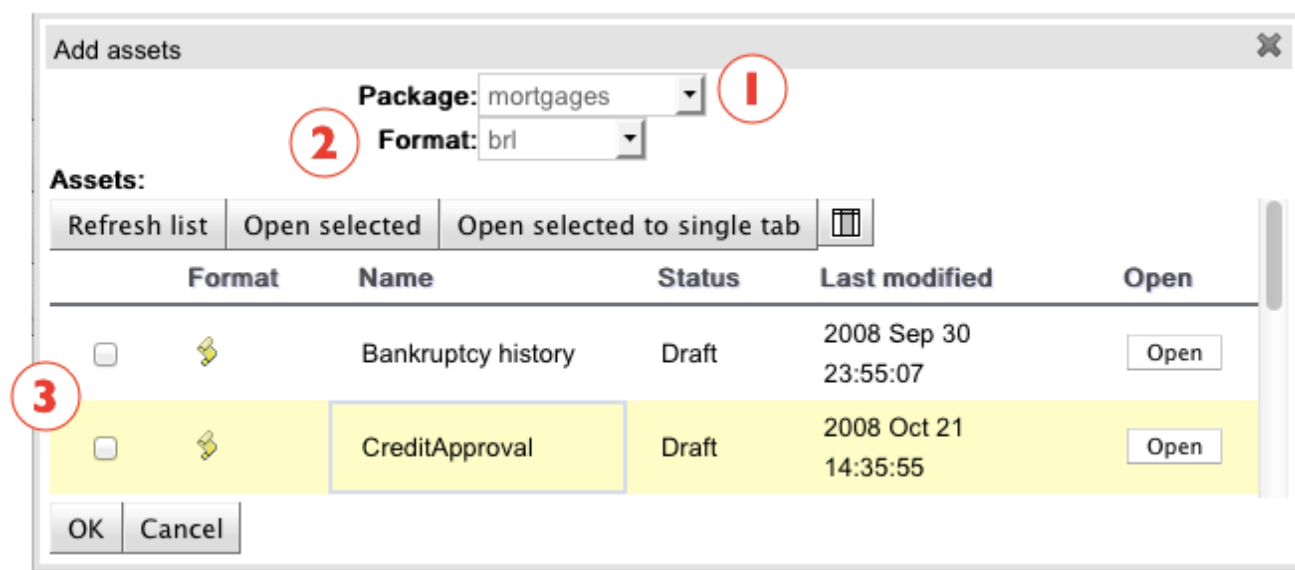


Figure 4.87. Knowledge Base Configuration

In the above configuration panel you have 6 items that you can interact with:

1. Add Asset Button: add new assets to the tree using a wizard. See next section for further information.
2. Remove Selected Button: remove the selected assets (the ones that are checked) from Assets Tree.
3. Rename KBase Button: this button will open a standard "prompt for rename" dialog box.
4. Advanced Options Button: opens the advanced knowledge base configuration dialog. Configurations that are not available in the panel, should be available in this advanced dialog. Detailed information about it can be found in the "Advanced Knowledge Base Configuration Options" section.
5. Assets Tree: Here you have the complete set of assets, grouped by package and type, that are referenced by the active knowledge base configuration. For additional information, please read the "Assets Tree" section.
6. Sessions Grid: Here users can create, delete and edit in place a list of desires ksessions. For detailed instructions on how to configure ksessions, please check the "Configuring Knowledge Sessions" section.

4.22.2.1. Adding Assets using Wizard

**Figure 4.88. Adding Assets resources**

This Wizard has 3 sections:

1. Package: Use this field to filter the assets of a particular Package. By default the selected package is the one that your Service belongs to, but you're not limited to it as you can select any other package that you want.
2. Format: Use this field to filter the type of assets you are looking for.
3. Assets: The list of assets that match the selected package and format. You can select one or more assets using the check-box column and then click "OK" to add each resource to the Assets Tree of your active knowledge base configuration.

4.22.2.2. Advanced Knowledge Base Configuration Options

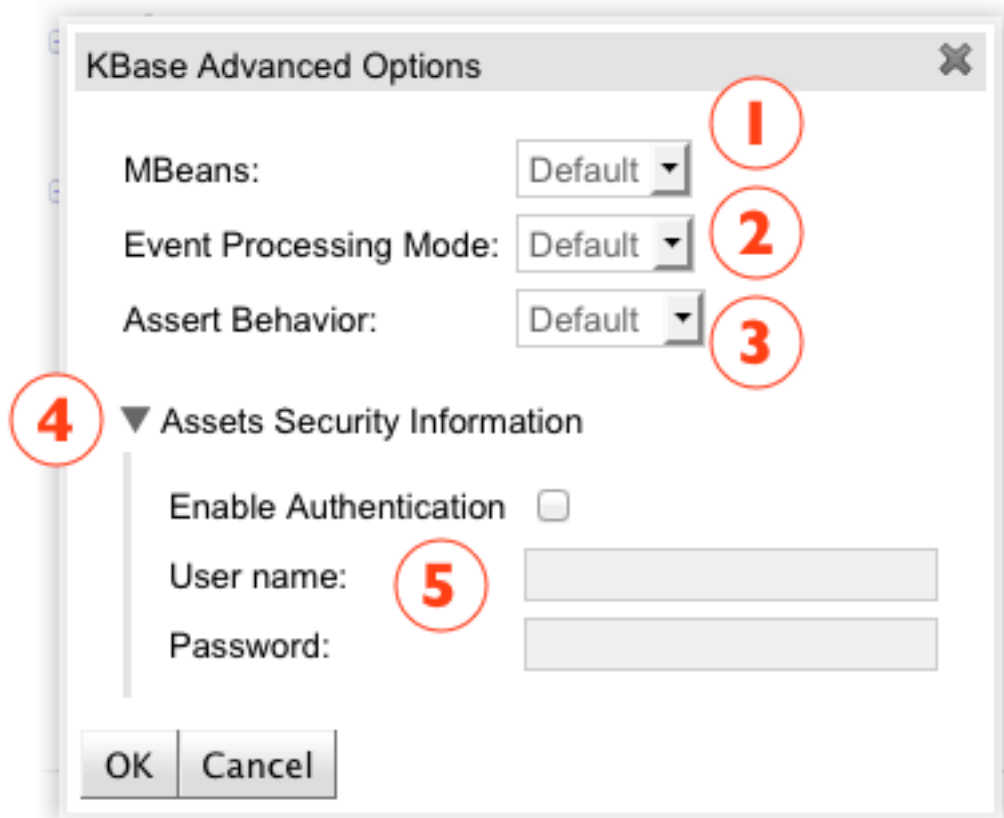


Figure 4.89. Advanced Options Dialog

This dialog has the following sections:

1. MBeans: Can enable or disable MBeans on active knowledge base. Users can also keep the default engine behavior by choosing the Default option.
2. Event Processing Mode: Defines the event processing mode, which options are Cloud or Stream. Users can also keep the default engine behavior by choosing the Default option. For more information about Event Processing Mode, consult the Drools Fusion documentation.
3. Assert Behavior: Defines the expected assert behavior, which options are Identity and Equality. Users can also keep the default engine behavior by choosing the Default option.
4. Assets Security Information: This section holds any security related configuration to remotely access the guvnor repository.
5. Enable Authentication, User name and Password: If explicit authentication is enabled, you should provide the user name and password for remote access.

4.22.2.3. Assets Tree

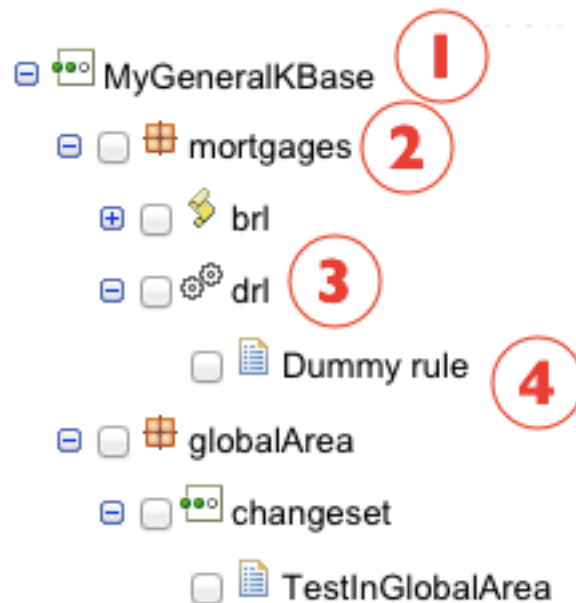


Figure 4.90. The Assets Tree

The assets tree is composed by 4 different types of nodes:

1. Knowledge Base: Root node that represents the active kbase.
2. Packages: Selected assets are grouped by its packages. Important to note that those packages are just informative and are not relevant for the service runtime.
3. Asset Types: Groups assets of the same package by type. The actual list of supported types are: brl, drl, dsl, bpmn2, gdst, changeset and jars.
4. Assets: Selected assets of a given type (parent) and package (grandparent).

4.22.2.4. Configuring Knowledge Sessions

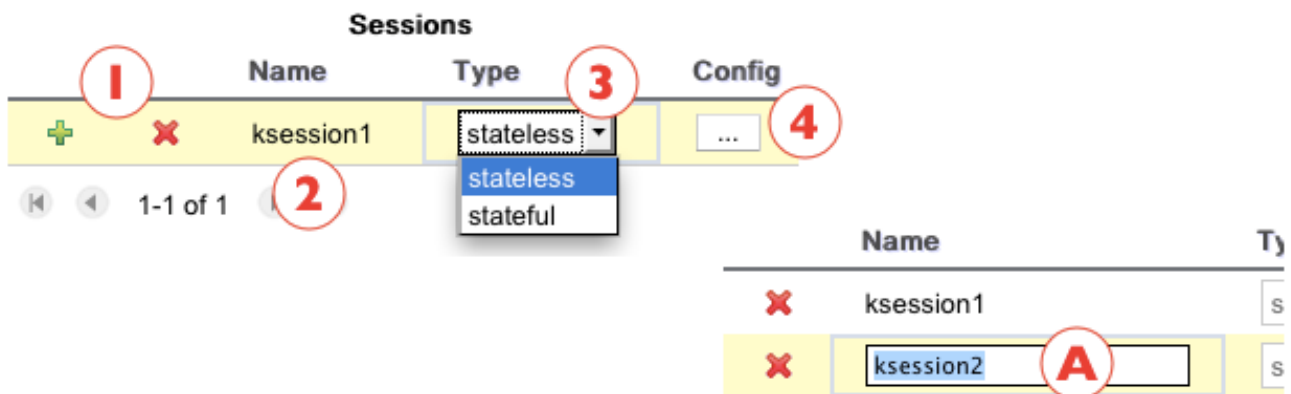


Figure 4.91. Knowledge Sessions Grid - basic information to define a ksession are available to be edited in place.

The ksessions grid is composed by the following columns:

1. Add and Remove Icons: Users can use these icons to add new or remove existing knowledge sessions. Important to note that should exists at least one knowledge session.
2. Knowledge Session Name: Defines the ksession name. Users can edit this particular field in place as highlited on box "A".
3. Session Type: Defines if a ksession is Stateless or Stateful. Users can also select the desired type directly on the grid.
4. Configuration Button: This button opens the knowledge session configuration dialog. See next section for further information.

4.22.2.4.1. Knowledge Session Configuration Options

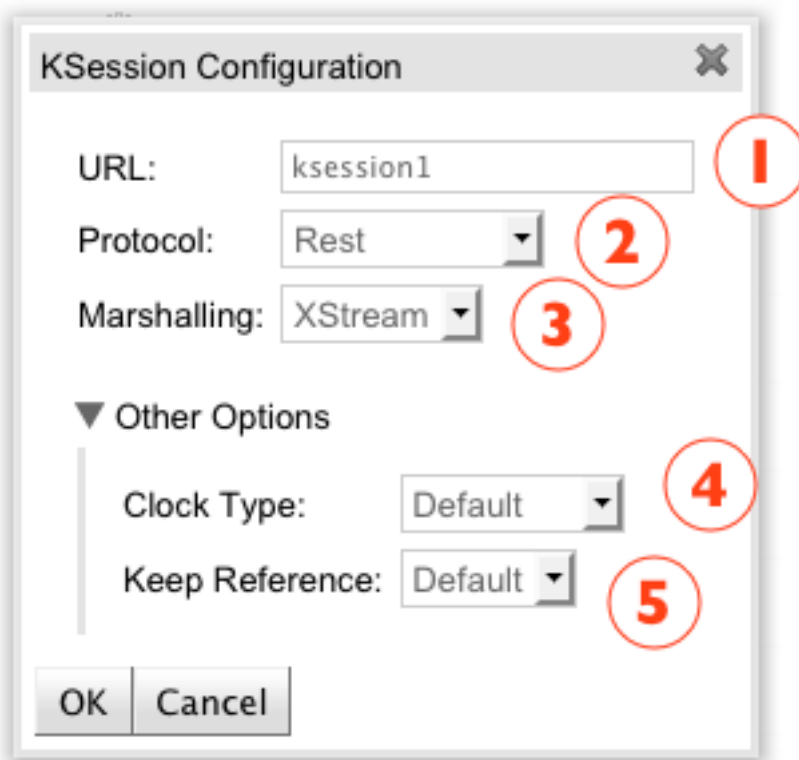


Figure 4.92. Knowledge Session Configuration Dialog

This ksession configuration dialog has the following 5 fields:

1. URL: Defines a custom URL to access the active ksession.
2. Protocol: Defines the protocol that ksession should be accessed by, default options is REST.
3. Marshalling: Defines which marshalling technology should be used, default option is XStream.
4. Clock Type: Defines the type of the session clock, which options are Pseudo or Real Time. Users can also keep the default engine behavior by choosing the Default option.
5. Keep Reference: Enable or disable the engine to keep a weak reference of this section on the kbase. Users can also use the default engine behavior by choosing the Default option.

4.22.3. Downloading War & Managing Dependencies

After configuring your knowledge bases and sessions, you're ready to download the service war. For so, all you need to do is click on the "Download War" button and the following screen (or a similar one, depending on your browser) will show up.

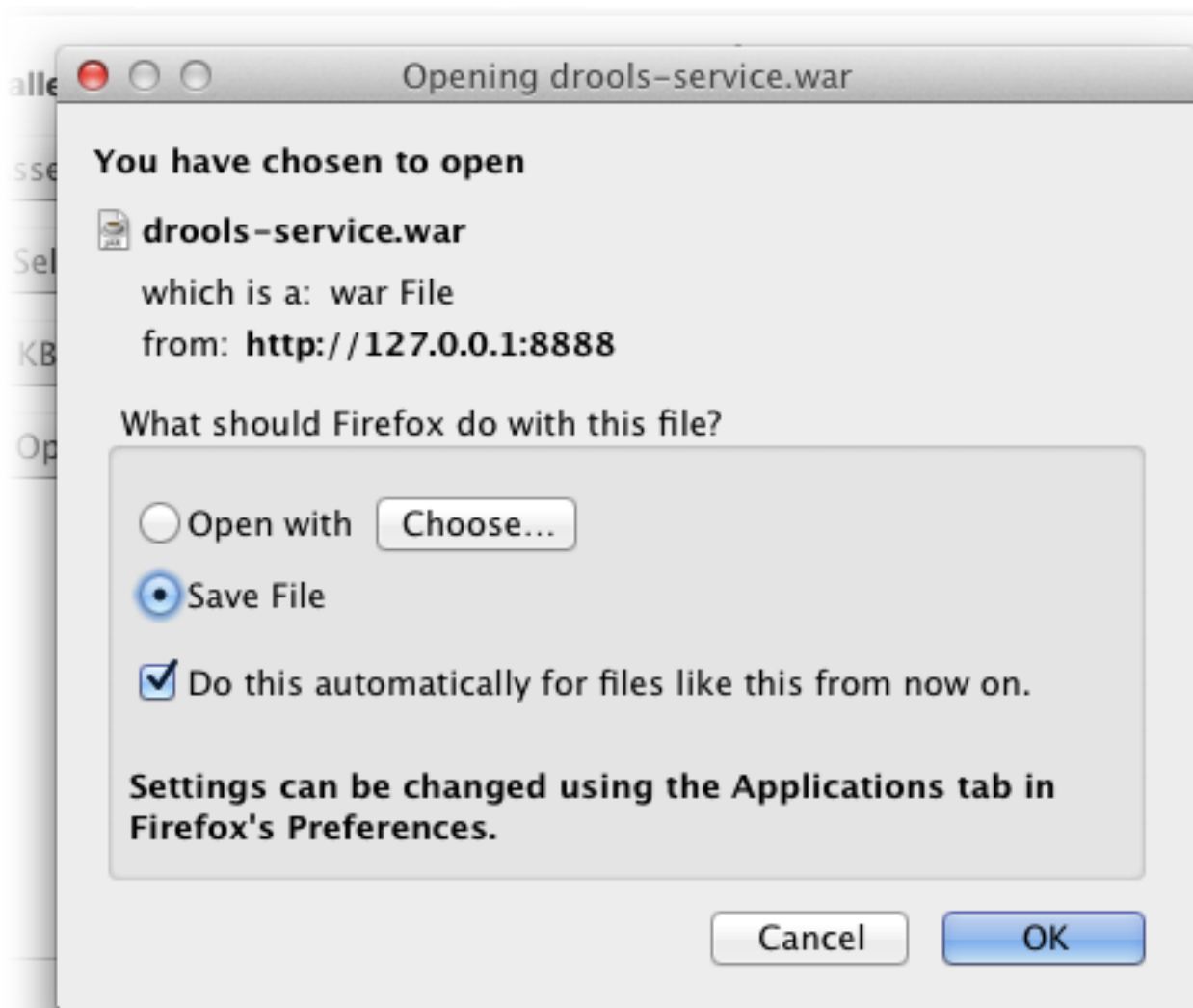


Figure 4.93. Downloading WAR



Warning

Before click on the "Download War" button, makes sure that you don't have pending changes. This is a very important step, once the the war file is generated based on the latest saved information and not on the current screen state.

The generated file is a ready to be deployed war file. Which mean that it contains all the necessary configuration files and all the needed dependencies as well. As dependency management is a crucial subject in a real world applications deployment, we will discuss that in more detail in the next section.

4.22.3.1. Dependency Management

As already mentioned, the generated war file contains all the necessary dependencies to execute the configured Service, it works seamlessly on most standard containers.

But the reality is a bit different, rarely you'll have a "standard container", usually what you have is an already configured and running container. Probably this container even have already configured a set of shared libs between your deployed application. And, of course, you don't want those shared libs on your wars.

Based on real world scenarios, we made available a simple dependency management tool, that enables you explore and configure all the dependencies contained in the generated war file. This tool can be accessed just by clicking the "Manage Dependencies" button, and the following dialog will be displayed.

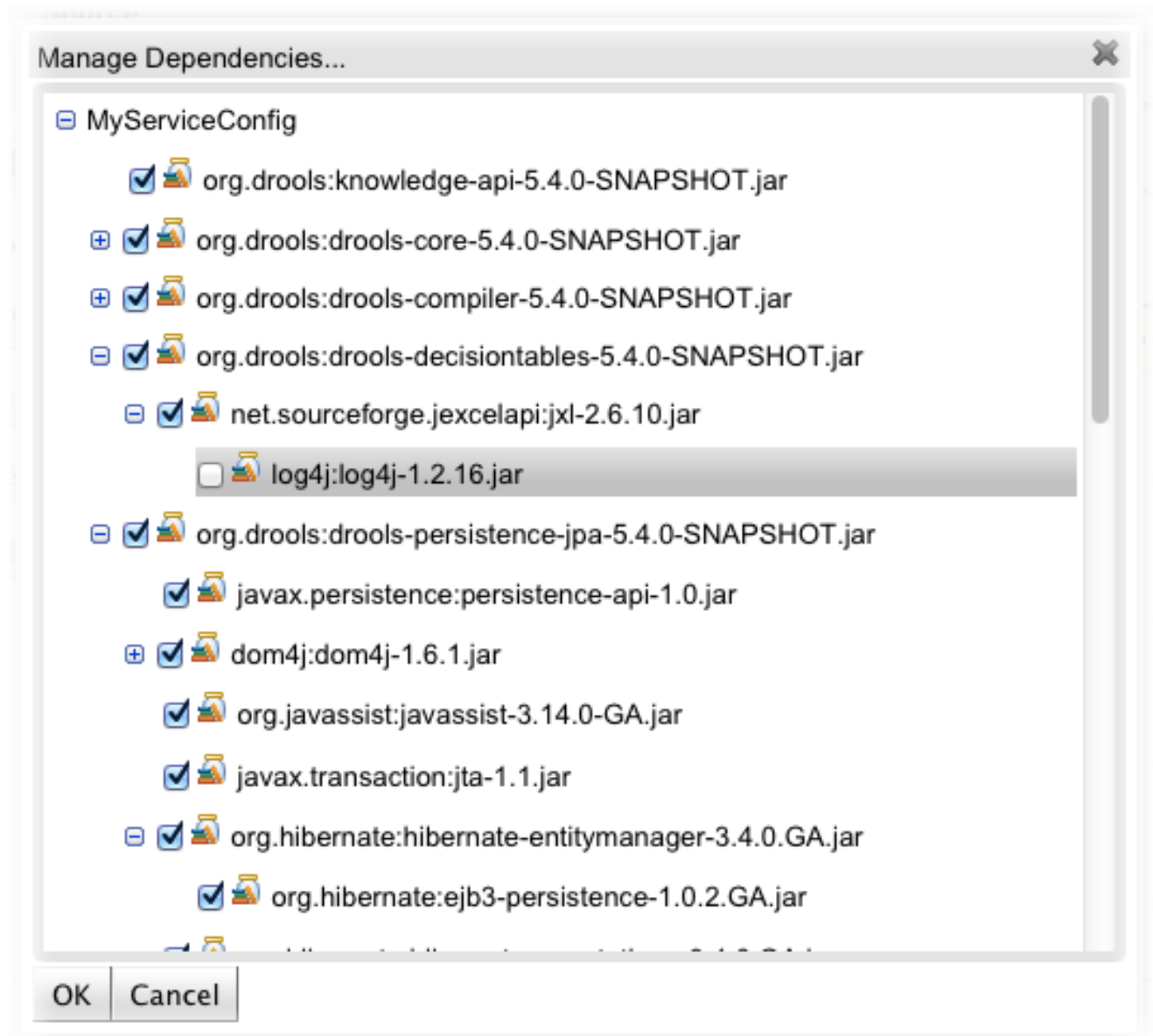


Figure 4.94. Dependency Management Dialog

This dialog presents the complete dependency tree of the service. To disable any particular dependency, all you need to do is "uncheck" it and press the "OK" button.

Important to note that after configuring your dependencies you have to save the service configuration, in order to have the generated war file reflecting your configuration.

Chapter 5. Managing Assets

5.1. Navigating and finding rules

The two main ways of viewing the repository are by using the Package Explorer or user-driven Categorization (tagging) as outlined previously.

5.1.1. Package explorer

The Package Explorer appears in the "Knowledge Bases" section of the left-hand side navigation bar. The Package Explorer shows a list of all packages (other than those that have been archived).

The list of packages can either be viewed hierarchically or flat and fully expanded or collapsed using the icons below the tree-view.

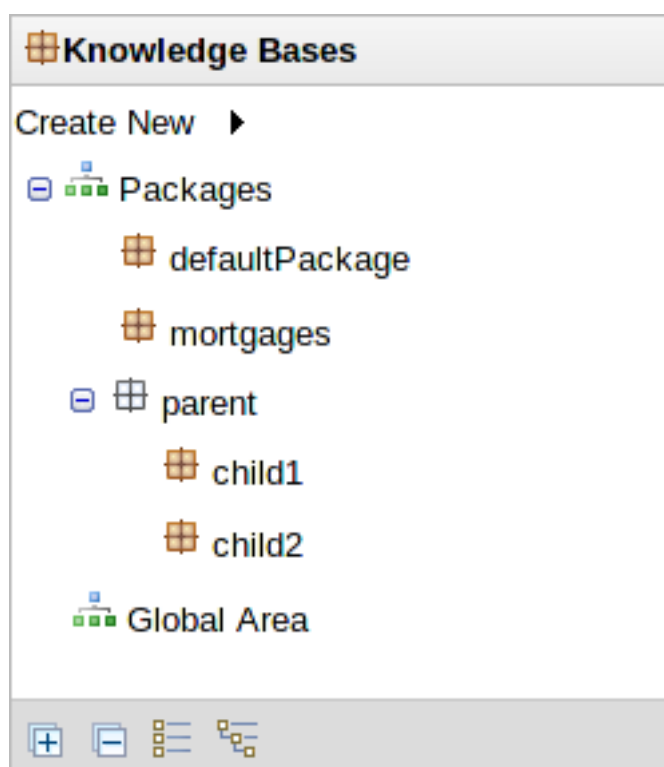


Figure 5.1. Hierarchical view

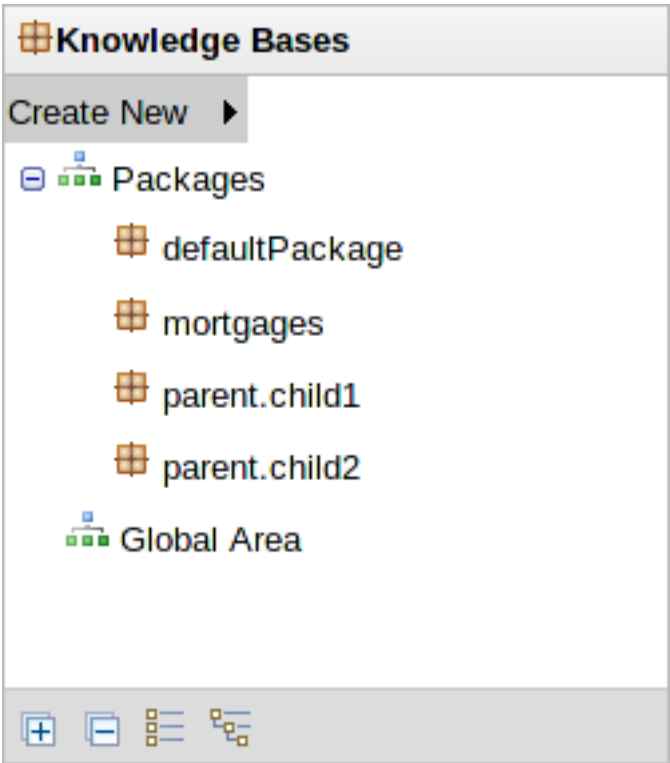


Figure 5.2. Flat view

Clicking on a package name in the tree-view will launch the Asset Viewer.

5.1.1.1. Asset Viewer

The Asset Viewer is launched when a package in the Package Explorer is clicked.

The Asset Viewer shows a list of assets in the package grouped by their type. If no assets exist in the package a warning is displayed.

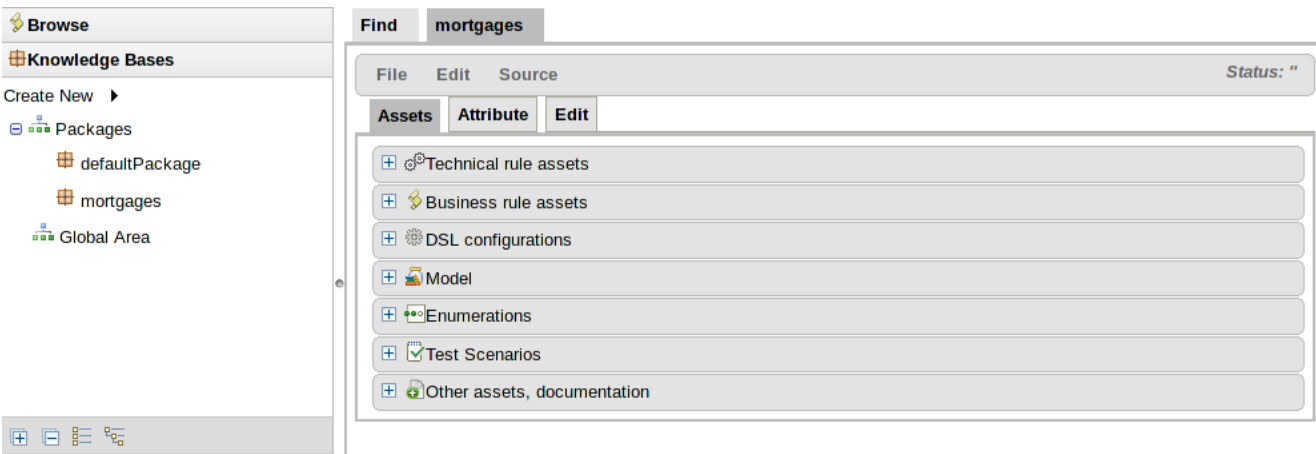


Figure 5.3. A package with assets



Figure 5.4. A package without assets

Expanding a section using the [+] icon reveals the assets contained therein.

Assets are listed in a table that can be paged forwards or backwards and can be opened by either clicking their respective "Open" button, or selecting the required rows and clicking the "Open selected" button. Multiple assets can be opened into a single editor by selecting the required rows and clicking the "Open selected to single tab" button.



Figure 5.5. Showing individual assets

5.1.1.2. Package configuration

A package can be configured by clicking on the "Edit" tab on the Asset Viewer.

The facilities here relate to compiling the package into a binary form. The "Packaging" section of this user-guide provides a detailed narrative on this section.

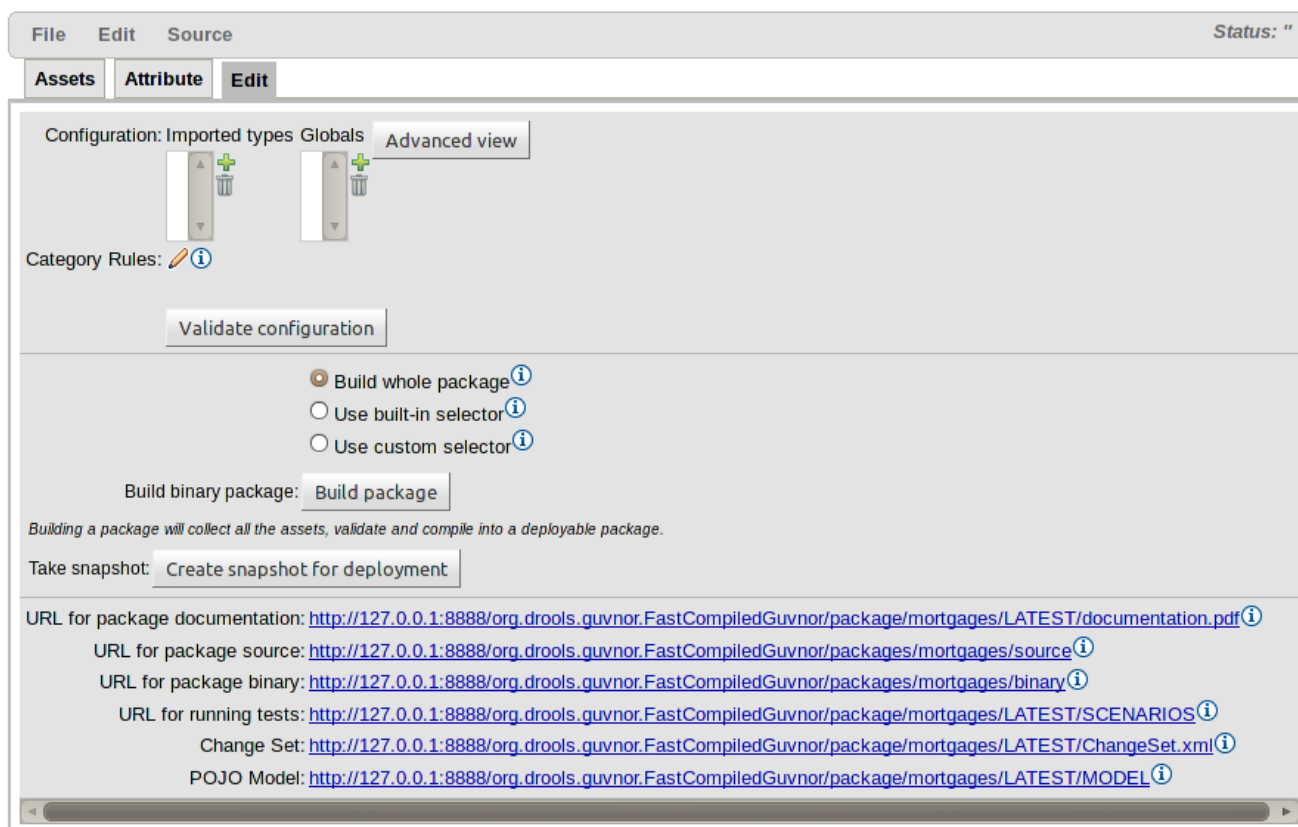


Figure 5.6. Package configuration

5.1.2. Category view

The category view provides a way to navigate your rules in a way that makes sense to your organization.



Figure 5.7. Category view

The above diagram shows categories in action. Generally under each category you should have no more than a few dozen rules, if possible.

5.1.3. Inbox and comments

Built into the Guvnor are two useful features to helping manage changes: an Inbox, and a comments section. These features do not affect any execution or access to rules, but are purely for documentation and notification purposes, and are of course always optional to use.

5.1.3.1. Inbox

In the "browse" section of the application, there is an "Inbox" tree item, below this are 3 inboxes. "Incoming changes" contains changes to any artifacts that the current logged in user has edited in the past, or commented on. Simply editing or commenting on an artifact registers interest in it to be notified of changes the next time you log in. "Recently opened" contains items that have been

recently opened (simply opening an artifact will make it appear here, the last 100 recently opened items will appear here). "Recently edited" contains the last 100 recently edited items (artifacts that the current user has made changes to).

5.1.3.2. Comments

Below the documentation box of each artifact, is a "comments" section - simply, you can add a new comment. Administrators can clear all comments on a given artifact, but other users can only append comments. Each comment records what user made the comment, and when. Users who can't edit artifacts can still comment on them.

5.2. Feed

Feeds make it easier for users to keep track of Guvnor content. Following Guvnor content are published in Atom Feed:

- Package version history
- Asset version history
- Asset Discussion
- Categories
- Package assets

Chapter 6. Quality Assurance

6.1. Test scenarios





Scenarios for package:mortgages					
 <input type="button" value="Run all scenarios"/>					
<input type="button" value="Refresh list"/> <input type="button" value="Open selected"/> <input type="button" value="Open selected to single tab"/> 					
	Format	Name	Status	Last modified	Open
<input type="checkbox"/>		Are they old enough <i>Test really basic validation, common sense stuff</i>	Draft	2011 May 2 12:23:39	<input type="button" value="Open"/>
<input type="checkbox"/>		Good credit history only <i>Obviously we only want people who can pay stuff back</i>	Draft	2011 May 2 12:23:39	<input type="button" value="Open"/>
<input type="checkbox"/>		NINJAs	Draft	2011 May 2 12:23:39	<input type="button" value="Open"/>
<input type="checkbox"/>		No bankruptcies <i>We don't want bankrupt people since 1990, or if their amount was over a certa...</i>	Draft	2011 May 2 12:23:39	<input type="button" value="Open"/>
<input type="checkbox"/>		Pricing low end	Draft	2011 May 2 12:23:39	<input type="button" value="Open"/>
  1-5 of 5  					

Figure 6.1. Test Scenarios before execution

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

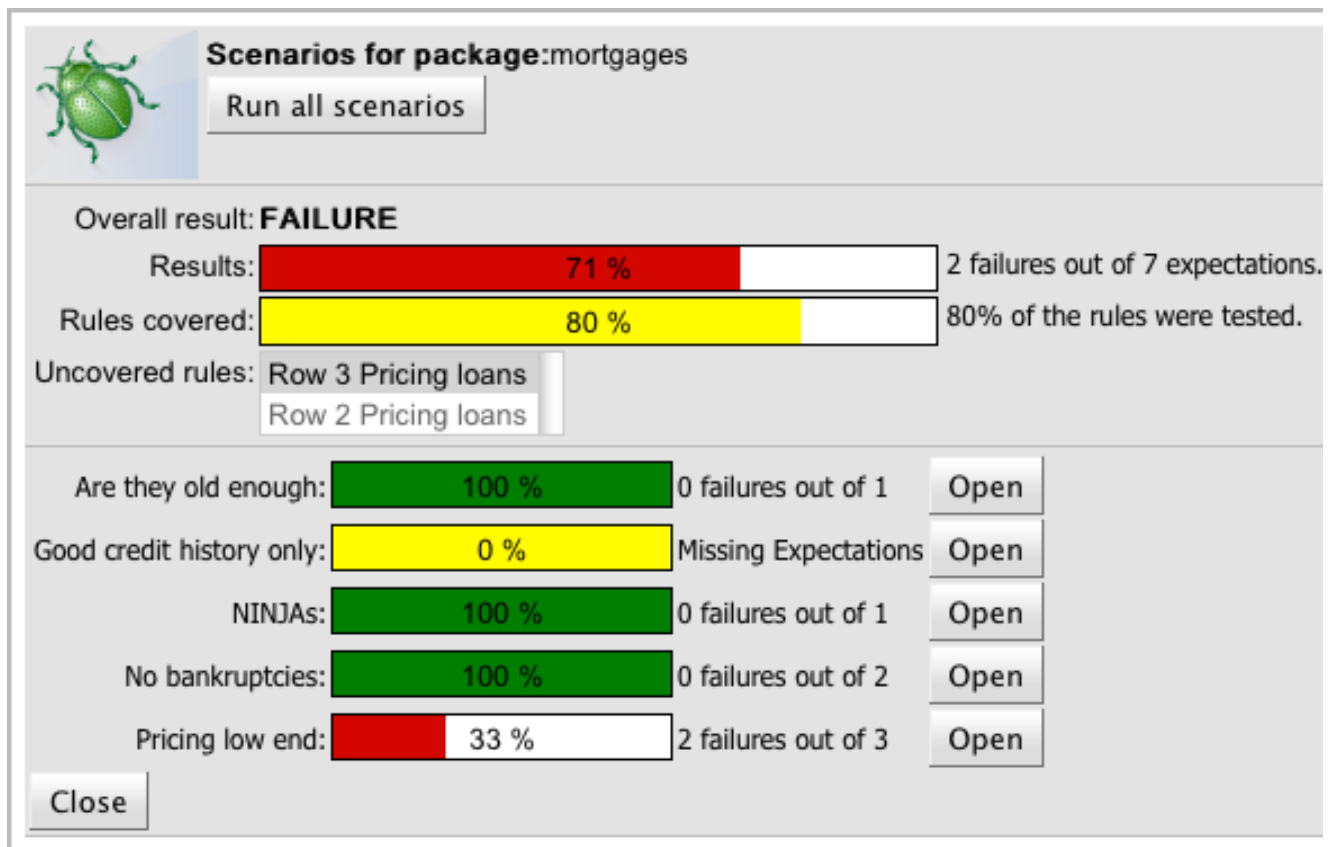


Figure 6.2. Example Test Scenarios after execution

After running all the Test Scenarios a report is shown. In a valid knowledge base all tests should pass and rule coverage should be 100%. Passing tests are marked with green, red shows failing tests and yellow notifies about missing rule coverage or missing expectations.

6.2. Package analysis

Package analysis uses Drools rules to statically analyse knowledge modules. It produces a report that contains information about the quality of the knowledge module.

Package analysis message types:

- Notes - Needs attention, may be ignored.
- Warnings- Possible problem, usually there is an alternative way.
- Errors - Needs correction, logical fallacy etc.

6.2.1. Items that the analyser is searching for

- Range Validation - Making sure all the ranges for a field are covered. For example: Person's age is greater than 20 restriction exists. Check that the person's age is 20 or less than 20 is missing.

- Missing Equality - When there are inequalities, but not equality being catered for. For example: LoanApplication has a field called approved. Approved field is checked to not equal to true (`LoanApplication.approved != true`), but a check where approved is equal to true is missing.
- Reduncancy - Rules or restrictions inside a rule are redundant. Reduncancy in rule: Two rules fire with the same conditions and execute the same actions. Restriction redundancy example: Person has two restrictions for name, both are checking if the name is "Toni".
- Subsumption - Rule subsumption: Both rules have the same actions. There exists one set of facts that can satisfy both of the rules and another set of facts that only satisfies the other rule. Restriction subsumption: Person fact has two restrictions, one checks that the age is greater than 20 and another checks that the age is greater than 50. Age greater than 50 restriction is subsumptant and can be removed.
- Rule Incoherence - Nothing can meet the rule conditions. Example: Rule author has added two restrictions on one field that are incoherent. Person age is equal to 20 and age is equal to 30. The rule can never be satisfied.
- Excessive Use of eval()- Eval should be avoided, eval is slow and doesn't make use of the RETE-network. If the rule consists mainly of eval statements then the user should be warned.
- Rule Optimisation - The most strict restrictions should be declared first. Slower restrictions like eval last.
- Rule Should Have an Action - If the rule does not have an action it does nothing.
- Rules and Patterns That Always Pass - Pattern example: Persons age field has restrictions that check that the age is greater or equal to 20 and a restriction that checks that the age is less than 20. All the ranges for age are checked in one pattern.

Chapter 7. Packaging

7.1. Packaging

Packaging is the process of assembling required assets into a single deployable unit, called a package, and configuring the package as necessary.

Some aspects of the configuration are also critical for authoring assets, such as the import of model classes and the definition of global variables. For example, you may add a model which has a class called `com.something.Hello`, you would then add `import com.something.Hello` in your package configuration and save the change.

The package configuration screen is reached by clicking on the required package in the "Package Explorer" and then selecting the "Edit" tab.

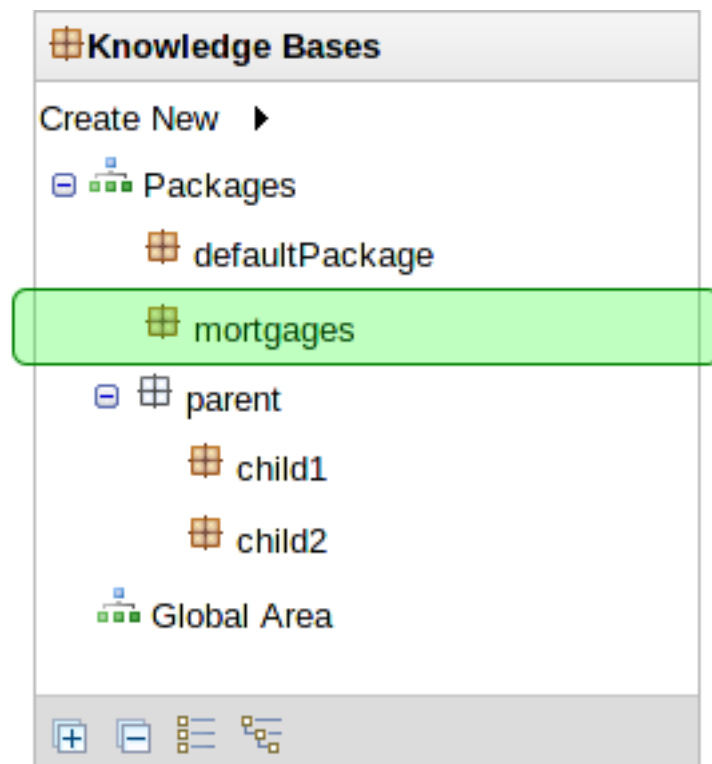


Figure 7.1. Select a package from the "Package Explorer"

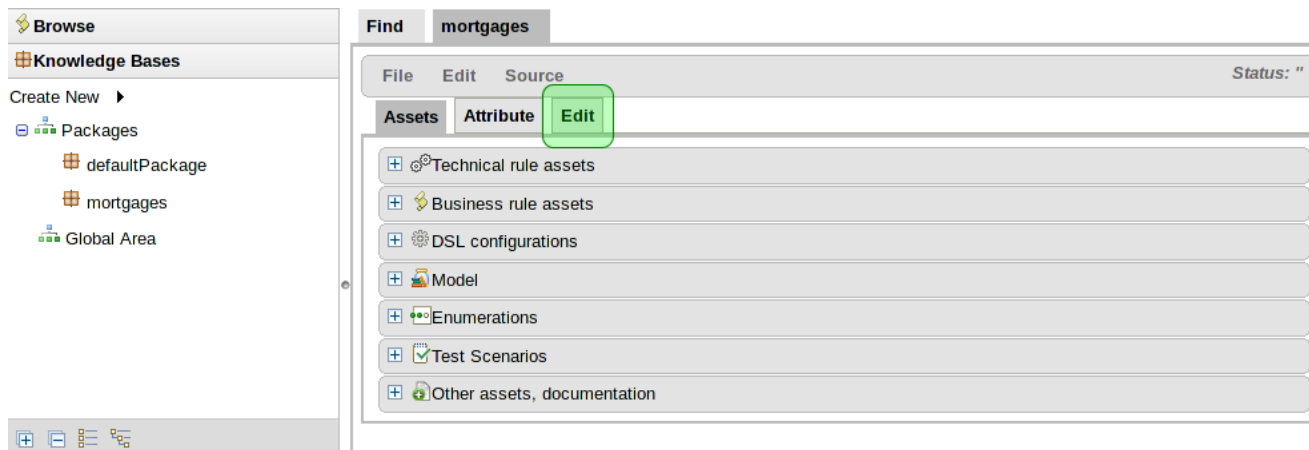


Figure 7.2. Select the "Edit" tab

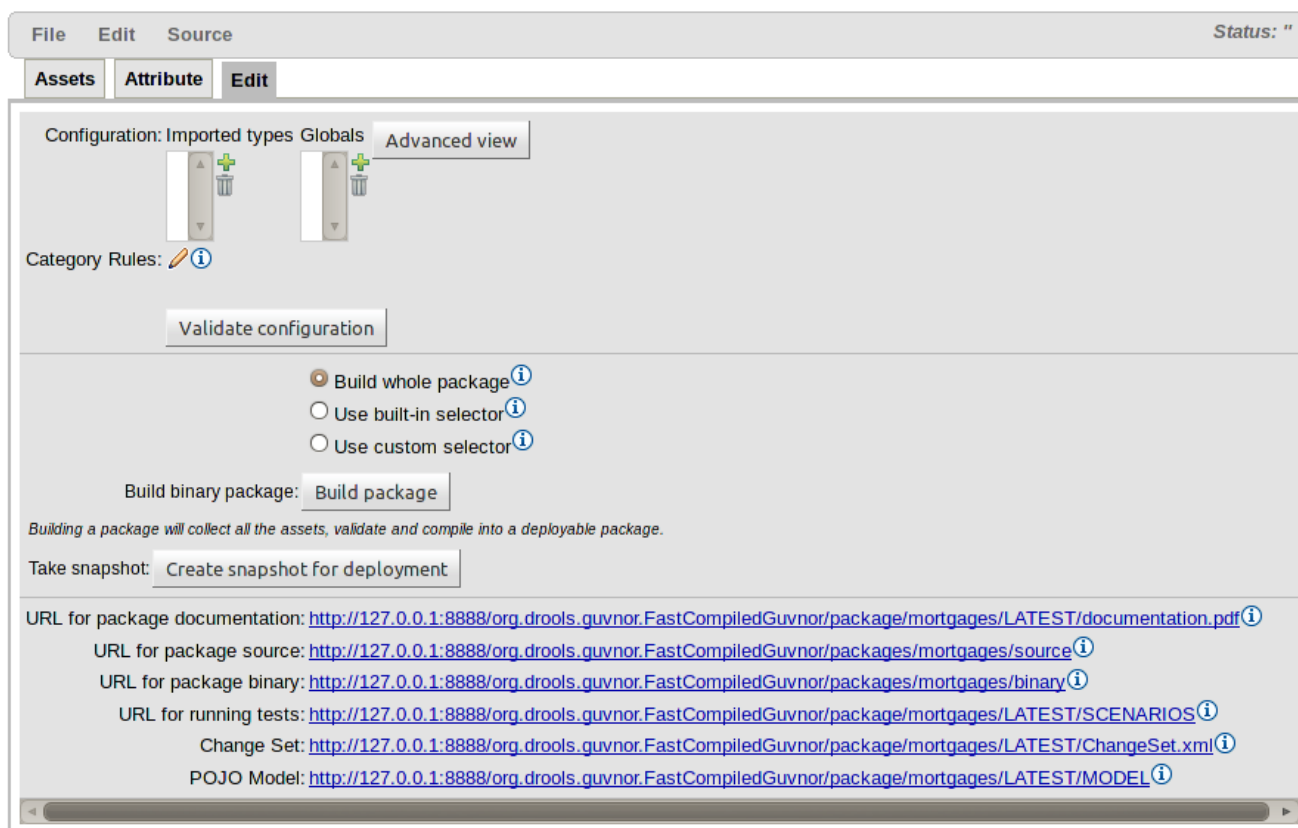


Figure 7.3. The Package Configuration screen

7.2. Imports and Globals

The "imports" section of the package configuration screen allows you to import Java classes needed by your assets.

When a POJO Model file is uploaded it is scanned for classes and imports are automatically created.

Globals are DRL Global variables that can be accessed by rules. Please consult the Drools Expert documentation for details.



Figure 7.4. Imports and Globals configuration

7.2.1. Basic View

The "Basic View" of Imports and Globals is shown by default. This view allows either to be managed visually.

Imports or Globals can be inserted by clicking the [+] icon beside the applicable panel. Either can also be deleted by selecting the appropriate Import or Global and clicking the dustbin icon.

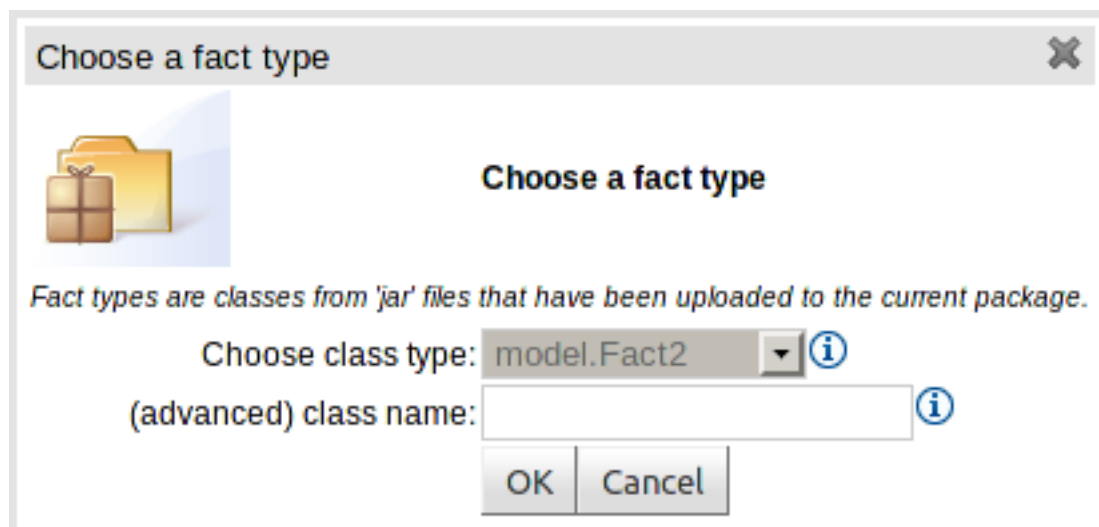


Figure 7.5. Adding an Import with the "Basic View"



Figure 7.6. Adding a Global with the "Basic View"

7.2.2. Advanced View

You can switch to the "Advanced View" by clicking the "Advanced View" button. This view provides a simple text-box in which package-level DRL can be entered.

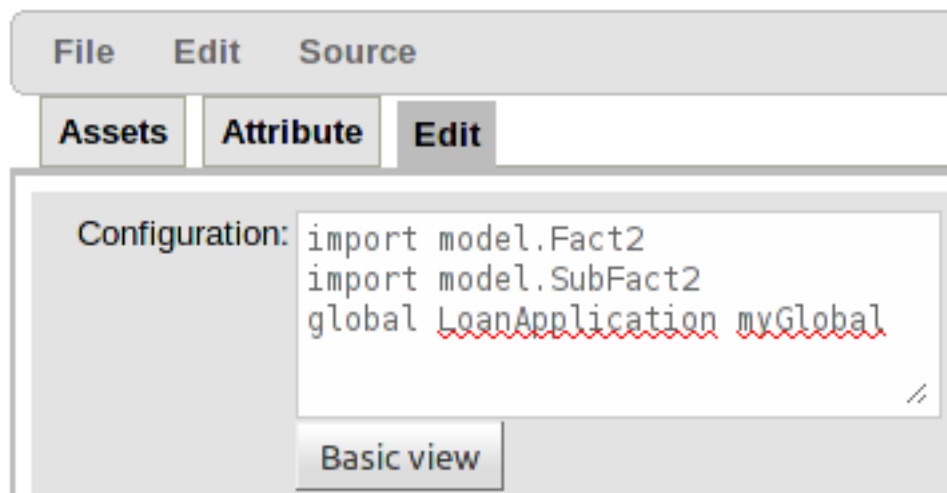


Figure 7.7. The Advanced View

7.3. Category rules

TODO

7.4. Building

TODO

Finally you would "build" a package. Any errors caught are then shown at this point. If the build was successful, then you will have the option to create a snapshot for deployment. You can also view the DRL that this package results in.

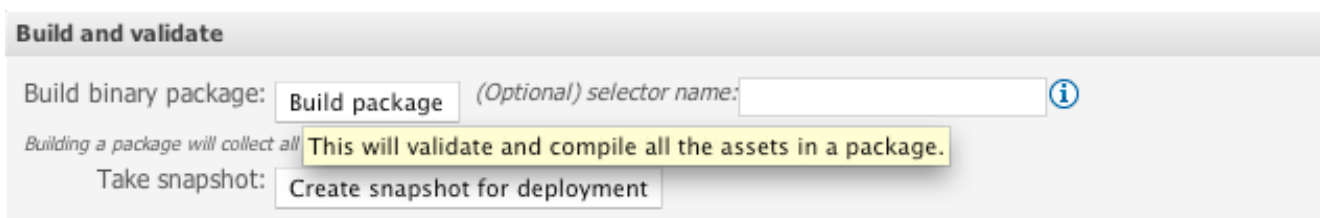


Figure 7.8. Package building



Warning

In cases of large numbers of rules, all these operations can take some time.

7.5. Selectors

When building packages using the "Packages" feature you have the option to use a "selector". This selector will filter the list of rules that are built into the package.

7.5.1. Built-in Selector

Guvnor provides several built-in selectors which allow you to choose what assets form part of a package build according to asset's status and category.

The screenshot shows the 'Built-in Selector' configuration window in Guvnor. At the top, there are three radio buttons: 'Build whole package' (unselected), 'Use built-in selector' (selected), and 'Use custom selector' (unselected). Below these, the text 'Build package using following assets:' is followed by two checkboxes: 'When status' (checked) and 'When category' (unchecked). The 'When status' checkbox is followed by a dropdown menu showing 'Draft'. To the right of the 'When category' checkbox is a list of asset types: 'Eligibility rules', 'Pricing rules', and 'Test scenarios', each with a plus icon and a dropdown arrow. Below this list is a 'Build binary package:' label and a 'Build package' button. At the bottom, there is a 'Take snapshot:' label and a 'Create snapshot for deployment' button. A note at the bottom states: 'Building a package will collect all the assets, validate and compile into a deployable package.'

Figure 7.9. Built-in Selector

7.5.2. Custom Selector

You can use a custom selector if the built-in selector does not satisfy your requirement.

To configure a custom selector, you will need to "explode" the WAR file for Guvnor, and locate the `selectors.properties` file (note you can also put your own `selectors.properties` file in the system classpath if you like). In this file, you will find details on how you can configure a custom selector. The options are to use a DRL file, or the name of a class that you have written (and which is available on the classpath). Classes must implement the `AssetSelector` interface. DRL files can also be used and there is an example one in the `selectors.properties` file). Each selector you configure has a unique name in this properties file - and this is the name that you can use when building packages.

7.6. Snapshots

Snapshots, URLs and binary packages:

URLs are central to how built packages are provided. The Guvnor provides packages via URLs (for download and use by the Knowledge Agent). These URLs take the form of:

```
http://<server>/guvnor-webapp/org.drools.guvnor.Guvnor/package/  
<packageName>/<packageVersion>
```

`<packageName>` is the name you gave the package. `<packageVersion>` is either the name of a snapshot, or "LATEST" (if its LATEST, then it will be the latest built version from the main package,

not a snapshot). You can use these in the agent, or you can paste them into your browser and it will download them as a file.

Refer to the section on the Knowledge Agent for details on how you can use these URLs (and binary downloads) in your application, and how rules can be updated on the fly.

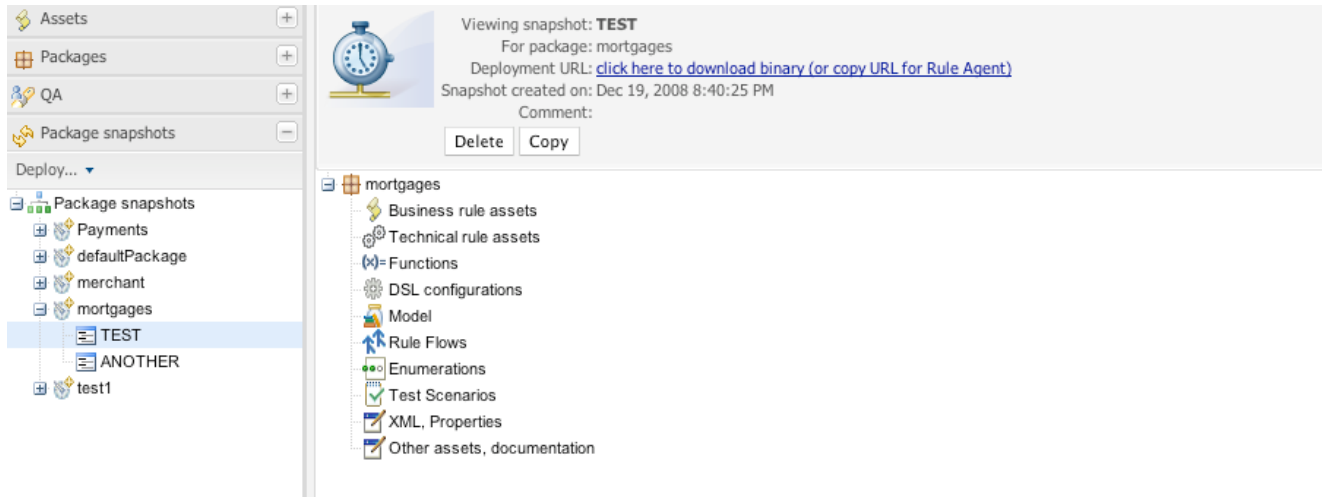


Figure 7.10. Deployment snapshots

The above shows deployment snapshots view. On the left there is a list of packages. Clicking on a specific package will show you a list of snapshots for that package (if any). From there you can copy, remove or view an asset snapshot. Each snapshot is available for download or access via a URL for deployment.

7.7. Advanced configuration

As drools supports various configuration options for a package (such as adding functions for "accumulate" etc), this can be done by adding a `x.package` or `x.conf` file to the package - files which contain name/value pairs in the "properties" style. These will then be automatically added to the package configuration. See the main drools documentation for all the things you can do.

Chapter 8. Administrative Functions

8.1. Categories

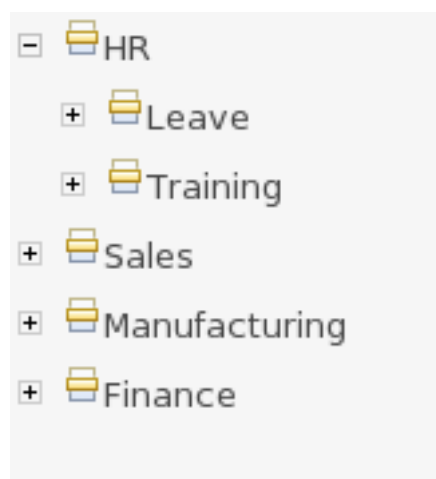


Figure 8.1. Categories

Categories allow rules (assets) to be labeled (or tagged) with any number of categories that you define. This means that you can then view a list of rules that match a specific category. Rules can belong to any number of categories. In the above diagram, you can see this can in effect create a folder/explorer like view of assets. The names can be anything you want, and are defined by the Guvnor administrator (you can also remove/add new categories - you can only remove them if they are not currently in use).

Generally categories are created with meaningful name that match the area of the business the rule applies to (if the rule applies to multiple areas, multiple categories can be attached). Categories can also be used to "tag" rules as part of their life-cycle, for example to mark as "Draft" or "For Review".

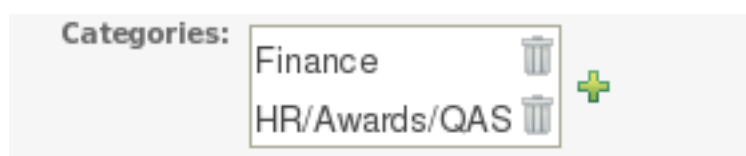


Figure 8.2. Assets can have multiple categories

The view above shows the category editor/viewer that is seen when you open an asset. In this example you can see the asset belongs to 2 categories, with a "+" button to add additional items

(use the trash can item to remove them). This means that when either category is used to show a list of assets, you will see that asset.

In the above example, the first Category "Finance" is a "top level" category. The second one: "HR/Awards/QAS" is still a single category, but it's a nested category: Categories are hierarchical. This means there is a category called "HR", which contains a category "Awards" (it will in fact have more sub-categories of course), and "Awards" has a sub-category of QAS. The screen shows this as "HR/Awards/QAS" - it's very much like a folder structure you would have on your hard disk (the notable exception is of course that rules can appear in multiple places).

When you open an asset to view or edit, it will show a list of categories that it currently belongs to. If you make a change (remove or add a category) you will need to save the asset - this will create a new item in the version history. Changing the categories of a rule has no effect on its execution.

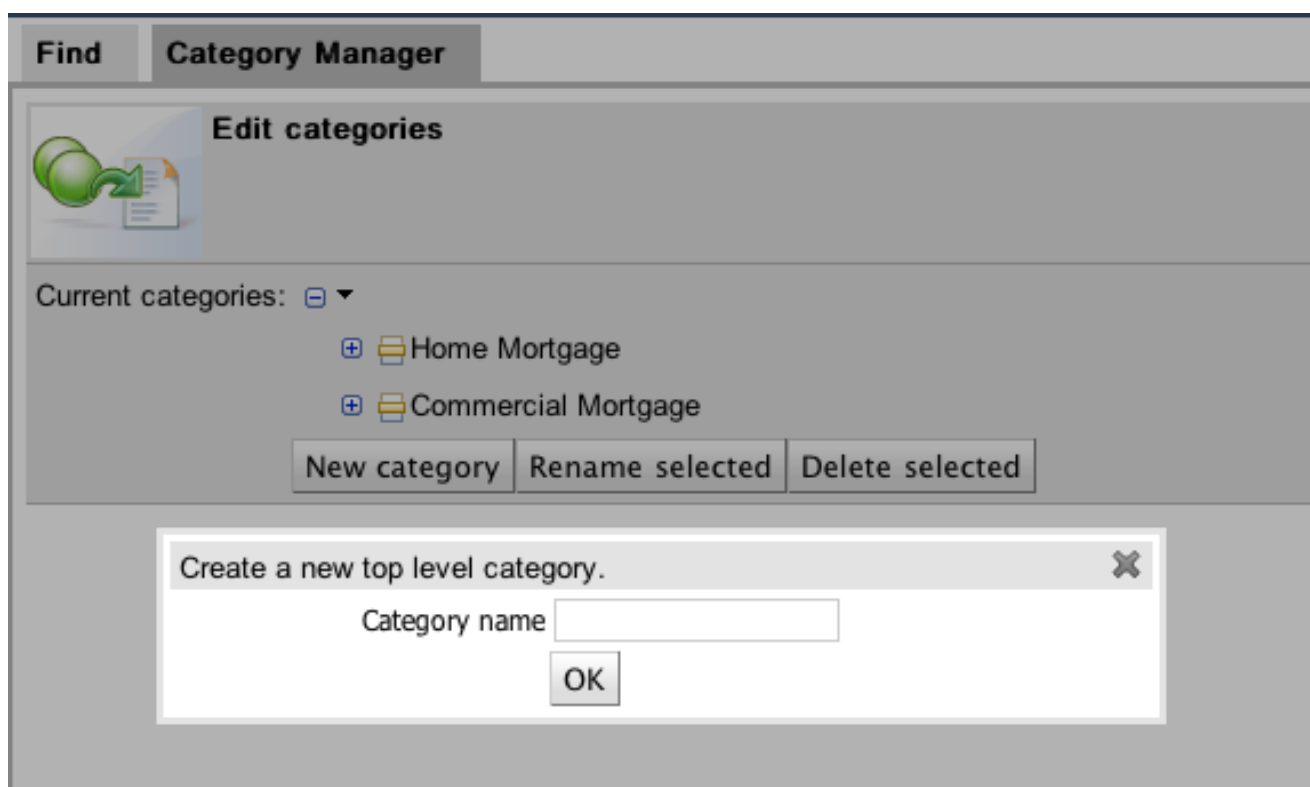


Figure 8.3. Creating categories

The above view shows the administration screen for setting up categories (there are no categories in the system by default). As the categories can be hierarchical you choose the "parent" category that you want to create a sub-category for. From here categories can also be removed (but only if they are not in use by any current versions of assets).

As a general rule, an asset should only belong to 1 or 2 categories at a time. Categories are critical in cases where you have large numbers of rules. The hierarchies do not need to be too deep, but should be able to see how this can help you break down rules/assets into manageable chunks. It's OK if it's not clear at first, you are free to change categories as you go.

8.2. Status management

Each asset (and also package) in Guvnor has a status flag set. The values of the status flag are set in the Administration section of the Guvnor. (you can add your own status names). Similar to Categories, Statuses do NOT effect the execution in any way, and are purely informational. Unlike categories, assets only have one status AT A TIME.

Using statuses is completely optional. You can use it to manage the lifecycle of assets (which you can alternatively do with categories if you like).

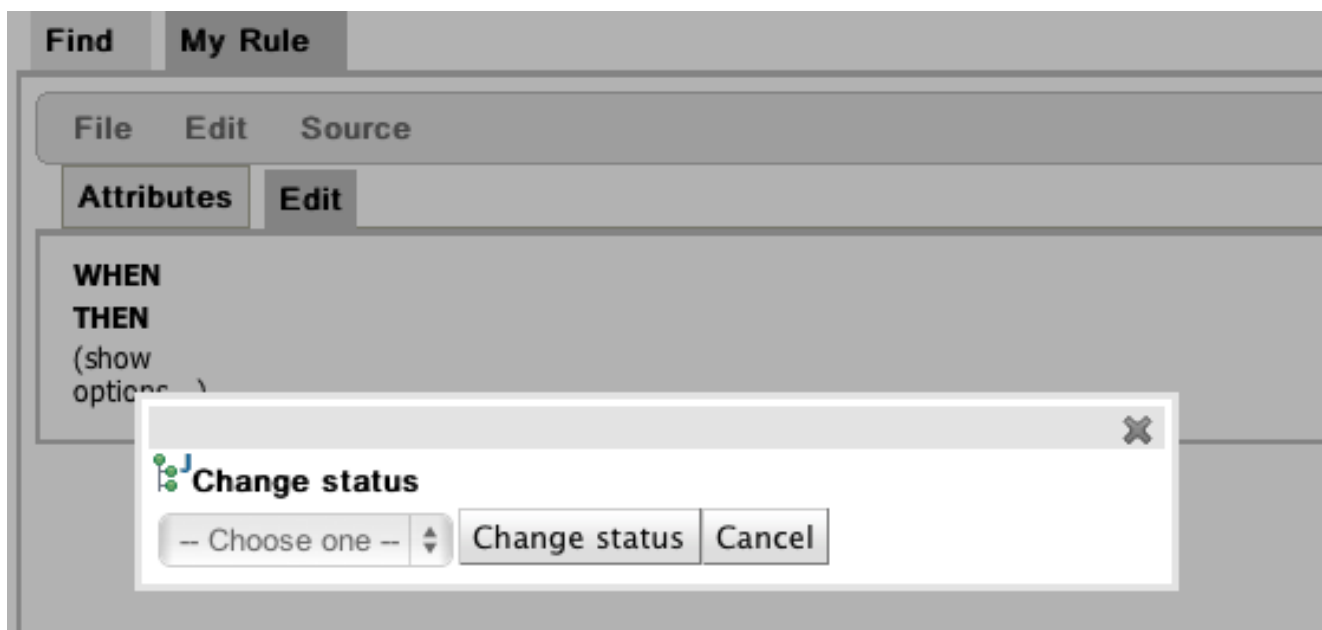


Figure 8.4. Asset status

You can change the status of an individual asset (like in the diagram above). Its change takes effect immediately, no separate save is needed.

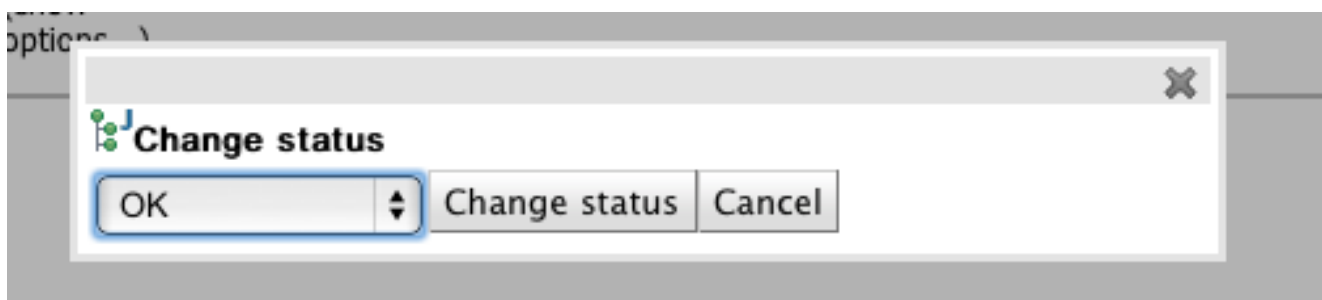


Figure 8.5. Asset status

You can change the status of a whole package - this sets the status flag on the package itself, but it ALSO changes the statuses on ALL the assets that belong to this package in one hit (to be the same as what you set the package to).

8.3. Archived items

This is basically the trash bin.

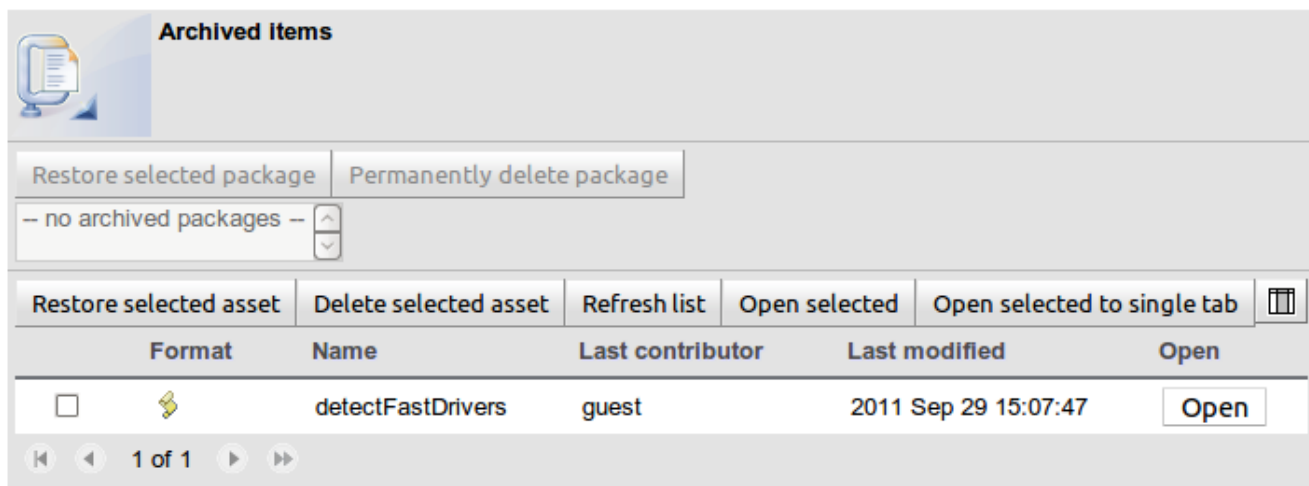


Figure 8.6. Archived items

If you've archived an asset (effectively deleting it), and you want to restore it, open this screen, select the asset and click the button **Restore selected asset**.

If instead, you want to permanently delete it, select it and click the button **Delete selected asset**.

8.4. Event Log

The event log shows the server events that have occurred.

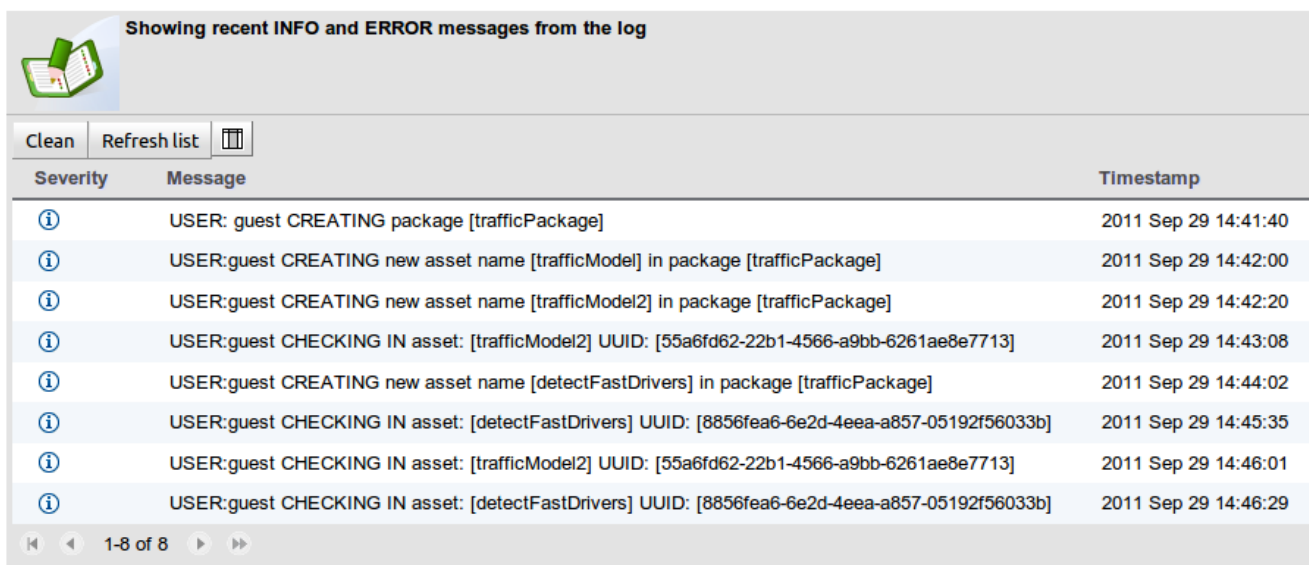




Figure 8.7. Event log

To clean the log, click the button **Clean**.

8.5. User permissions

When role based permissions are enabled (see administration guide), it's possible limit user permissions.


Permission details
TIP: To enable or disable authorization, open components.xml in WEB-INF

[Create new user mapping](#)
[Delete selected user](#)
[Open selected](#)
[Refresh list](#)


	User name	Administrator	Has package permissions	Has category permissions	Open
<input type="checkbox"/>	mailman				Open
<input type="checkbox"/>	admin	Yes			Open
<input type="checkbox"/>	guest				Open
<input type="checkbox"/>	alan_parsons				Open

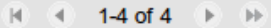



1-4 of 4

Figure 8.8. User permission mappings

To add a new user, click the button **Create new user mapping**.


Note
The user `guest` is special: if the user exists, Guvnor will automatically log in users as `guest`.


Warning
Do not delete or cripple the users `admin` or `mailman`. They are users used by the system.

To add or remove a permission to a user, select a user and click on the button **Open**. Click on the green **+** to add a permission type:

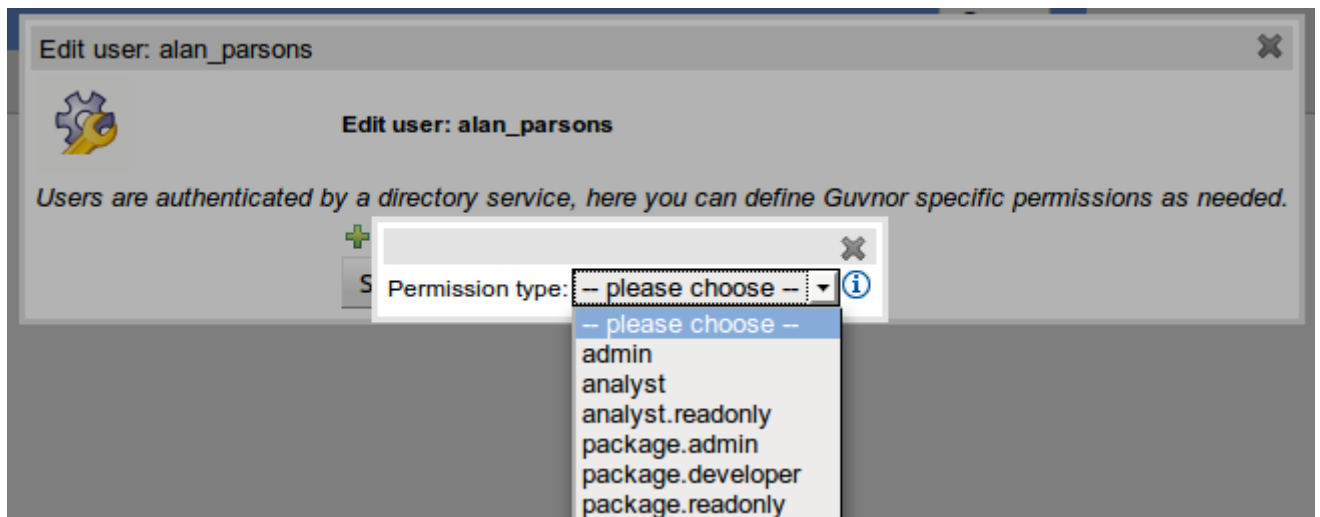


Figure 8.9. User permission mapping dialog

There are several permission types:

- **Admin:** this user can do everything
- **Analyst** or **Analyst read-only:** analyst permissions for a specific *category*. Create the category before attempting to assign users to it.
- **Package admin**, **Package developer** or **Package read-only:** package permissions for a specific *package*. Create the package before attempting to assign users to it.

8.6. Import and Export

The entire Guvnor repository can be exported to a file. This is useful as a backup. It can be imported in the same or another Guvnor server instance.

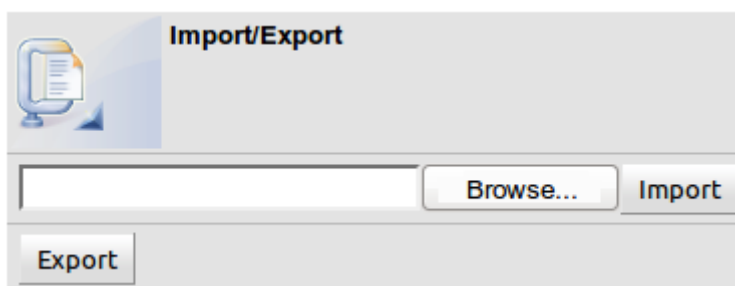


Figure 8.10. Import and export

To export the entire repository content into a zip file, click the button **Export**. You'll get a pop-up in your browser to save the zip file.

To import such a zip file, click the button **Browse**, select the zip file and click the button **Import**.

Both actions may take some time, especially if the repository is big.

8.7. Repository Configuration

Check Chapter "Configuring Guvnor to use an external RDBMS" for more information.

Part II. Developer Guide

This part covers Guvnor for software developers.

Chapter 9. Integrating rules with your applications

It's all very interesting to manage rules, but how do you use or "consume" them in your application? This section covers the usage of the KnowledgeAgent deployment component that automates most of this for you.

9.1. The Knowledge Agent

The knowledge agent is a component which is embedded in knowledge-api. To use this, you don't need any extra components. In fact, if you are using Guvnor, your application should only need to include the knowledge-api and drools-core dependencies in its classpath (drools and mvel JARs only), and no other rules specific dependencies.

Note that there is also a drools-ant ant task, so you can build rules as part of an Ant script (for example in cases where the rules are edited in the IDE) without using Guvnor at all - the drools-ant task will generate .pkg files the same as Guvnor.

Once you have "built" your rules in a package in Guvnor (or from the ant task), you are ready to use the agent in your target application.

The following example constructs an agent that will build a new KnowledgeBase from the files specified in the path String. It will poll those files every 60 seconds, which is the default, to see if they are updated. If new files are found it will construct a new KnowledgeBase. If the change set specifies a resource that is a directory its contents will be scanned for changes too.

```
KnowledgeAgent kagent = KnowledgeAgentFactory.newKnowledgeAgent( "MyAgent" );
kagent.applyChangeSet( ResourceFactory.newUrlResource( url ) );
KnowledgeBase kbase = kagent.getKnowledgeBase();
```

The KnowledgeAgent can accept a configuration that allows for some of the defaults to be changed. An example property is `drools.agent.scanDirectories`, by default any specified directories are scanned for new additions, it is possible to disable this.

```
KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
KnowledgeAgentConfiguration kaconf = KnowledgeAgentFactory.newKnowledgeAgentConfiguration();

// we do not want to scan directories, just files
kaconf.setProperty( "drools.agent.scanDirectories", "false" );

// the name of the agent
KnowledgeAgent kagent = KnowledgeAgentFactory.newKnowledgeAgent( "test
agent", kaconf );
```

```
// resource to the change-set xml for the resources to add

kagent.applyChangeSet( ResourceFactory.newUrlResource( url ) );
```

An example of the `change-set.xml` file.

```
<?xml version="1.0" encoding="UTF-8"?>
<change-set xmlns="http://drools.org/drools-5.0/change-set"
            xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
            xs:schemaLocation="http://drools.org/drools-5.0/change-set
http://anonsvn.jboss.org/repos/labs/labs/jbossrules/trunk/drools-api/src/
main/resources/change-set-1.0.0.xsd" >
    <add>
        <resource source="http://localhost:8080/guvnor/org.drools.guvnor.Guvnor/
package/mortgages/
LATEST" type="PKG" basicAuthentication="enabled" username="uid" password="pwd"/>
    </add>
</change-set>
```



Important

The User ID and Password in the change-set should be consistent with the requirements of the Authenticator configured in `components.xml`. By default, a `NilAuthenticator` is configured that does not authenticate HTTP requests and hence the `"basicAuthentication"`, `"username"` and `"password"` attributes are not required. If you change `components.xml` to use another Authenticator you will need to ensure appropriate authentication credentials are set in the change-set. Please refer to the "Security - Authentication and basic access" section of the "Administration Guide" for more details.



Note

The change-set schema, `change-set-1.0.0.xsd`, is also included in the `knowledge-api` JAR file applicable to the version of Drools you are running.

Resource scanning is not on by default, it's a service and must be started, the same is for notification. This can be done via the `ResourceFactory`.

```
ResourceFactory.getResourceChangeNotifierService().start();
```



```
ResourceFactory.getResourceChangeScannerService().start();
```

Following shows the deployment screen of Guvnor, which provides URLs and downloads of packages.

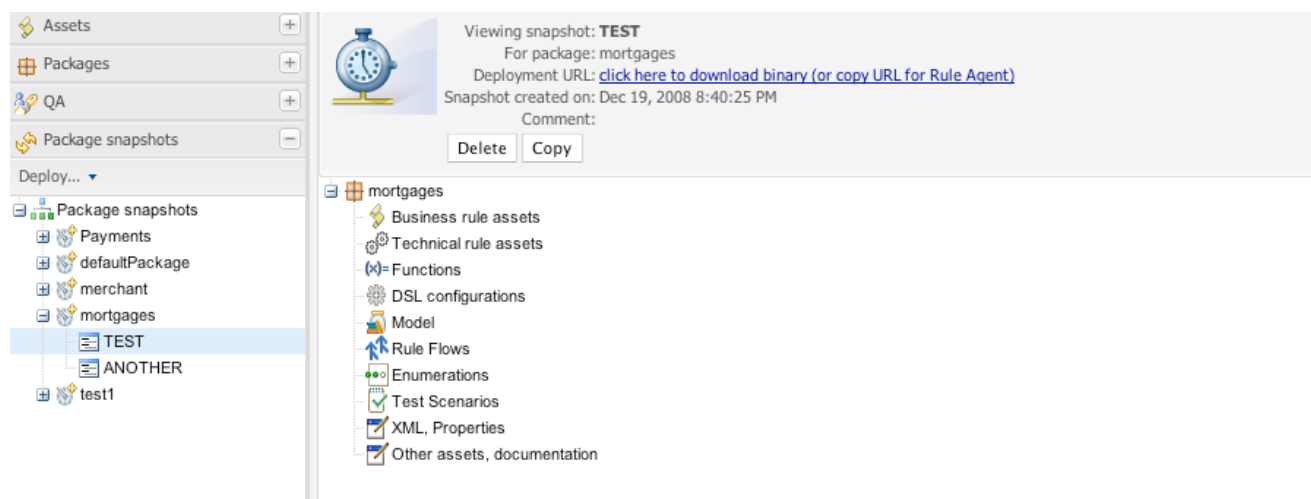


Figure 9.1. Snapshot deployment

You can see the "Package URI" - this is the URL that you would copy and paste into the `change-set.xml` file to specify that you want this package. It specifies an exact version (in this case to a snapshot) - each snapshot has its own URL. If you want the "latest" - then replace "NewSnapshot" with "LATEST".

You can also download a .pkg file from here, which you can drop in a directory and use the "file" or "dir" feature of the KnowledgeAgent if needed (in some cases people will not want to have the runtime automatically contact Guvnor for updates - but that is generally the easiest way for many people).

9.2. REST API

The repository back end can also be accessed via Rest. Rest is a http based protocol API - which has clients on all platforms and in all programming languages.

9.2.1. REST

Representational State Transfer (REST) is a style of software architecture for distributed hypermedia systems such as the World Wide Web. The term Representational State Transfer was introduced and defined in 2000 by [Roy Fielding](http://en.wikipedia.org/wiki/Roy_Fielding) [http://en.wikipedia.org/wiki/Roy_Fielding] in his doctoral dissertation.

REST-style architectures consist of clients and servers. Clients initiate requests to servers; servers process requests and return appropriate responses. Requests and responses are built around the transfer of representations of resources. A resource can be essentially any coherent and meaningful concept that may be addressed. A representation of a resource is typically a document

that captures the current or intended state of a resource. The REST protocol is often considered as a light protocol versus SOAP

9.2.2. Guvnor REST API

The Guvnor Rest API is divided in two groups of services : one around accessing rule assets by their names and packages and the second accessing rule assets by categories.

A rule asset represents all elements that can be stored and handled in Guvnor : a guided rule, a web decision table, a test scenario, etc.

The http address to use as base address is `http://{ServerName}/{httpPort}/{drools-Guvnor}/rest` where `ServerName` is the host name on the server on which Guvnor is deployed, `httpPort` the port number (8080 by default development), `drools-guvnor` the name of the archived deployed (guvnor-webapp-5.3.0 for version 5.3.0).

9.2.2.1. Accessing Rule Assets by Package

This part of the API allows to access all rule assets by package (with its name) and by the asset name.

Table 9.1. Accessing Rule Assets by Package

URL	Mode	Produces MIME-Type	Consumes MIME-Type	Description
/packages	GET	application/atom+xml	none	Returns all packages contained in the Guvnor repository in Atom Feed format
/packages	GET	application/json and application/xml	none	Returns all packages contained in the Guvnor repository in JSON or XML format
/packages	POST	application/atom+xml	application/octet-stream	Creates a new package from an input stream of DRL. Returns the newly created package in Atom Entry format.

URL	Mode	Produces MIME-Type	Consumes MIME-Type	Description
/packages	POST	application/json and application/ xml	application/octet- stream	Creates a package from an input stream of DRL. Returns the newly created package in JSON or XML format.
/packages	POST	application/atom +xml	application/atom +xml	Creates a package from an Atom Entry. Returns the newly created package in Atom Entry format
/packages	POST	application/json and application/ xml	application/json and application/ xml	Creates a package from JSON or XML. Returns the newly created package in JSON or XML format.
/packages/ {packageName}	GET	application/atom +xml	none	Returns the metadata of the package {packageName} as an Atom Entry.
/packages/ {packageName}	GET	application/json and application/ xml	none	Returns the metadata of the package {packageName} as a Package element (see UML representation below).
/packages/ {packageName}/ source	GET	plain/text	none	Returns the source code of the package {packageName} as a text file.

URL	Mode	Produces MIME-Type	Consumes MIME-Type	Description
/packages/ {packageName}/ binary	GET	application/octet- stream	none	Returns the compiled binary of the package {packageName} as a binary stream. If the package has not been compiled yet or its binary is not up to date, this will compile the package first.
/packages/ {packageName}/ versions	GET	application/atom +xml	none	Returns the list of package {packageName} versions as an Atom Feed .
/packages/ {packageName}/ versions/ {version}	GET	application/atom +xml	none	Returns the metadata of package {packageName} and of version {version} as an Atom Entry .
/packages/ {packageName}/ versions/ {version}/source	GET	text/plain	none	Returns the source code of package {packageName} and of version {version} as a text file.
/packages/ {packageName}/ versions/ {version}/binary	GET	application/octet- stream	none	Returns the binary (compiled code) of package {packageName} and of version {version} as an octet stream. If the package version was never built, it

URL	Mode	Produces MIME-Type	Consumes MIME-Type	Description
				returns HTTP code 500 with an error message.
/packages/{packageName}	PUT	application/atom+xml	none	Updates the metadata of package {packageName} with a given Atom Entry.
/packages/{packageName}	DELETE	none	none	Deletes package {packageName}.
/packages/{packageName}/assets	GET	application/atom+xml	none	Returns the list of rule assets contained in package {packageName} as an Atom Feed.
/packages/{packageName}/assets	GET	application/json and application/xml	none	Returns the list of rule assets contained in package {packageName} in JSON or XML format
/packages/{packageName}/assets/{assetName}	GET	application/atom+xml	none	Returns the rule asset {assetName} contained in package {packageName} in Atom Entry format.
/packages/{packageName}/assets/{assetName}	GET	application/json and application/xml	none	Returns the rule asset {assetName} contained in package {packageName} as an Asset in JSON or

URL	Mode	Produces MIME-Type	Consumes MIME-Type	Description
				XML format (see the UML representation below).
/packages/{packageName}/assets/{assetName}/binary	GET	application/octet-stream	none	Returns the binary content of rule asset {assetName} contained in package {packageName}. If this asset has no binary content, returns its source content instead.
/packages/{packageName}/assets/{assetName}/source	GET	plain/text	none	Returns the content of rule asset {assetName} contained in package {packageName}. If this is a binary asset, returns the binary data as a byte array.
/packages/{packageName}/assets	POST	application/atom+xml	application/atom+xml	Creates an asset in package {packageName} from the Atom Entry provided. Following info are required from the input Atom Entry: asset name, asset description, asset initial category, asset format.

URL	Mode	Produces MIME-Type	Consumes MIME-Type	Description
/packages/ {packageName}/ assets	POST	application/octet- stream	application/atom +xml	Creates a binary asset in package {packageName} from the input stream. The value of Slug header is used to indicate the name of the asset. This returns HTTP 500 error if the slug header is missing.
/packages/ {packageName}/ assets/ {assetName}	PUT	application/atom +xml	none	Updates the metadata of the rule asset {assetName} contained in package {packageName} with a given Atom Entry.
/packages/ {packageName}/ assets/ {assetName}	PUT	application/json and application/ xml	none	Updates the metadata of the rule asset {assetName} contained in package {packageName} with a given Asset (see the UML representation below).
/packages/ {packageName}/ assets/ {assetName}/ source	PUT	plain/text	none	Updates the source code of the rule asset {assetName} contained in

URL	Mode	Produces MIME-Type	Consumes MIME-Type	Description
				package {packageName}.
/packages/ {packageName}/ assets/ {assetName}/ binary	PUT	application/octet- stream	none	Updates the binary content of the rule asset {assetName} contained in package {packageName}.
/packages/ {packageName}/ assets/ {assetName}/ versions	GET	application/atom +xml	none	Returns the list of rule asset {assetName} versions contained in package {packageName} as an Atom Feed .
/packages/ {packageName}/ assets/ {assetName}/ versions/ {version}	GET	application/atom +xml	none	Returns the metadata of rule asset {assetName} of version {version} contained in package {packageName} as an Atom Entry .
/packages/ {packageName}/ assets/ {assetName}/ versions/ {version}/source	GET	text/plain	none	Returns the source code of rule asset {assetName} of version {version} contained in package {packageName} as a text file.
/packages/ {packageName}/ assets/ {assetName}/	GET	application/octet- stream	none	Returns the binary content of rule asset {assetName} of

URL	Mode	Produces MIME-Type	Consumes MIME-Type	Description
versions/ {version}/binary				version {version} contained in package {packageName}. If this asset has no binary content, returns its source content instead.
/packages/ {packageName}/ assets/ {assetName}	DELETE	none	none	Deletes the rule asset {assetName} contained in package {packageName}.
/packages/ {packageName}/ snapshot/ {snapshotName}	POST	none	none	Creates a snapshot {assetName} snapshotName for the package {packageName}.
/packages/ {packageName}/ exists	GET	none	none	Determines if the {packageName} exists.
/packages/ {packageName}/ assets/ {assetName}/ exists	GET	none	none	Determines if the asset {assetName} in the {packageName} exists.
{packageName}/ assets	POST	application/json and application/ xml	multipart/form- data	Creates the asset {assetName} in the {packageName}.
{packageName}/ assets/ {assetName}	PUT	application/json and application/ xml	multipart/form- data	Updates the asset {assetName} in the {packageName}.

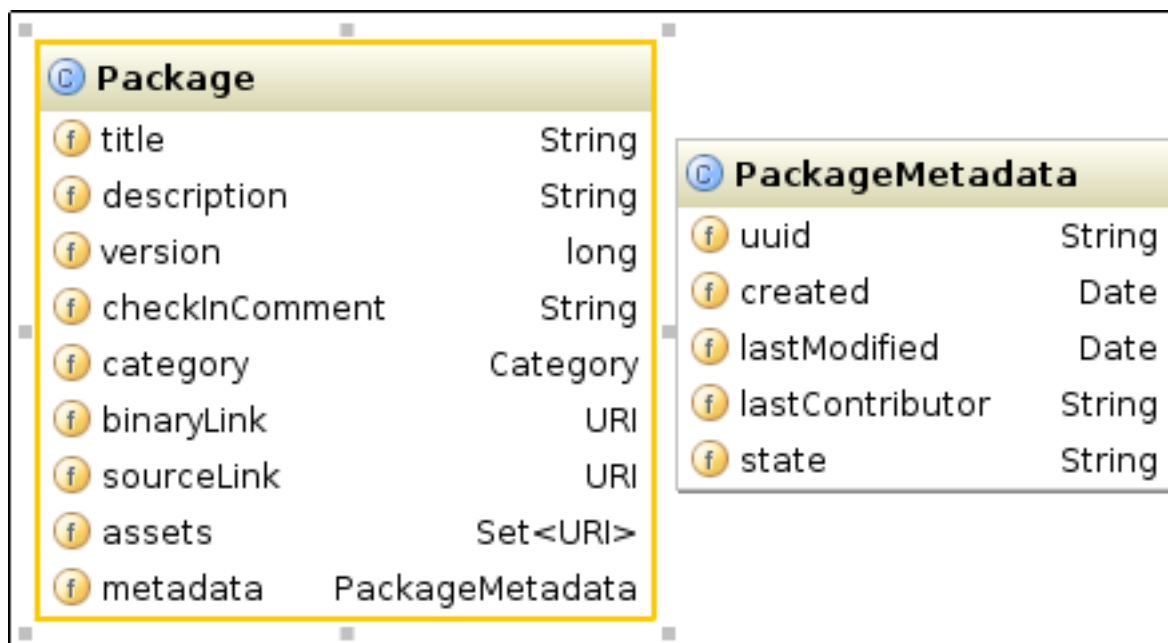


Figure 9.2. UML representation of the Package Object

9.2.2.2. Getting rule Assets by Category

This part of the API uses as entry point the categories defined for each rule asset when it is created by the user in Guvnor

Table 9.2. Accessing Rule Assets by Category

URL	Mode	MIME-Type	Description
/categories/	GET	application/json and application/xml	Lists all categories
/categories/{categoryPath}	GET	application/json and application/xml	Returns the info of a single category.
/categories/{categoryPath}/children	GET	application/json and application/xml	Lists all subcategories under the category {categoryPath}
/categories/{categoryPath}/assets	GET	application/atom/xml	Retrieves an Atom feed to all Rule assets that have the category {categoryPath}
/categories/{categoryPath}/assets	GET	application/json and application/xml	Returns a list of objects of Assets representing a Rule Asset that have the category {categoryPath}

URL	Mode	MIME-Type	Description
/categories/ {categoryPath}/ assets//page/{page}	GET	application/json and application/xml	Returns a list of objects of Assets representing a Rule Asset that have the category {categoryPath} and retrieves page {page} with is a numeric (1,2, 3, etc..). A page contains 10 elements. If the list contains 20 elements then the list will have 2 pages. You first have to call for page 1 and then call for page 2
/categories/ {categoryPath}	PUT	none	Creates a new category with the category path {categoryPath}.
/categories/ {categoryPath}	DELETE	none	Deletes the category with the category path {categoryPath}.

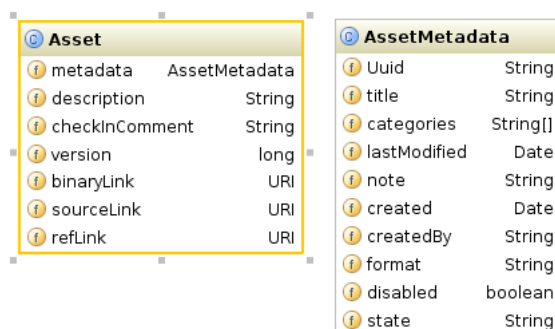


Figure 9.3. UML representation of the Asset Object

9.2.3. Source code Example

We are giving a list of example to help using the Guvnor's Rest API

9.2.3.1. Retrieving and updating Web decision table

We are using apache CXF in our example to show how to access the Rest API of Guvnor. In the example here we are getting and updating a web decision table. But this example applies to all Guvnor asset type.

Example 9.1. Retrieving the source code of the web decision table

```
WebClient client = WebClient.create("http://127.0.0.1:8080/");
String content=client.path("guvnor-webapp-${project.version}/rest/packages/
essaiRest/assets/tab2/source").accept("text/plain").get(String.class);
GuidedDecisionTable52 dt =
    GuidedDTXMLPersistence.getInstance().unmarshal(content);
```

In the first line of code above, we are first creating a `WebClient` variable that points to the server (here on localhost on port 8080).

In the second line of code above, we are retrieving the source content by accessing the `/rest/packages/{packageName}/assets/{assetName}/source` where in our case `packageName` is "essaiRest" and `assetName` is "tab2".

In the third line of code above, the source code we get is the data structure of a Web decision table. So to be able to manipulate the Web decision table, we have to transform the string variable (the source code that contains the xml of the data structure of the web decision table) in the java structure (a java class) for web decision table `GuidedDecisionTable52`. All guided asset in Guvnor have a java structure to manipulate them

Example 9.2. updating the source code of the web decision table

```
String authorizationHeader = "Basic " +
    org.apache.cxf.common.util.Base64Utility.encode("guest:".getBytes());
GuidedDecisionTable52 dt = new GuidedDecisionTable52();
..
Do some stuff here
..
String newContent = GuidedDTXMLPersistence.getInstance().marshal(dt);
WebClient client2 = WebClient.create("http://127.0.0.1:8080/");
client2.header("Authorization", authorizationHeader);
Response response= client2.path("guvnor-webapp-${project.version}/rest/
packages/essaiRest/assets/tab2/source").accept("application/
xml").put(newContent);
```

In the first line of code above, we are first creating a java `String` variable that contains the authorization element needed to update an asset on the Guvnor repository.

In the next lines of code above, we are doing some stuff to modify the Web decision table.

In the following lines of code above, we are first transforming the java structure in an xml structure that we put in a java String variable. Then we again create a WebClient but this time we are also filling the header with a variable "Authorization" that contains the String we built in the first line and that contains the user name "guest" and its password (here no password). We then put the new content on the Guvnor repository.

Next you can find the pom.xml file to use the Guvnor Rest API in case you are using Maven.

Example 9.3. pom.xml content for our example

```
<project      xmlns="http://maven.apache.org/POM/4.0.0"      xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
      xsi:schemaLocation="http://maven.apache.org/POM/4.0.0      http://
maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>org.drools.examples</groupId>
  <artifactId>dt-example</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <build>
    <finalName>cxfr-rest-example</finalName>
    <plugins>
      <plugin>
        <groupId>org.apache.maven.plugins</groupId>
        <artifactId>maven-compiler-plugin</artifactId>
        <configuration>
          <source>1.6</source>
          <target>1.6</target>
        </configuration>
      </plugin>
    </plugins>
  </build>
  <dependencies>
    <dependency>
      <groupId>org.apache.cxf</groupId>
      <artifactId>cxf-bundle-jaxrs</artifactId>
      <version>2.3.0</version>
    </dependency>
    <dependency>
      <groupId>org.drools</groupId>
      <artifactId>droolsjbpm-ide-common</artifactId>
      <version>5.3.0</version>
    </dependency>
  </dependencies>
</project>
```

9.3. WebDAV and HTTP

The repository back end can also be accessed via webdav. WebDAV is a http based file system API - which has clients on all platforms (some operating systems such as windows can connect directly to WebDAV repositories almost like a file system).

9.3.1. WebDAV

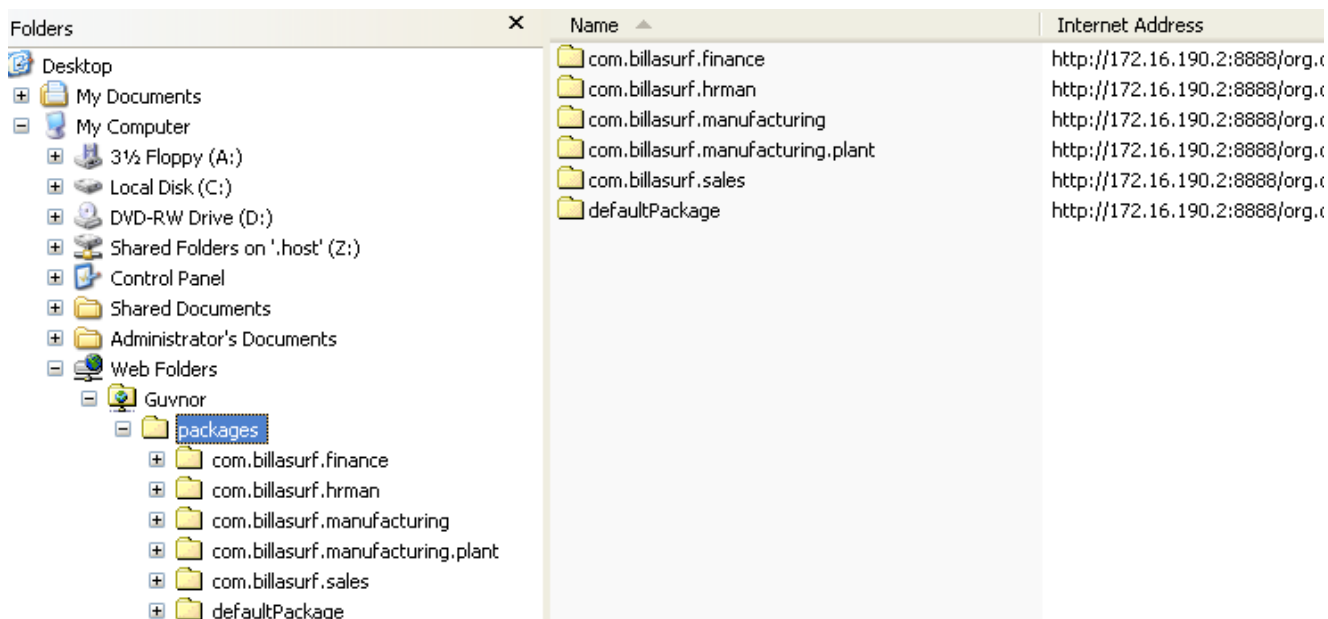


Figure 9.4. Windows webdav

In windows the "web folders" feature can be used. On OSX - the free Cyberduck client can work well. To access the repository as webdav, you the url is the same as the web interface, only with / webdav at the end, instead of `Guvnor.html`. Authentication will be required to get access this way. This will show a packages and snapshots directory - the snapshots directory is read only (a view of created snapshots of packages). The packages directory will contain a list of packages in the repository, drilling in to them will show the individual assets as files.

9.3.2. URLs

There are a few other URLs which are handy to know exist. The package deployment URL mentioned in the section about knowledge agent deployment also has a few other features: By appending `.drl` to the end of a URL, you will show the generated DRL for that package. e.g.: `/package/testPDSGetPackage/LATEST.drl` - will show the DRL (not the binary package) for the latest package. Further to this, you can append `/assetName.drl` - and it will show the generated DRL for that item. (even if it isn't a DRL file). E.g. `/package/testPDSGetPackage/LATEST/SomeFile.drl`.

9.4. Eclipse Guvnor integration

The Eclipse Guvnor tools (EGT) provide the ability to push/pull artifacts from the Guvnor repository server and the developers workspace in eclipse. It is therefore possible for artifacts to be both managed via Guvnor as well as in traditional developer friendly SCM systems (such as subversion). The Guvnor repository is not intended as a Source Code Management (SCM) solution, and the EGT are not intended to be Eclipse “team provider” extensions or replacements. Rather, the Guvnor repository is a location where certain artifacts (such as rules and SOA policy definitions) are controlled (“governed”) by policies defined by the deployment environment. The purpose of the EGT is then to enable access to resources held by the Guvnor repository, so they can be used in development. Thus, limited capabilities for reading, writing, adding, and removing Guvnor repository resources are provided in the EGT.

9.4.1. Source Code and Plug-in Details

The source code for the EGT is available in [github](http://github.com/droolsjbpm/droolsjbpm-tools/tree/master/drools-eclipse) [http://github.com/droolsjbpm/droolsjbpm-tools/tree/master/drools-eclipse]. EGT consist of two plug-ins: org.guvnor.tools and org.eclipse.webdav. They require Eclipse 3.3.x. The current Eclipse Drools plug-ins are also useful for viewing Guvnor repository resources such as rule definitions, but not required for operation of the EGT.

9.4.2. Functionality Overview

Views and Perspective: The EGT contains two views – Repository Explorer and Version History – that will be the center of most interaction with Guvnor. Eclipse standard views such as Properties and the Resource Navigator are also useful. While each of these views can be opened and positioned independently within an Eclipse workbench, the Guvnor perspective provides a convenient method of getting a suggested layout. In the Eclipse workbench menu, choose Window, Open Perspective, Other to get the perspective list and then choose "Guvnor Repository Exploring."

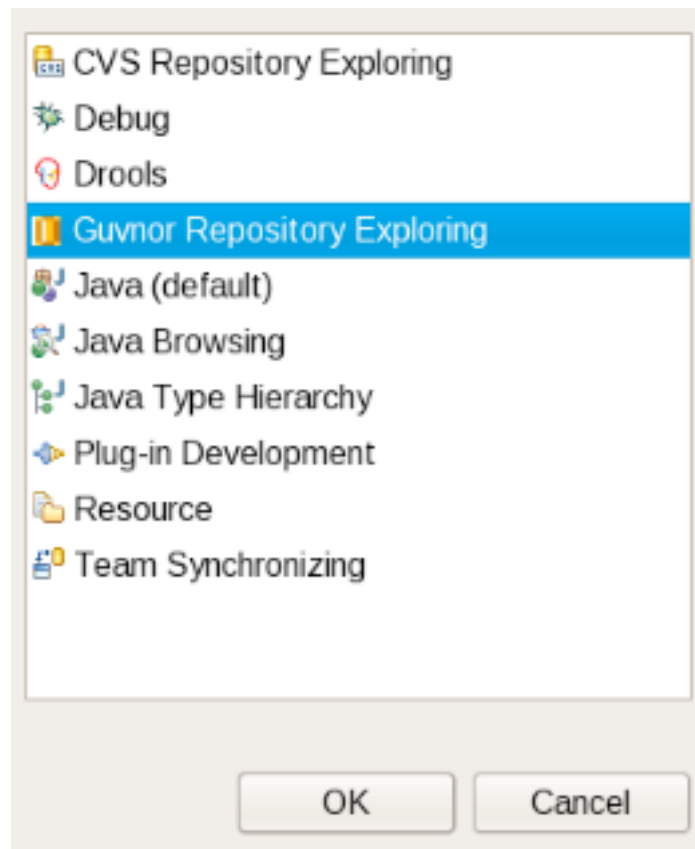


Figure 9.5. Views and perspectives

This opens the Guvnor perspective.

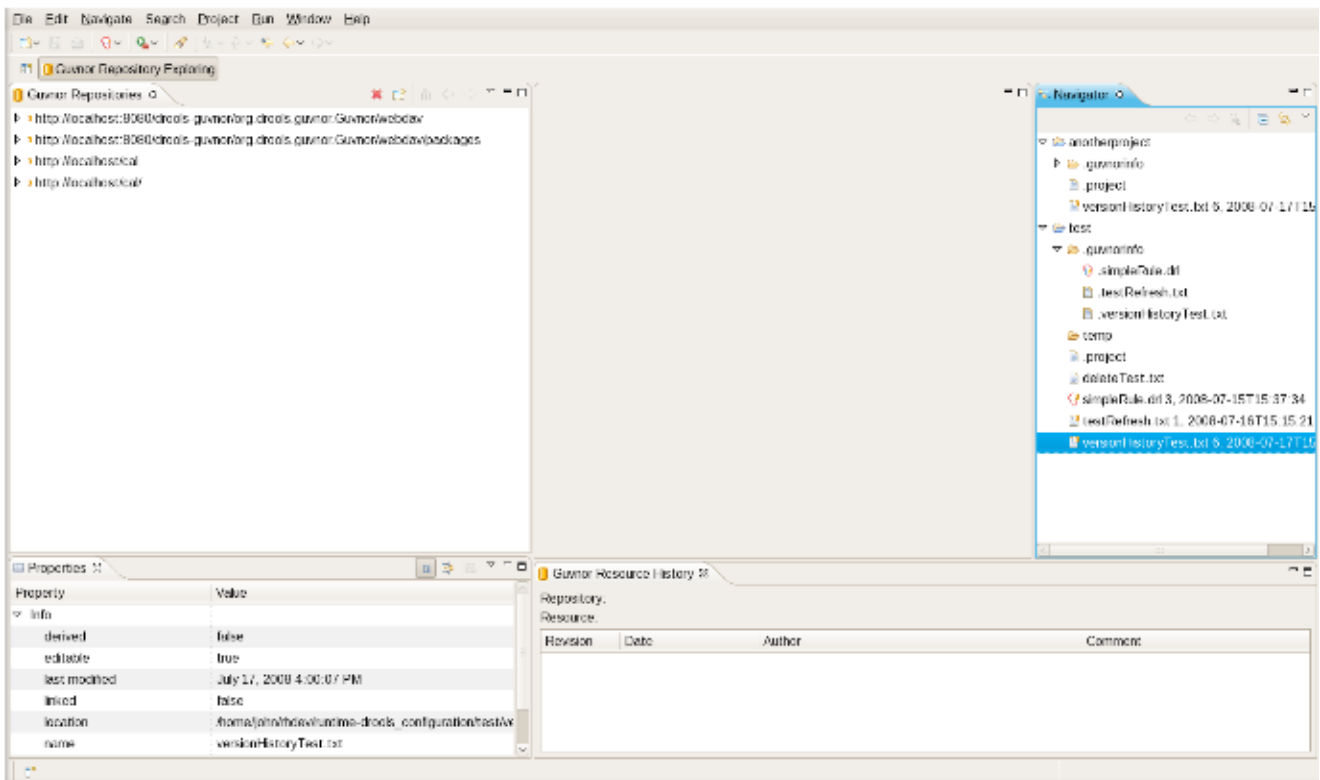


Figure 9.6. Views and perspectives

On the left side is the Guvnor Repository Explorer and the Eclipse Properties views, the Guvnor Resource History view is on the bottom, and the Eclipse Resource Navigator is on the right side. The purpose of the Guvnor Repository Explorer is to enable access to Guvnor repository resources in a standard tree format, and the Guvnor Resource History view shows revisions of specific resources available in the repository.

9.4.3. Guvnor Connection Wizard

After opening the Guvnor perspective, the first task is to make a connection to a Guvnor repository. This is handled by the Guvnor Connection wizard. This wizard appears in a number of places within the EGT (as detailed below), but in this section we will cover only the two most basic entry points. The Guvnor Connection wizard can be started using the Eclipse menu: File , New , Other , Guvnor , Guvnor repository location, or in the Guvnor Explorer using the drop-down menu:

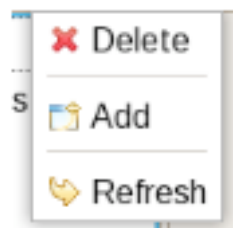


Figure 9.7. Connection wizard

or the menu button:

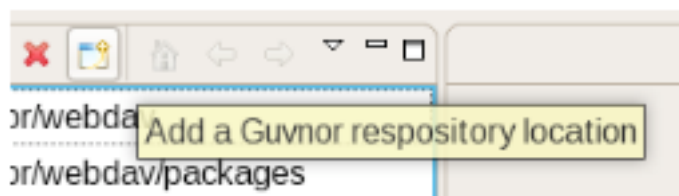


Figure 9.8. Connection wizard

Choosing either of these will start the Guvnor connection wizard:

New Guvnor location

Create a new Guvnor repository connection



Location:	<input type="text" value="localhost"/>
Port:	<input type="text" value="8080"/>
Repository:	<input type="text" value="/drools-guvnor/org.drools.guvnor.Guvnor/webdav"/>
User Name:	<input type="text"/>
Password:	<input type="password"/>
<input type="checkbox"/> Save user name and password	
<p>NOTE: Saved passwords are stored on your computer in a file that is difficult, but not impossible, for an intruder to read.</p>	
<div><input type="button" value="Finish"/> <input type="button" value="Cancel"/></div>	

Figure 9.9. Connection wizard

Default values appear in the Location, Port, and Repository fields. (See "[Guvnor plugin Preferences](#)" for details about how to change these default values.) Of course, any of these fields can be edited by typing in the corresponding text box. Drag-and-drop or paste into the

Location field of a typical Guvnor repository URL such as <http://localhost:8080/guvnor-webapp/org.drools.guvnor.Guvnor/webdav> results in the URL being parsed into the respective fields as well. The authentication information (user name and password) can optionally be stored in the Eclipse workbench's key-ring file based on the selection of "Save user name and password." If the authentication information is not stored in the key-ring, then the EGT uses session authentication, which means that the credentials supplied are used only for the lifetime of the Eclipse workbench instance.

If authentication information is not stored in the key-ring or the authentication information (key-ring or session) is not valid, the EGT will prompt for authentication information when it has to access the Guvnor repository:

Guvnor Repository Log in

Authentication required for repository: <http://localhost/cal>



User Name:

Password:

☐ Save user name and password

NOTE: Saved passwords are stored on your computer in a file that is difficult, but not impossible, for an intruder to read.




Figure 9.10. Login

If authentication fails, the EGT will retry once and then issue an authentication failure error. (If an authentication failure error occurs, you can retry the same operation and supply different authentication information.) Note that the EGT calls the Guvnor repository at various times, such as when determining if resource updates are available, so, if you use session authentication, the authentication dialog will appear at different times during the Eclipse workbench session, depending on what actions you take. For ease of use, we recommend saving the authentication information in the Eclipse key-ring. (The Eclipse key-ring file is distinct from key-ring files found in some platforms such as Mac OS X and many forms of Linux. Thus, sometimes if you access a Guvnor repository outside the EGT, the key-ring files might become unsynchronized and you will be unexpectedly prompted for authentication in Eclipse. This is nuisance, but your usual credentials should apply in this case.)

Once the Guvnor connection wizard is complete, the new Guvnor repository connection will appear in the Guvnor Repository Explorer. You can then expand the tree to view Guvnor repository contents.

9.4.4. Guvnor Repository Explorer

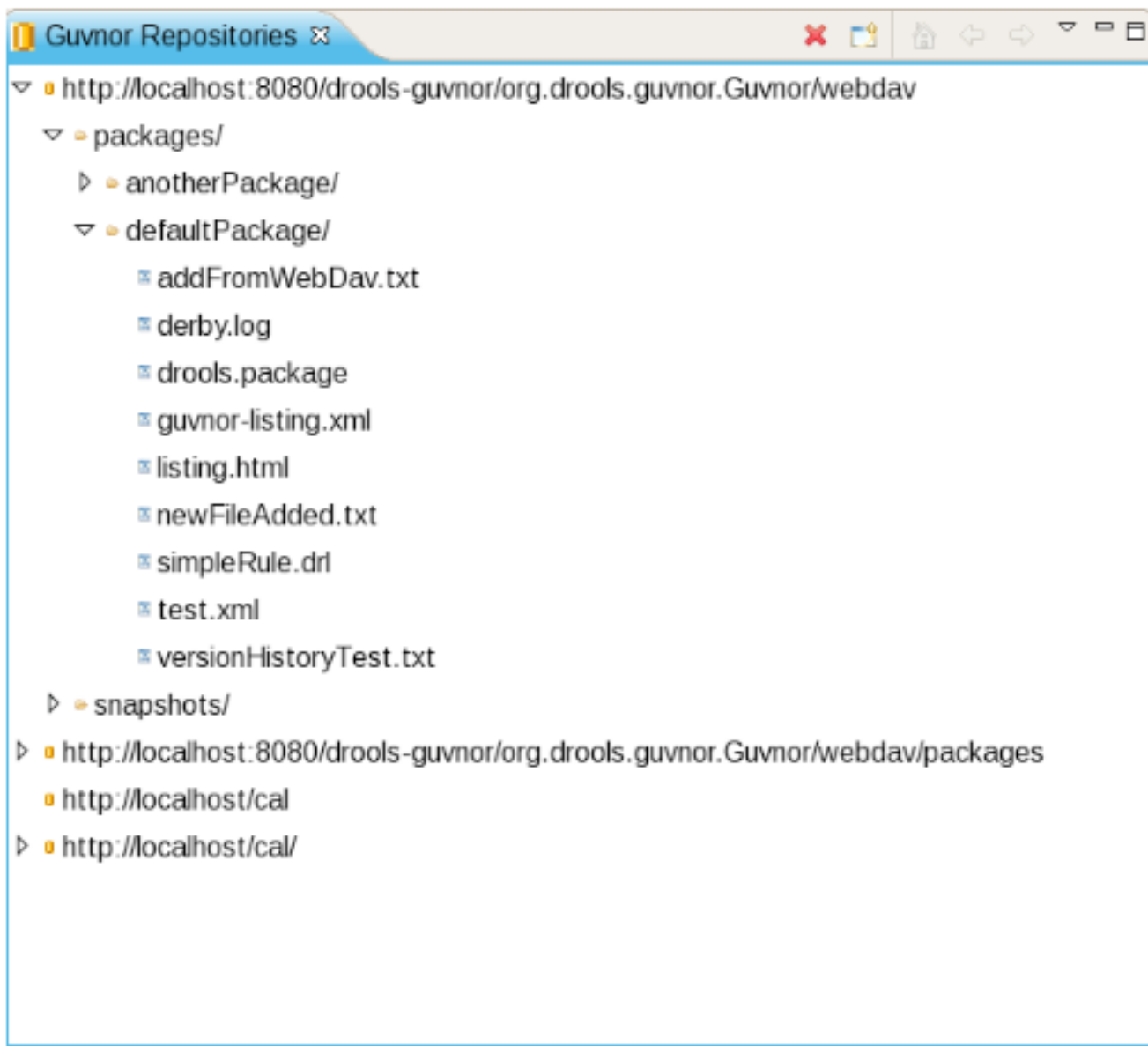
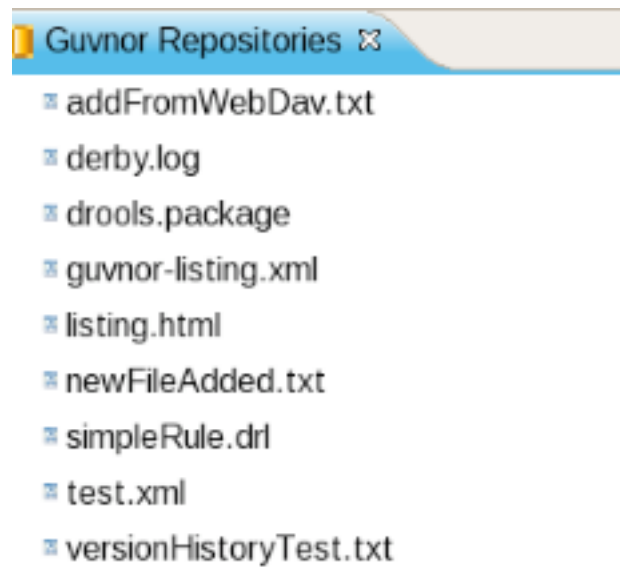


Figure 9.11. Explorer

The Guvnor Repository Explorer view contains tree structures for Guvnor repository contents. As described above, there are menu and tool-bar actions for creating Guvnor repository connections. The red "X" in the tool-bar and "Delete" in the menu removes a Guvnor repository connection, and the "Refresh" menu item reloads tree content for the selected node. Finally, there are a number of tool-bar/menu items in support of "drill-into" functionality: one the tool-bar these are represented by the house ("return to top level/home") and the arrows (go into/back). Drill-down is useful when working with deeply nested tree structures and when you wish to concentrate on only branch of the tree. For example, drilling into the "defaultPackage" node shown above changes the tree view to:

**Figure 9.12. Explorer**

That is, we see only the contents of “defaultPackage” in the tree. Clicking on the house button, or selecting “Go Home” returns the tree to the top-level structure shown in the previous picture above.

There are a number of operations that can be performed on Guvnor repository files. Selecting a file in the Guvnor repository causes the Eclipse Properties view to update with details about that file:

The screenshot shows the 'Properties' view for the file `versionHistoryTest.txt`. The table below represents the data shown in the view:

Property	Value
Created	2008-07-15T15:28:00Z
Last Modified	2008-07-17T15:41:51
Location	/packages/defaultPackage/versionHistoryTest.txt
Name	versionHistoryTest.txt
Revision	6
Type	file

Figure 9.13. Properties

Double-clicking on a folder (directory) in the tree will cause that folder to expand if collapsed and collapse if expanded. Double-clicking on a file in the tree will cause a read-only editor in Eclipse to open, showing the contents of that file:

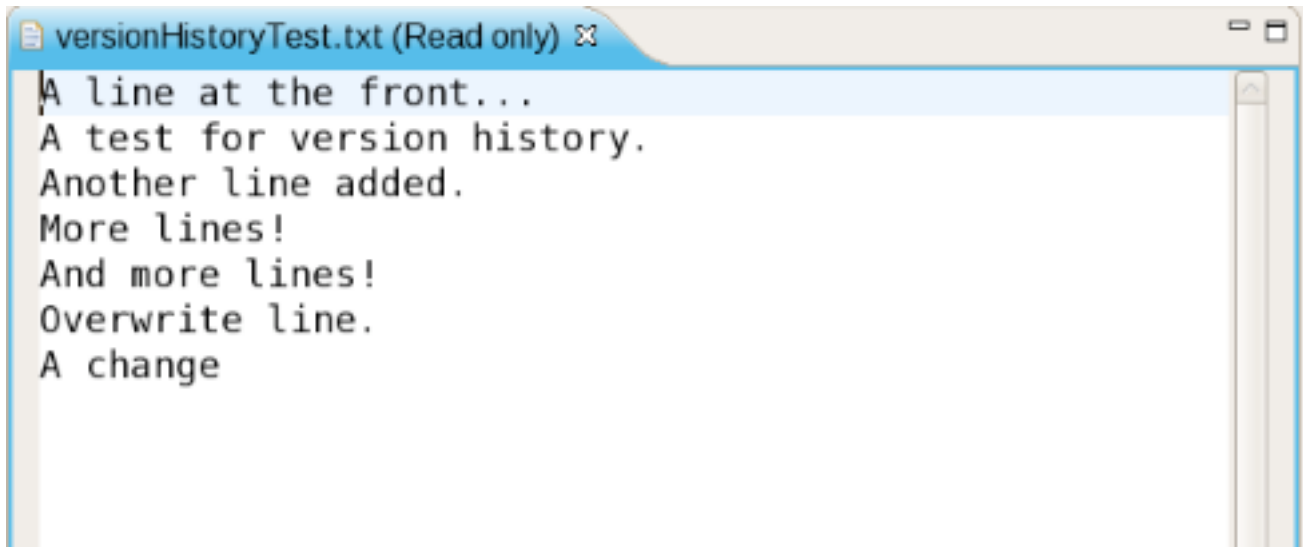


Figure 9.14. Comments

Dragging a file from the Guvnor repository tree to a folder in an Eclipse local project (for example in the Eclipse Resource Navigator view) will cause a copy of that file to be made in the local Eclipse workspace. (Note: You can also “Save As...” when a file is open in a read-only editor to save a local writable copy of the contents. Doing so, however, will not associate the file created with its Guvnor source.) Finally, you can view the revision history of a file selected in the tree using the “Show History” context menu item. (The details of resource history will be discussed below.)

9.4.5. Local Copies of Guvnor Files

As mentioned in the Introduction, the main purpose of the EGT is to allow development using resources held in a Guvnor repository. There are two method of getting local copies of Guvnor repository resources:

1. Drag-and-drop from the Guvnor Repository Explorer, as described above.
2. Using the “import from Guvnor” wizard, as described below.

When local copies of Guvnor repository files are created, the EGT sets an association between the local copy and the master file in the repository. (This information is kept in the (normally) hidden `.guvnorinfo` folder in the local project and, like all metadata, should not be changed by end users.) This association allows for operations such as update and commit in synchronization with the master copy held in the Guvnor repository. The EGT decorates local resources associated with Guvnor repository master copies. This decoration appears in Eclipse views conforming to

the Eclipse Common Navigator framework, such as the Eclipse Resource Navigator and the Java Package Explorer. The image below shows decoration in the Eclipse Resource Navigator:

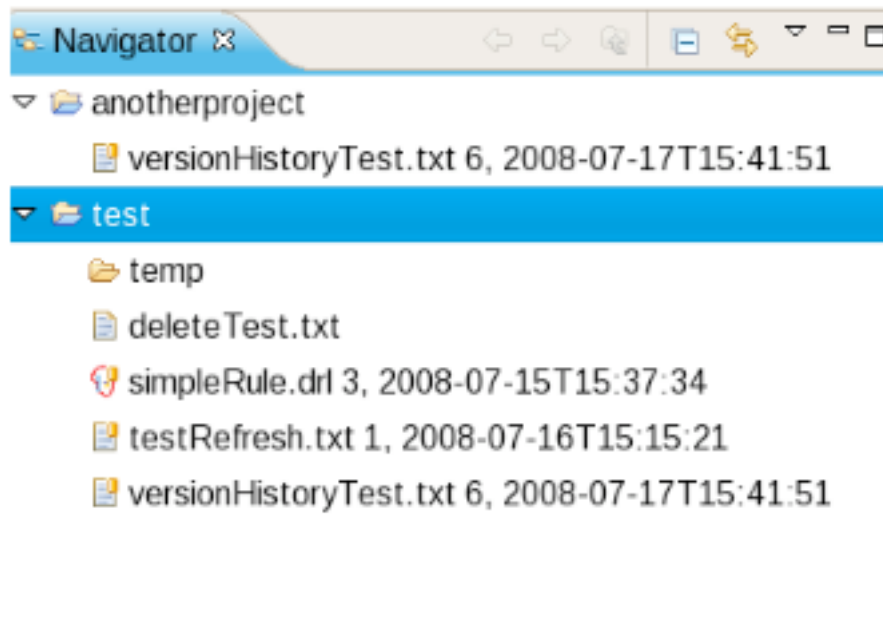


Figure 9.15. Navigator

Note the Guvnor icon decorator on the top right of the file images, and the Guvnor revision details appended to the file names. (The presence/location of these can be changed. See "[Guvnor plugin Preferences](#)" for details.) Here we see that, for example, `simpleRule.drl` is associated with a Guvnor repository resource and the local copy is based on revision 3, with a 7-15-2008, 15:37:34 date/time stamp. The file `deleteTest.txt`, however, is not associated with a Guvnor repository file. Further details about the association can be found in the standard Eclipse properties page, via the context menu "Properties" selection:



Figure 9.16. Properties

The EGT contributes a property page to the standard Eclipse properties dialog, the contents of which are shown above. The specific Guvnor repository, the location within the repository, the version (date/time stamp) and revision number are displayed.

9.4.6. Actions for Local Guvnor Resources

The EGT provides a number of actions (available through the “Guvnor” context menu on files) for working with files, both those associated with Guvnor repository master copies and those not associated. The actions are: 1.Update 2.Add 3.Commit 4.Show History 5.Compare with Version 6.Switch to Version 7.Delete 8.Disconnect Each of these actions will be described below.

Update Action:

The Update action is available for one or more Guvnor resources that are not in synchronization with the Guvnor repository master copies. These resources would not be in synchronization because either/both (1) there are local changes to these resources or (2) the master copies have changed in the Guvnor repository. Performing the Update action replaces the local file contents with the current contents from the Guvnor repository master copies (equivalent to “Switch to version” for latest version).

Add Action

The Add action is available for one or more local files that are not associated with a Guvnor repository master copy. Choosing the Add action launches the “Add to Guvnor” wizard:

Select Guvnor repository location

Select an existing Guvnor repository location or create a new one



☐ Create a new Guvnor repository location

☒ Use an existing Guvnor repository location

`http://localhost:8080/drools-guvnor/org.drools.guvnor.Guvnor/webdav`

`http://localhost/cal/`

`http://localhost/cal`

`http://localhost:8080/drools-guvnor/org.drools.guvnor.Guvnor/webdav/packages`



< Back

Next >

Finish

Cancel

Figure 9.17. Add action

The first page of the wizard asks for the selection of the target Guvnor repository and gives the choice to create a new Guvnor repository connection (in which case the second page is the same as the Guvnor Connection wizard described above). Once the target Guvnor repository is chosen, the wizard then asks for the folder location to add the selection files:

Select folder

Select the target folder in the Guvnor repository

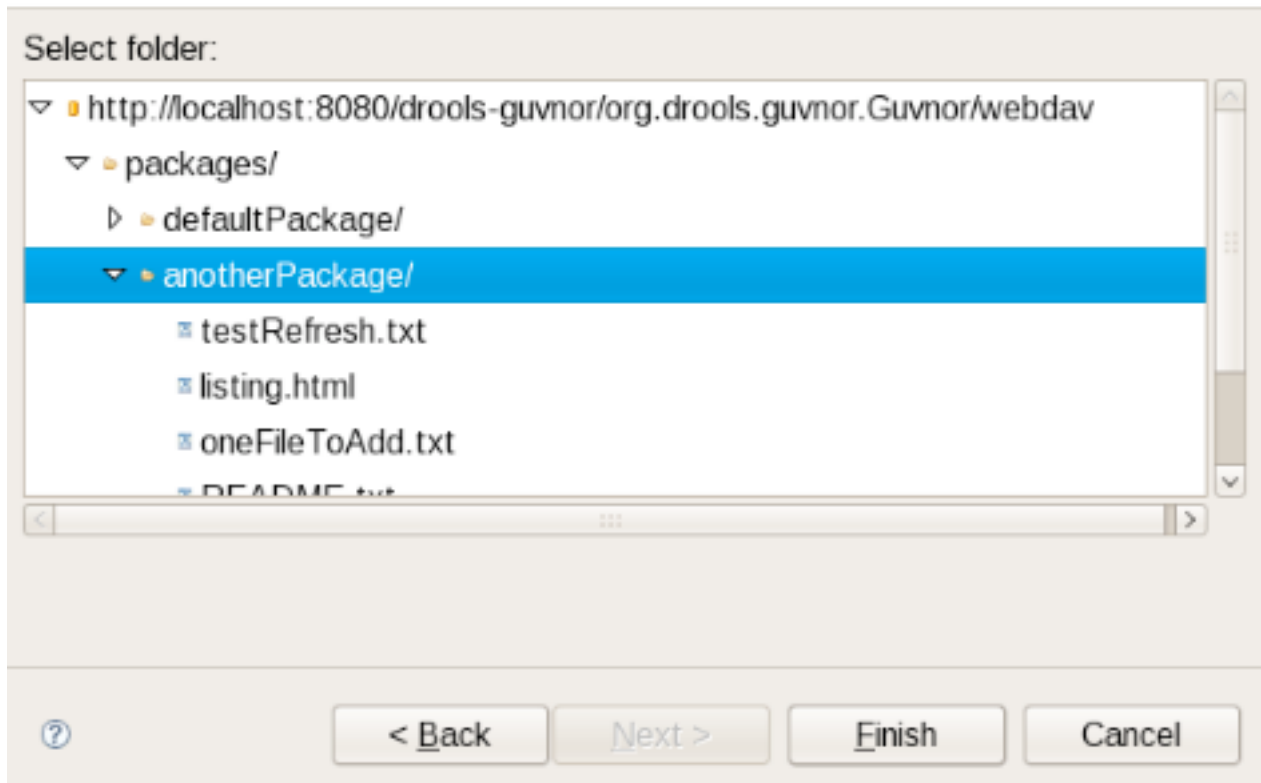


Figure 9.18. Add action

Here I have selected the folder "anotherPackage" as the destination location¹. Clicking on "Finish" adds the selected files to the Guvnor repository and creates an association between the local and Guvnor repository files. (Not that the wizard will not allow for overwrite of existing Guvnor repository files – another target location must be chosen.)

Compare with Version Action:

The Compare with Version action is enabled for one Guvnor repository associated file. This action first opens a wizard asking for the version for comparison (with the local file contents):



Resource Versions

Choose a version for versionHistoryTest.txt

Revision	Date	Author	Comment
6	2008-07-17T15:41:51	john	
5	2008-07-17T09:37:11	john	
4	2008-07-16T14:41:16	john	
3	2008-07-16T13:35:33	john	
2	2008-07-15T15:40:32	john	
1	2008-07-15T10:28:00	john	<from webdav>

Figure 9.19. Compare

Once the revision is selected, the action opens the Eclipse compare editor (read-only):

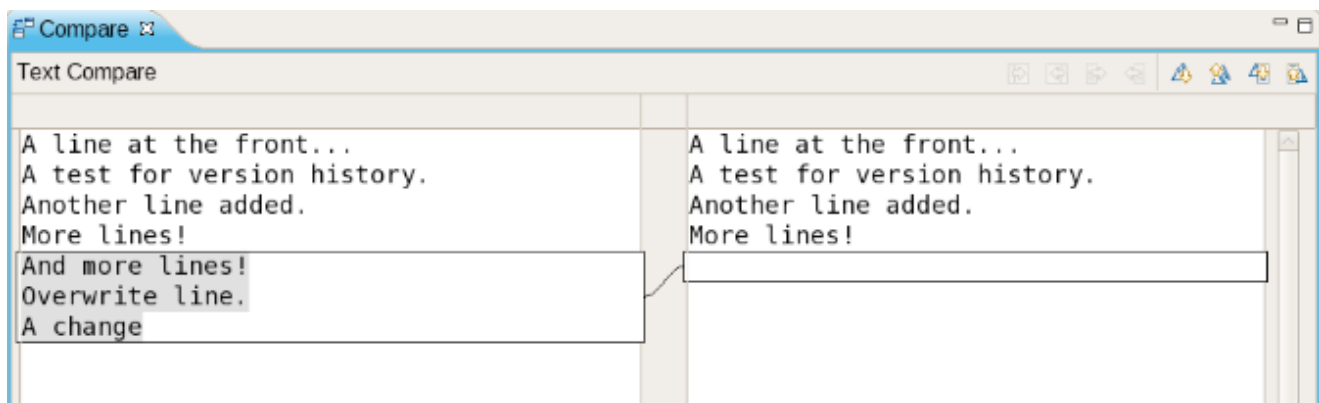


Figure 9.20. Compare

This editor uses Eclipse-standard comparison techniques to show the differences in the two versions. In cases where there are no differences, the editor will not open: rather, a dialog saying that there are no differences will appear.

Switch to Version Action:

The Switch to Version action is enabled for one Guvnor repository associated file. First the Switch to Version action prompts for selection of version:

Resource Versions

Choose a version for versionHistoryTest.txt

Revision	Date	Author	Comment
6	2008-07-17T15:41:51	john	
5	2008-07-17T09:37:11	john	
4	2008-07-16T14:41:16	john	
3	2008-07-16T13:35:33	john	
2	2008-07-15T15:40:32	john	
1	2008-07-15T10:28:00	john	<from webdav>

?
OK
Cancel

Figure 9.21. Versions

Once the version is selected, the Switch to Version action replaces the local file contents with those from the revision selected.

Delete Action:

The Delete action is enabled for one or more Guvnor repository associated files. After confirmation via a dialog, the Delete action removes the files in the Guvnor repository and deletes local metadata for the Guvnor repository association.

Disconnect Action:

The Disconnect action is enabled for one or more Guvnor repository associated files, and removes local metadata for the Guvnor repository association.

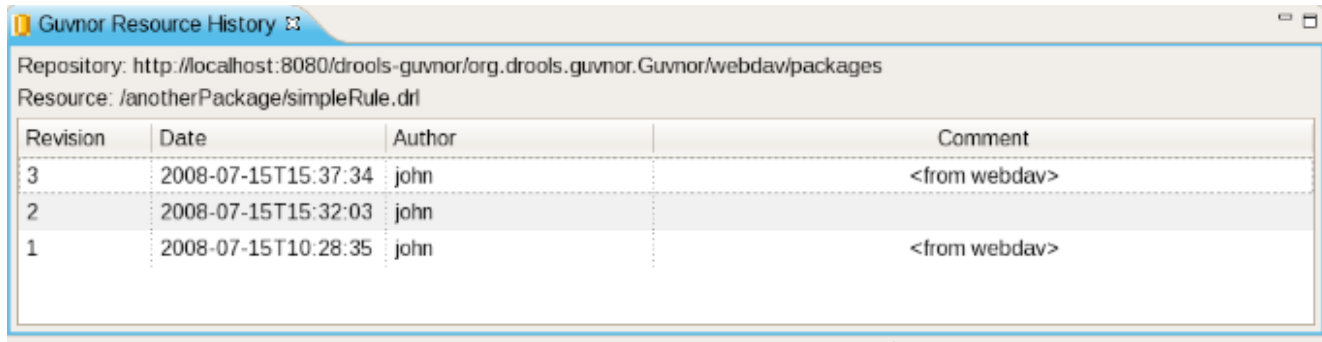
Guvnor Resource History View:

The Guvnor Resource History view should details about revision history for selected files, both local and those in Guvnor repositories. The initial state of this view is:

Guvnor Resource History			
Repository:			
Resource:			
Revision	Date	Author	Comment

Figure 9.22. History

The Guvnor Resource History view is populated by “Show History” actions in either the local “Guvnor” context menu or in the context menu for a Guvnor repository file in the Guvnor Repository Explorer. Once this action is performed, the Guvnor Resource History view updates to show the revision history:



Revision	Date	Author	Comment
3	2008-07-15T15:37:34	john	<from webdav>
2	2008-07-15T15:32:03	john	
1	2008-07-15T10:28:35	john	<from webdav>

Figure 9.23. History

Here we see that the file “simpleRule.drl” has three revisions. Double clicking on a revision row (or context menu “Open (Read only)”) opens an Eclipse read-only editor with the revision contents. (Note: You can also “Save As...” when a file is open in a read-only editor to save a local writable copy of the contents. Doing so, however, will not associate the file created with its Guvnor source.)

9.4.7. Importing Guvnor Repository Resources

In addition to the single file drag-and-drop from the Guvnor Repository Explorer view, the EGT also includes a wizard for copying one or more files from a Guvnor repository to the local workspace (and setting the association with the Guvnor repository). This wizard is available from the Eclipse Import , Guvnor, Resource from Guvnor and the Eclipse File, New, Other, Guvnor, Resource from Guvnor menu items. (Note: the wizard is identical but appears in both locations to accommodate users who tend to view this functionality as being in either category.) The first page of the wizard asks for the selection of the source Guvnor repository and gives the choice to create a new Guvnor repository connection (in which case the second page is the same as the Guvnor Connection wizard described above).

Select Guvnor repository location

Select an existing Guvnor repository location or create a new one



- ☐ Create a new Guvnor repository location
- ☒ Use an existing Guvnor repository location

```
http://localhost:8080/drools-guvnor/org.drools.guvnor.Guvnor/webdav  
http://localhost/cal/  
http://localhost/cal  
http://localhost:8080/drools-guvnor/org.drools.guvnor.Guvnor/webdav/packages
```



< Back

Next >

Finish

Cancel

Figure 9.24. Import

Once the source Guvnor repository is chosen, the wizard prompts for resource selection:

Select resources

Select resources to copy from the Guvnor repository

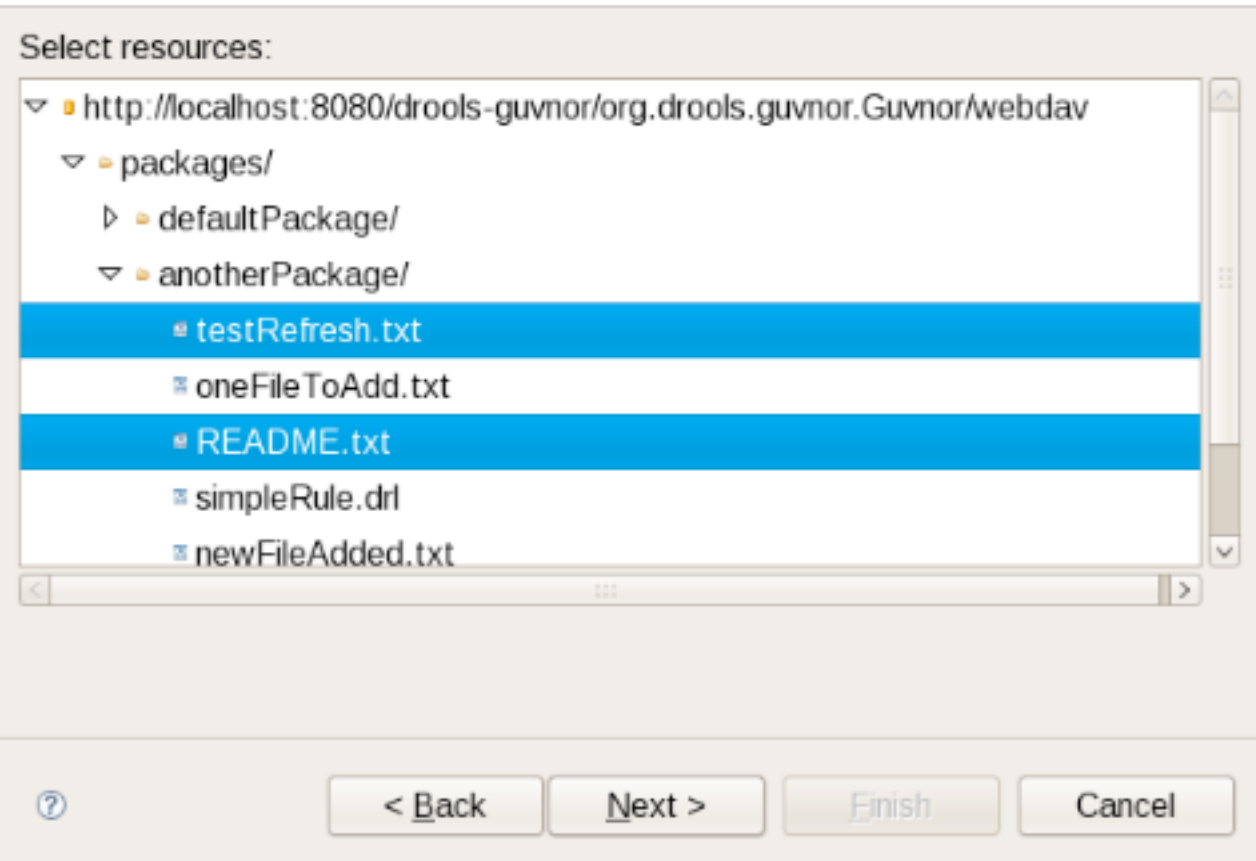


Figure 9.25. Import

Finally, the target location in the local workspace is chosen:

Select copy location

Select the destination location

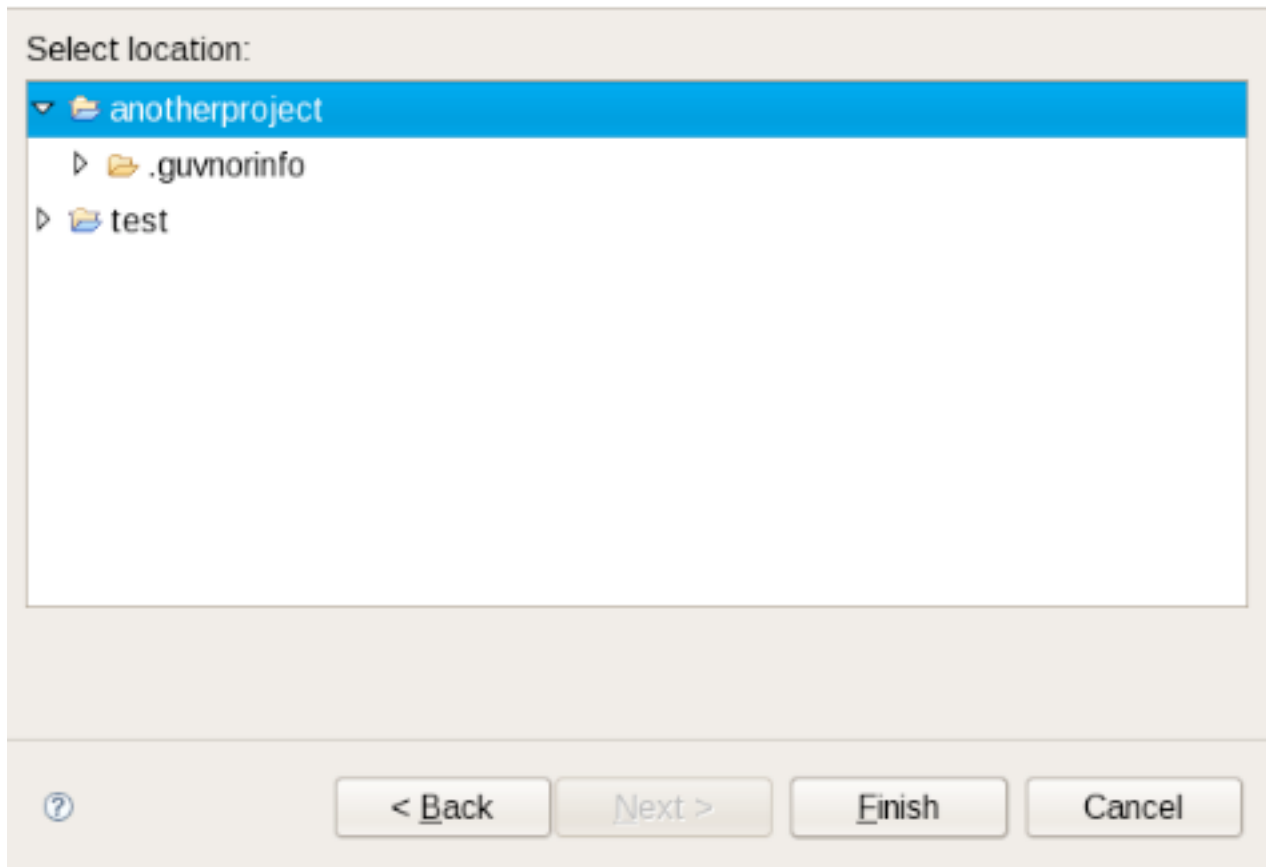


Figure 9.26. Import

On completion the wizard copies the selected files from the Guvnor repository to the local workspace. If a file with the same name already exists in the destination, the wizard uses the Eclipse standard “prompt for rename” dialog:

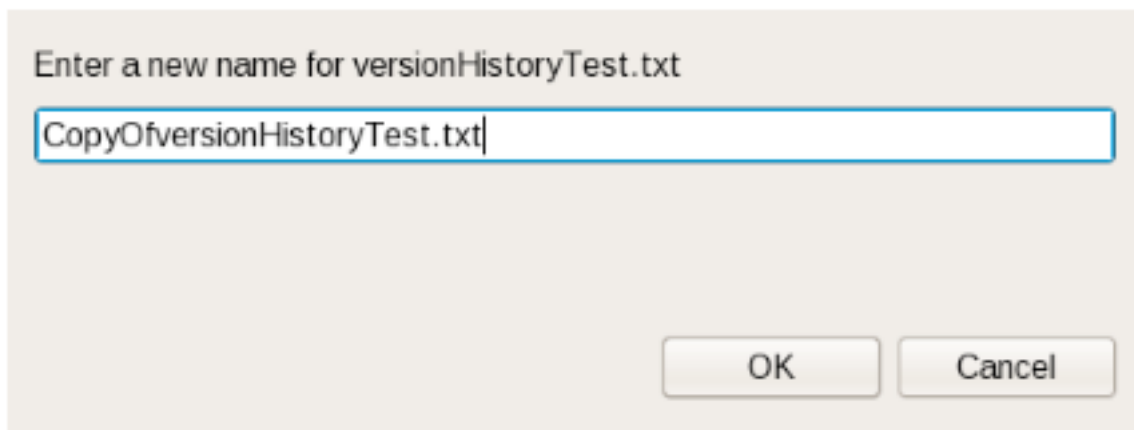


Figure 9.27. Copy

9.4.8. Guvnor plugin Preferences

The EGT provides a preference page in the “Guvnor” category:

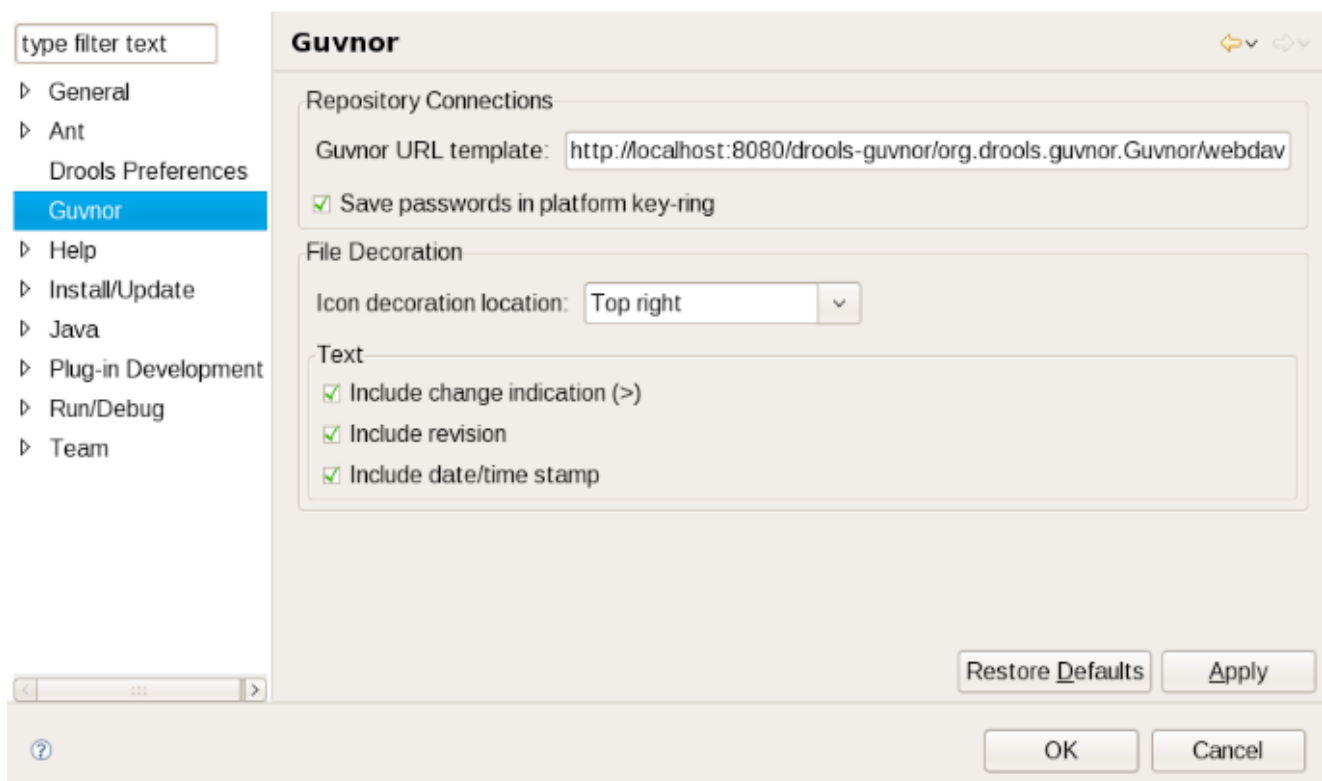


Figure 9.28. Preferences

The preferences cover two categories: Guvnor repository connections and local Guvnor repository resource decorations.

Guvnor Repository Connection Preferences

There are two preferences that can be set for Guvnor repository connections, and these are used when creating new connections. The first is a default Guvnor repository URL template, which can make it easier to create multiple similar connections by simply changing part of the field, such as the host name. The second is whether saving of authentication information in the Eclipse platform key-ring should be enabled by default. As with the Guvnor repository URL template, actually whether to save a specific instance of authentication information in the Eclipse platform key-ring can be determined when actually creating the connection. That is, both of these preferences are simply convenience values set to reasonable defaults.

Local Guvnor Repository Resource Decoration Preferences

The second category of preferences provided by the EGT deals with how decoration of local resources associated with Guvnor repository resources is presented. Since the Guvnor repository is not a substitute for a SCM, and since SCM tools in Eclipse tend to decorate local resources, it is useful to be able to control just how the EGT decorate its local resources to avoid messy conflicts with SCM packages. In the “File Decoration” section of the preference page, you can choose the location (top right, bottom right, top left, bottom left) of the decoration icon, or you can choose not to display it. In the “Text” section, you can format the Guvnor metadata that is appended to the file names: Whether to show an indicator (>) when the local file has changes not committed back to the Guvnor repository. Whether to show the revision number. Whether to show the date/time stamp. Any changes to these preferences take effect immediately upon clicking the “Apply” or “Ok” buttons.

Chapter 10. Embedding Guvnor In Your Application

As we already know, Guvnor provides a set of editors to author rules in different ways. According to rule's format a specialized editor is used. Some of the supported formats are: brl (guided editor), drl (plain editor), dsl (dsl editor), template (guided editor) and decision table (decision table editor).

One of the features introduced in Guvnor 5.2 was the ability to embed Guvnor's editor in your own (Web) Applications. So, if you want to edit rules, processes, decision tables, etc. In your applications without switch to Guvnor, you can.

This section covers all the steps you need to follow to embed Guvnor's editors in your Web Application

10.1. Getting Started

The Embedded Version of Guvnor's Editors lets you to use just the editor window inside your applications. So, basically what you need to do to is to embed just the editor you want to use in your web application using an iframe. In order to be able to invoke an Editor instance from an external application, Guvnor must be running: Standalone Editor is just a part of Guvnor and not a different application. The first step is to have Guvnor deployed and running in a web/application server.

Using the Embedded version of Guvnor's Editor you can create or edit assets only inside existing categories and packages. You must configure at least one category and package with a valid model inside Guvnor to start working with this feature.

10.2. Embedded Editor Entry-Point: StandaloneEditorServlet

Guvnor defines a single entry-point to embed any of its editors in a web application: StandaloneEditorServlet. This servlet is found in /standaloneEditorServlet URL, and according to the parameters you pass to it (parameters names and possible values are going to be explained later), you can open the editor in 3 different modes: BRL Edition Mode, Edition of Existing Assets Mode and Create New Asset Mode. So, if you want to embed a Guvnor Editor in your application you will need to perform a request to /standaloneEditorServlet URL. Once opened, you can interact with the editor using JavaScript.



Important

Because you need to invoke JavaScript in order to interact with the editor, Guvnor and your application must be running in the same server. Otherwise you will find cross-domain JavaScript invocation problems.

10.3. Edition Modes

Depending on the parameters used to invoke the Editor, you can use it in 3 different ways: *BRL Edition Mode*, *Edition of Existing Assets Mode* and *Create New Asset Mode*.

10.3.1. BRL Edition Mode

BRL Edition Mode is used if you want to use BRL code in your application. You can provide multiple BRL sources to the Editor, each of them will be converted to a temporal RuleAsset (a Guvnor's internal representation) and displayed in a separate editor.

These are the parameters you must use in the invocation of `StandaloneEditorServlet` to open Guvnor's Editor in BRL Edition Mode:

Table 10.1. BRL Edition Mode HTTP parameters:

Parameter Name	Explanation	Allow multiple values	Example
client	Defines the menu of the editor. The only supported value right now is <i>oryx</i>	false	oryx
packageName	The name of the package used to hold the created assets. The package must exist in Guvnor.	false	mortgages
categoryName	Each rule must have at least one category. The created rules will belong to this category. The category must exist in Guvnor	false	Home Mortgage
brlSource	The BRL source used by the editor. You can pass multiple brlSource parameters for multiple rules.	true	<pre> <rule> <name> Bankruptcy history </name> <modelVersion> 1.0 </pre>

Parameter Name	Explanation	Allow multiple values	Example
			<pre> </modelVersion> <attributes> ... </attributes> <lhs> ... </lhs> <rhs> ... </rhs> </rule> </pre>

All the assets created when using this mode are temporal. They are never going to be persisted in Guvnor. The purpose of this mode is to use just the Guided Editor and not Guvnor persistence layer. You can provide one or more initial brls, work on them using the Guided Editor and then retrieve the modified source from your application using javascript. Every time you want to edit a rule, you must provide its brl code.



Warning

BRL syntax is an internal format used by Guvnor. It is not supposed to be used in external applications, so drastic changes in its syntax can occur without any advise.

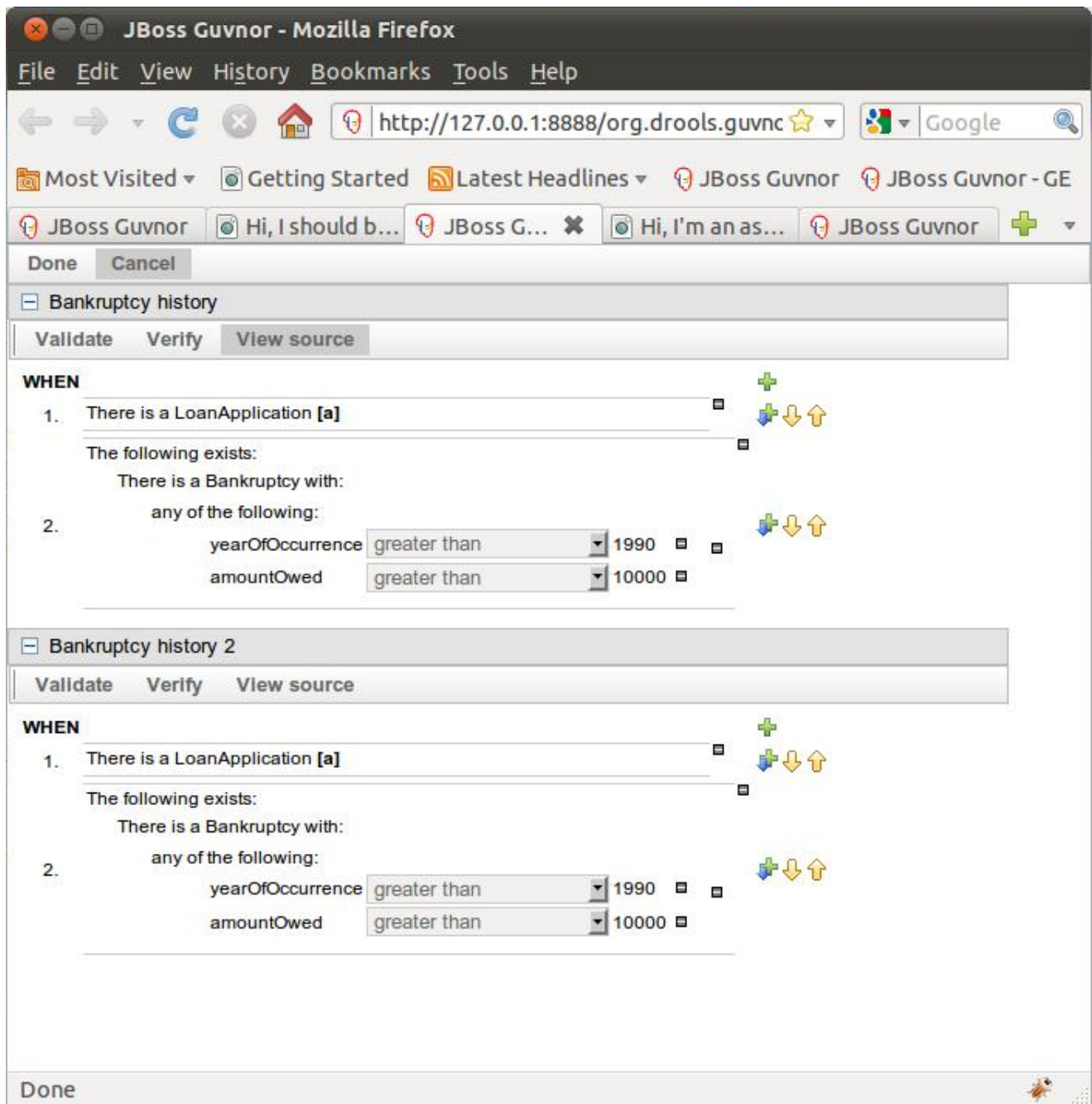


Figure 10.1. Embedded Editor with 2 BRL Rules

10.3.2. Edition of Existing Assets Mode

You can use this mode if you want to use the Standalone Editor for edit assets that already exist inside Guvnor. When editing existing assets, you will be able to save the changes in Guvnor as well as get the DRL and BRL code of them in your application.

The HTTP parameters involved in this mode are:

Table 10.2. HTTP parameters for Edition of Existing Assets Mode:

Parameter Name	Explanation	Allow multiple values	Example
client	Defines the menu of the editor. The only supported value right now is <i>oryx</i>	false	oryx
assetsUUIDs	The asset's UUID. Use multiple parameters for specify multiple assets.	true	968c9b3c-bc19-40ba-bb38-44435956ccee

**Note**

The asset's UUID could be found in Guvnor's UI or through REST-API.

When using this mode, you can edit assets from different types: like rules and decision tables.

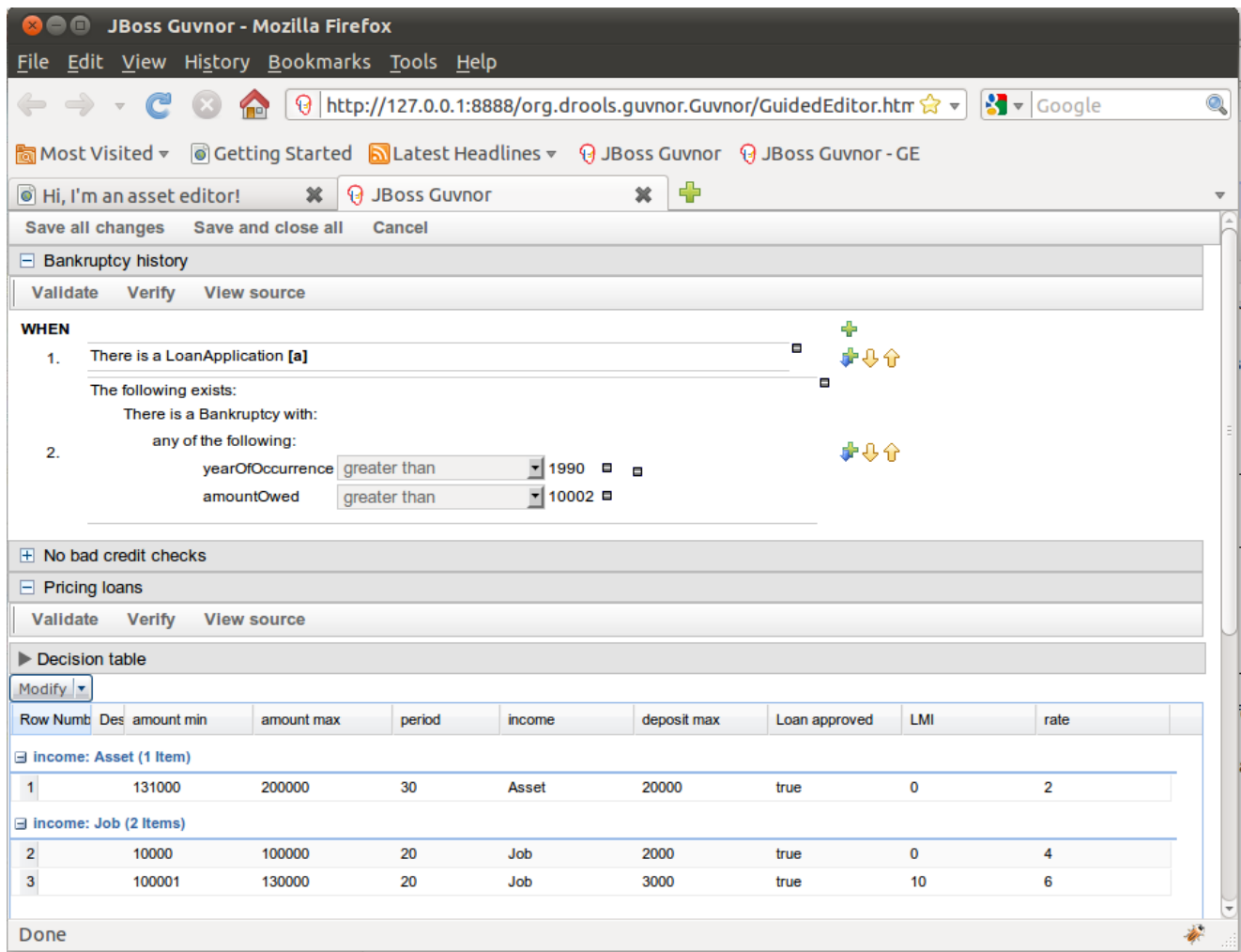


Figure 10.2. Embedded Editor with Multiple Assets

10.3.3. New Asset Mode

You can use this mode if you want to start a new asset from scratch. Assets created using this mode can be persisted inside Guvnor using the “Save all Changes” button. At this moment, you can only create one asset at a time using this mode.

The HTTP parameters involved in this mode are:

Table 10.3. HTTP parameters for New Asset Mode:

Parameter Name	Explanation	Allow multiple values	Example
client	Defines the menu of the editor. The only supported value right now is <i>oryx</i>	false	oryx

Parameter Name	Explanation	Allow multiple values	Example
packageName	The name of the package used to hold the created assets. The package must exist in Guvnor.	false	mortgages
categoryName	Each rule must have at least one category. The created rules will belong to this category. The category must exist in Guvnor	false	Home Mortgage
createNewAsset	Flag indicating that we want to start a new rule from the scratch. Must be <i>true</i>	false	true
assetName	The name for the asset to be created	false	New Rule
assetFormat	The format of the asset to be created. The format identifies the editor to be used.	false	<ul style="list-style-type: none"> • brl (default) • dsl • drl • gdst • template • Any format defined in <code>org.drools.guvnor.client.common.AssetF</code>

10.4. Extra HTTP parameters

We already covered all the HTTP parameters that must be used for each Edition Mode. You can combine these parameters with some others to customize the appearance and behavior of the editor.

10.4.1. Rule's Sections Visibility Parameters

When you edit or create rules using the Rule Guided Editor, you can choose which part of the rules would be visible (LHS, RHS and Attributes). By default, all these sections are visible in the editor.

You can specify which sections should be hidden using 3 different HTTP parameters:

Table 10.4. HTTP parameters for Rule's Sections Visibility:

Parameter Name	Explanation	Allow multiple values	Example
hideRuleLHS	Should the LHS of the rules be hidden?	false	true / false
hideRuleRHS	Should the RHS of the rules be hidden?	false	true / false
hideRuleAttributes	Should the Attributes of the rules be hidden?	false	true / false

10.4.2. Constraining Fact Types

When you edit or create rules in BRL format (using the Guided Editor), you can define a subset of the Fact Types defined in the rule's package. When authoring the rule, you will only see the Fact Types defined in that subset. This is the same concept as Working-Sets. In fact, a Working-Set will be created and applied on-the-fly using the provided subset.

If you want to define this set of Fact Types you could use this parameter:

Table 10.5. HTTP parameters for Fact Types Constraints:

Parameter Name	Explanation	Allow multiple values	Example
validFactType	The name of a valid Fact Type. This is just the class name and not the fqcn.	true	Bankruptcy

10.4.3. Use existing Working-Sets

When you edit or create rules in BRL format (using the Guided Editor), you can define which working-sets must be activated in the editor. In order to do this you must provide one or more UUID of existing working-sets. If you want to use this feature when editing multiple rules, all of them must belong to the same package where the working-sets you specify are defined.

In order to you want to define this set of Fact Types you could use this parameter:

Table 10.6. HTTP parameters to specify the Working-Sets to be applied:

Parameter Name	Explanation	Allow multiple values	Example
activeWorkingSetUUIDs	The UUID of an existing Working-Set	true	968c9b3c-bc19-40ba-bb38-44435956ccab

**Note**

The asset's UUID could be found in Guvnor's UI or through REST-API.

10.5. Interacting with the Editor

After the Editor is open, you can interact with it using JavaScript. The Editor defines a JavaScript object in the Window element where it is rendered. This object looks like this:

The *window.guvnorEditorObject* defines 5 functions that you can use to interact with it. *getDRL()* and *getBRL()* receive a callback function as parameter. This function will receive a String containing the DRL or BRL of the rules you are editing.

```
var guvnorEditorObject = {  
  getDRL: function (callbackFunction),  
  getBRL: function (callbackFunction),  
    registerAfterSaveAndCloseButtonCallbackFunction: function  
(callbackFunction),  
  registerAfterCancelButtonCallbackFunction: function (callbackFunction),  
  getAssetsUUIDs: function()  
}
```

The next 2 functions are for register callbacks for “Save”, “Done” and “Cancel” buttons. These callback functions don't accept any parameter. The last function is used to retrieve the UUIDs of the assets you are editing. This is very useful when you are creating a new rule asset and you don't know the UUID of it.

Part III. Administration Guide

This part covers installation and administration issues of Drools Guvnor.

Drools Guvnor is a web application that can run in multiple environments, and be configured to suit most situations. There is also some initial setup of data, and export/import functions covered.

Chapter 11. Installation

The Guvnor application is packaged as a `.war` file, which can be deployed to any application server (such as JBoss AS, ...) or servlet container (such as Tomcat, ...) out-of-the-box.

11.1. Installation step by step

Installation is simple: download, deploy, start and surf.

1. If you don't have an application server or servlet container, download and install one. For example, [download JBoss AS](http://www.jboss.org/jbossas/) [http://www.jboss.org/jbossas/].
2. **Download the Guvnor distribution** from [the download site](#). In the download zip, there's 1 war file per app server version, for example `guvnor-5.2.0-jboss-eap-5.1.war` for JBoss EAP 5.1. Use the war file best suited for your app server. Essentially there's little difference between those war files: mostly it's a matter of excluded jars which are already available on the app server.

If no war specifically for your app server exists yet, take the latest Tomcat war. It might require minor configuration tweaks. Consult our wiki for specific tips. The community has been able to make it run on various platforms. Patches (pull requests) to expand our war assemblies for another app server or version are welcome.

3. Optionally rename that war file to `guvnor.war` to have a nicer URL. For the rest of the manual we'll presume you've done this.
4. Optionally customize the configuration. First *explode* (unzip) the `war` file, and change any configuration and then *unexplode* (zip) it again.
5. **Deploy the war file** by copying it into the deployment directory of the app server. For JBoss AS 5 and 6, that directory is `server/default/deploy`. Alternatively you can first explode (unzip) the war file and copy that exploded directory. Note: in JBoss AS, the exploded directory name must end with the suffix `.war`.
6. **Start the app server**. For JBoss AS 5 and 6, run `$JBASS_AS_HOME/bin/run.sh` (or `run.bat`).
7. **Surf to the Guvnor webapp**. This will probably be at `localhost` on port 8080, for example at <http://localhost:8080/guvnor/> or (if you haven't rename the war file) at something like <http://localhost:8080/guvnor-5.2.0-jboss-eap-5.1/>.

11.2. Supported and recommended platforms

Guvnor runs in any application server or servlet container that supports Java SE 5 (JEE 5 is not required), this includes JBoss AS, Tomcat, Jetty and many more.

JBoss AS is the recommended application server, because it is actively tested/developed on it. If you're looking for mission critical, enterprise support, [take a look BRMS subscription](http://www.jboss.com/products/platforms/brms/) [http://www.jboss.com/products/platforms/brms/].

Chapter 12. Database configuration

Guvnor uses the JCR standard for storing assets such as rules. The default implementation is Apache Jackrabbit, <http://jackrabbit.apache.org>. This includes an out of the box storage engine/database, which you can use as is, or configure to use an existing RDBMS if needed.

12.1. Changing the location of the data store

Assuming you are using one of the JBoss platforms, running Guvnor for the first time will create a database in the `bin/` directory of the application server. There you will find the default Jackrabbit configuration file, namely `repository.xml`, and a `repository` directory which contains your repository data. Both of these are created automatically for you.

The location of the data store should be a secure location, that is backed up. The default location may not be suitable for this, so the easiest way is to set a more suitable location. If you want to change this, please make sure you have stopped Guvnor (i.e. stopped the app server or undeployed the application).

To change the location, unzip the Guvnor WAR file, and locate the `beans.xml` file in the `WEB-INF` directory. This is a JBoss Seam configuration file (Seam 3 is the framework used) which allows various parts of the system to be customized. When you have located the `beans.xml` file, you should see something like the following:

```
<guvnorRepository:GuvnorBootstrapConfiguration>
  <s:modifies/>

  <!-- JackRabbit -->
  <guvnorRepository:properties>
    <s:entry><s:key>org.drools.repository.configurator</s:key><s:value>org.drools.repository.jackrabbit.JackrabbitRepositoryConfigurator</s:value></s:entry>
    <!-- the root directory for the repo storage the directory must exist. -->
    <!--<s:entry><s:key>repository.root.directory</s:key><s:value>/opt/yourpath</s:value></s:entry>-->
  </guvnorRepository:properties>

  ...
</guvnorRepository:GuvnorBootstrapConfiguration>
```

Find the component with a name of `GuvnorBootstrapConfiguration` and its section for JackRabbit configuration, then the key property with the name of `repository.root.directory`.

If you un-comment this key element (as in the example above it is commented out), you can set whatever file-system path you need for the repository data to be stored in. You can also use this to move the repository around. In that case, when you have set the location in the `beans.xml`

you can simply move the `repository.xml` AND the repository directory to the new location that you set in the `beans.xml`.

If there is no `repository.xml` configuration file, or the repository directory at the location specified (or in the default location) then Guvnor will create new empty ones.

There are many more options which can be configured in the `repository.xml`, but for the most part, it is not recommended to change the defaults.

12.2. Configuring Guvnor to use an external RDBMS

In some cases it may be a requirement that you use an external RDBMS, such as Oracle, MySQL, or Microsoft SQL Server as the data store - this is permitted and recommended as storing your repository data in an external RDBMS is much more reliable than the default file-system storage option. The JackRabbit `repository.xml` file contains the information where your repository data is stored, so changes to this file are necessary for RDBMS setup. You have two options to make changes to `repository.xml`, namely make all changes manually, or use the Guvnor Repository Configuration Manager.

If you opt for the manual configuration, the easiest thing to do is to start up Guvnor with defaults (or with a suitable `repository.root.directory` directory as specified above) to let it generate the default `repository.xml`. Locate the `repository.xml` file that was generated, and open it - it will be annotated with comments describing many of the different options. From here on, you will need to know a little about Jackrabbit Persistence managers, <http://wiki.apache.org/jackrabbit/PersistenceManagerFAQ>. There are a few persistence managers, some are database specific (eg Oracle). There is a `SimpleDBPersistenceManager` which works with any database that supports JDBC - you also specify the database type, so it uses the specific DDL to create the table structure (all major databases are supported). After you have added your configuration options, start the Guvnor application again. Guvnor will then create the database tables the first time it is started up if it is running against a fresh (empty) RDBMS - so its important to note that the user credentials supplied have permissions to create tables (at least initially, on first run, after that they could be locked down).

Using the Repository Configuration Manager in Guvnor is often a lot easier and a less error-prone options to make the necessary configuration changes in `repository.xml`. With Guvnor application running, select the Administration tab in the left-hand-side navigation bar, then select the Repository Configuration link.

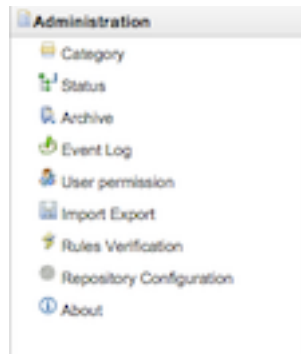


Figure 12.1. Finding the Repository Configuration Manager in the Administration section

Repository Configuration Manager includes template configuration files for many external RDBMS types. The first thing you have to do is select the RDBMS type from the dropdown menu and select if you are using JNDI to look up your data source or not.

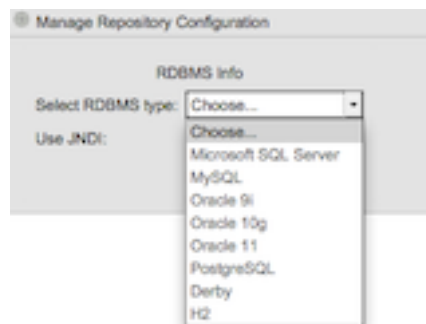


Figure 12.2. Select RDBMS type

If you opt to use JNDI, you have to enter the JNDI name configured in your deployed data source. Otherwise you need to enter your RDBMS information.

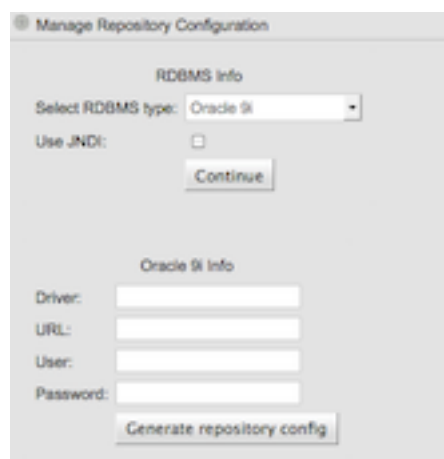


Figure 12.3. RDBMS information

At this point you are ready to generate your `repository.xml` give your RDBMS information. Click on the "Generate repository config" button.

Manage Repository Configuration

RDBMS Info

Select RDBMS type: Oracle 9i

Use JNDI: ☐

Continue

Oracle 9i Info

Driver: com.my.Driver

URL: myUrl

User: myUser

Password: *****

Generate repository config

Save Configuration

```
<?xml version="1.0"?>
<!--
Licensed to the Apache Software Foundation (ASF) under one or more
contributor license agreements. See the NOTICE file distributed with
this work for additional information regarding copyright ownership.
The ASF licenses this file to You under the Apache License, Version 2.0
(the "License"); you may not use this file except in compliance with
the License. You may obtain a copy of the License at

    http://www.apache.org/licenses/LICENSE-2.0

Unless required by applicable law or agreed to in writing, software
distributed under the License is distributed on an "AS IS" BASIS,
WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied.
See the License for the specific language governing permissions and
limitations under the License.
-->
<!DOCTYPE Repository PUBLIC "-//The Apache Software Foundation//DTD Jackrabbit 1.4//EN"
    "http://jackrabbit.apache.org/dtd/repository-1.4.dtd">
<!-- Example Repository Configuration File -->
<Repository>
  <!--
    virtual file system where the repository stores global state
    (e.g. registered namespaces, custom node types, etc.)
  -->
  <!--
    <FileSystem class="org.apache.jackrabbit.core.fs.local.LocalFileSystem">
      <param name="path" value="{rep.home}/repository"/>
    </FileSystem>
  -->
  <FileSystem class="org.apache.jackrabbit.core.fs.db.OracleFileSystem">
    <param name="driver" value="com.my.Driver"/>
    <param name="url" value="myUrl"/>
  </FileSystem>
</Repository>
```

Figure 12.4. Generated repository.xml

You can download the generated `repository.xml` file by clicking the "Save Configuration" button, or copy/paste the generated text manually. Replace your existing `repository.xml` with the generated one and restart Guvnor for the changes to get picked up.

12.3. Searching and indexing, Version storage

Jackrabbit has a separate storage area for version storage (as over time, the number of old versions will increase, yet it should not slow down the performance of the main data store). The version storage also has its own persistence manage configuration in the `repository.xml`, but for most purposes you can use the same database as the main storage (just with a different schema object prefix - ie in your database, all the version data will be prefixed with `version_` but otherwise in the same tablespace). See the `repository.xml` for more details of this.

Lucene is used to provide indexing across the semi structured data, and across versions. This indexing is generally best stored on a filesystem, local to Guvnor (as per the default in the `repository.xml`) - in most cases the default is fine.

Chapter 13. Switch from JackRabbit to ModeShape

Guvnor supports running on either JackRabbit and ModeShape as the underlying JCR-2.0 implementation. By default Guvnor ships using JackRabbit. However if you want to switch to using ModeShape then you need to install ModeShape as a service in JBossAS-5.x. Check the ModeShape project and download and install ModeShape 'kit' version 2.5.0 or later. After installing the kit, you should have a `modeshape-service.jar` directory in your deploy directory. Since ModeShape 2.5.0 only support deployment to JBoss-5.x, make sure to also use the `guvnor-5.2.0-jboss-eap-5.1.war`. Now we can remove some jars from the guvnor WAR that are no longer needed, and in fact will cause classloading issues if you don't remove them:

```
[localhost]$ rm -f WEB-INF/lib/jackrabbit-*
[localhost]$ rm -f WEB-INF/lib/hibernate-* WEB-INF/persistence-api-1.0.jar
WEB-INF/lucene-*.jar
[localhost]$ rm -f WEB-INF/lib/jcr-2.0.jar
```

Next you need to edit the `WEB-INF/beans.xml` file to switch over to ModeShape. Comment out the JackRabbit section and uncomment the ModeShape section:

```
<guvnorRepository:GuvnorBootstrapConfiguration>
  <s:modifies/>

  <!-- JackRabbit -->
  <guvnorRepository:properties>
    <s:entry><s:key>org.drools.repository.configurator</s:key><s:value>org.drools.repository.jackrabbit.JackrabbitRepositoryConfigurator</s:value></s:entry>
    <!-- the root directory for the repo storage the directory must exist. -->
    <!--<s:entry><s:key>repository.root.directory</s:key><s:value>/opt/yourpath</s:value></s:entry>-->
  </guvnorRepository:properties>

  <!-- ModeShape -->
  <!--
    passwords for the background users (logInAdmin and mailman), these need
    to match the setting
    you provided for JAAS (used by ModeShape only).
  -->
  <!--<guvnorRepository:properties>-->
    <!--<s:entry><s:key>org.drools.repository.configurator</s:key><s:value>org.drools.repository.modeshape.ModeShapeRepositoryConfigurator</s:value></s:entry>-->
```

```
<!--<s:entry><s:key>org.modeshape.jcr.URL</s:key><s:value>jndi:jcr/local?
repositoryName=brms</s:value></s:entry>-->
    <!--<s:entry><s:key>org.drools.repository.secure.passwords</
s:key><s:value>>false</s:value></s:entry>-->
        <!--<s:entry><s:key>org.drools.repository.logInAdmin.password</
s:key><s:value>logInAdmin</s:value></s:entry>-->
            <!--<s:entry><s:key>org.drools.repository.mailman.password</
s:key><s:value>mailman</s:value></s:entry>-->
        <!--</guvnorRepository:properties>-->

</guvnorRepository:GuvnorBootstrapConfiguration>
```

Note that you can use encrypted passwords by setting the `org.drools.repository.secure.passwords` setting to `true`. To encrypt a password use:

```
[localhost]$ java -cp client/jboss-logging-spi.jar:common/lib/jbosssx.jar
org.jboss.resource.security.SecureIdentityLoginModule <password>
```

ModeShape does not support 'trusted' access like JackRabbit does, and by default uses JAAS for authentication and authorization. For more detail on Guvnor and Security see the next section about Security. To use JAAS and the modeshape policy comment out the `defaultAuthenticator` section and uncomment the `jaas-configuration` section, and change the policy name from 'other' to 'modeshape':

```
<security:IdentityImpl>
  <s:modifies/>

  <!-- No real authentication: demo authentication for demo purposes -->

<security:authenticatorClass>org.drools.guvnor.server.security.DemoAuthenticator</
security:authenticatorClass>

  <!-- JAAS based authentication -->
      <!--<security:authenticatorName>jaasAuthenticator</
security:authenticatorName>-->

      <!-- IDM based authentication (supports LDAP, see Seam 3 and PicketLink
IDM documentation) -->
          <!--
<security:authenticatorClass>org.jboss.seam.security.management.IdmAuthenticator</
security:authenticator>-->
      </security:IdentityImpl>

  <!--<security:jaas.JaasAuthenticator>-->
      <!--<s:modifies/>-->
      <!--
```

```
The following one will use the jaas configuration called "other",
which in jboss AS means you can use properties files for users.
-->
<!--<jaasConfigName>other</jaasConfigName>-->
<!--</security:jaas.JaasAuthenticator>-->
```

You may have noticed the settings of two passwords in the modeshape property settings for the 'admin' and 'mailman' users. These users are used by guvnor to perform background tasks. Now that we are no longer allowing for anyone to run as 'guest', we need to assign these two users to the modeshape users and roles files. Open the `conf/props/modeshape-users.properties` file and add the mailman and admin users,

```
admin=admin
mailman=mailman
```

Finally open the `conf/props/modeshape-roles.properties` file and add the admin and mailman roles,

```
admin=connect,admin
mailman=connect,readonly,readwrite
```

By default JackRabbit uses InMemory storage, which is configured in the `modeshape-service.jar/modeshape-config.xml`. To change this we recommend reading the modeshape documentation. To use a referenced JNDI data source, replace the `<mode:source></mode:source>` segment with the following:

```
<mode:source jcr:name="store" mode:classname="org.modeshape.connector.store.jpa.JpaSource"
mode:dataSourceJndiName="your JNDI name"
mode:model="Simple"
mode:dialect="org.hibernate.dialect.HSQLDialect"
mode:referentialIntegrityEnforced="true"
mode:largeValueSizeInBytes="10000"
mode:retryLimit="3"
mode:compressData="false"
mode:predefinedWorkspaceNames="default,system"
mode:showSql="false"
mode:autoGenerateSchema="update"
mode:creatingWorkspacesAllowed="true"
mode:defaultWorkspaceName="default" />
```

Alternatively you can connect directly to a JDBC data source, use the same `<mode:source>` fragment as for JNDI except replace the `mode:dataSourceJndiName` attribute with these attributes:

```
mode:driverClassName=org.hsqldb.jdbcDriver
mode:username=sa
mode:password=
mode:url=jdbc:hsqldb:mem:target
mode:maximumConnectionsInPool=5
```

For purposes of illustration, the HSQL DB is being used, but simply replace the attribute values with the appropriate driver class name, username, password, and database URL. Don't forget to add a dependency to your JDBC jar, so the JDBC driver available in the classpath.

Chapter 14. Security -

Authentication and basic access

Please note that giving someone access to Guvnor indicates a level of trust. Being able to editing and build rules is providing a great deal of power to a user. Thus you should not open up Guvnor to your entire organization - but instead to a select few. Use https (http with TLS/SSL) where ever possible, even internally in a company network this is a good idea. Use this power wisely - this not a "run of the mill" application that provides read/write access to a database, but something much more power. Just imagine you are spider man - with great power comes great responsibility (of course even more so for super man).

Security is configured by using the `beans.xml` file in the war file. To customize this, you will need to unzip the WAR file, and locate the `beans.xml` file which is in the `WEB-INF` directory.

The JAAS standard is used as the underlying authentication and authorization mechanism, the upshot of which means its very flexible and able to integrate into most existing environments.

Out of the box, Guvnor shows a login screen, but no security credentials are enforced - the user name is used, but no password check is performed. To enforce authentication, you need to configure it to use an appropriate user directory, you may have Active Directory or similar already.

In the `beans.xml` file, you should located a security configuration section like the following:

```
<security:IdentityImpl>
  <s:modifies/>

  <!-- No real authentication: demo authentication for demo purposes -->
<security:authenticatorClass>org.drools.guvnor.server.security.DemoAuthenticator</
security:authenticatorClass>

  <!-- JAAS based authentication -->
                                <!--<security:authenticatorName>jaasAuthenticator</
security:authenticatorName>-->

  <!-- IDM based authentication (supports LDAP, see Seam 3 and PicketLink
IDM documentation) -->
                                <!--
<security:authenticatorClass>org.jboss.seam.security.management.IdmAuthenticator</
security:authenticator>-->
</security:IdentityImpl>
```

As you can see from above, the 2 "out of the box" options are pass through - which means any user is allowed in, or bypassed, in which case there is no login screen (e.g. you may be securing access to the app via a web server anyway).

14.1. Using your containers security and LDAP

Every application server supports advanced configurations which can work with your existing security infrastructure. The case of JBoss AS will be shown here as an example.

```
<security:identity authenticate-method="#{authenticator.authenticate}"
    jaas-config-name="other"/>
```

This will use the `other` JAAS config in JBoss AS. If you look in `jboss-as/server/default/conf` you will see a `login-config.xml` file. This file contains various configurations. If you use `other` like the one above, then it will look for `users.properties` and `roles.properties` in the `conf/` directory for usernames and passwords to authenticate against. This is maintainable only for a fixed small number of users.

LDAP is perhaps the most popular choice for larger enterprises. Here is an example that works with Active Directory. You can get much more information on how to configure JBoss AS for all scenarios with LDAP from <http://wiki.jboss.org/wiki/Wiki.jsp?page=LdapLoginModule> and <http://wiki.jboss.org/wiki/Wiki.jsp?page=LdapExtLoginModule>.

```
<application-policy name="brms">
    <authentication
        <login-
module code="org.jboss.security.auth.spi.LdapExtLoginModule" flag="required" >
        <!--
            Some AD configurations may require searching against
            the Global Catalog on port 3268 instead of the usual
            port 389. This is most likely when the AD forest
            includes multiple domains.
        -->
        <module-option name="java.naming.provider.url">ldap://
ldap.jboss.org:389</module-option>
        <module-option name="bindDN">JBoss\someadmin</module-option>
        <module-option name="bindCredential">password</module-option>
        <module-option name="baseCtxDN">cn=Users,dc=jboss,dc=org</module-option>
        <module-option name="baseFilter">(sAMAccountName={0})</module-option>

        <module-option name="rolesCtxDN">cn=Users,dc=jboss,dc=org</module-
option>
        <module-option name="roleFilter">(sAMAccountName={0})</module-option>
        <module-option name="roleAttributeID">memberOf</module-option>
        <module-option name="roleAttributeIsDN">true</module-option>
        <module-option name="roleNameAttributeID">cn</module-option>
```

```
<module-option name="roleRecursion">-1</module-option>
<module-option name="searchScope">ONELEVEL_SCOPE</module-option>
</login-module>
</authentication>
</application-policy>
```

To use the above, you would put `jaas-config-name="brms"` in the `security:identity` tag in the `beans.xml` for Guvnor.

Similar configuration examples can be found for other directory services.

LDAP isn't the final word, you can use JDBC against a database of user name, or you can write your own login module to use any sort of weird and wonderful authentication and authorization systems that you may have to deal with (that would be an extreme case, but its possible). Refer to JBoss AS documentation (or documentation for your existing application server).

Chapter 15. Fine grained permissions and security

The above section talks about establishing identity and access for users. This section talks about granting specific permissions to these users (to control data visibility and access). This can be used to partition data, or to control access for "non power users" which can limit the damage they can do.

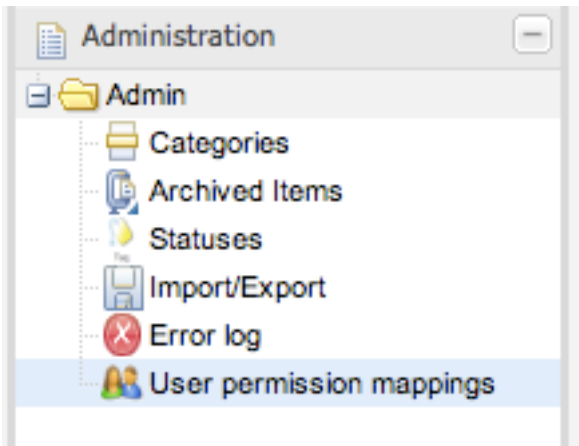


Figure 15.1. Administer user permissions

A common need and desire of the web interface of Guvnor is to be able to have users of different technical abilities interact with it. Another need is to be able to allocate people different sets of data to "own".

Typically users identities are managed in a centralized directory - application servers can integrate with these directories (e.g. active directory, LDAP) so users to Guvnor can be authenticated without having to create another duplicate identity. It is also possible (thanks to JAAS) to define what users have the "admin" role for Guvnor (note that an Admin user of Guvnor doesn't have to really be a system administrator). Further to this, Guvnor augments this identity with data specific permissions, which are managed in Guvnor itself.

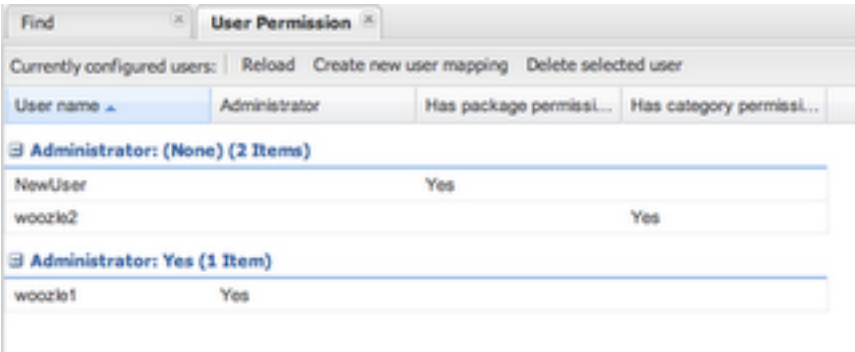


Figure 15.2. User listing

Note that the above users identities are not stored in Guvnor, only their permission mappings are which are specific to Guvnor.

There are really two system wide roles: Users who are Administrators and users who are not. Administrators can see and do anything. Out of the box, the permission system is turned off, and every user is an administrator (this is pretty much how things used to work). There is also a system setting in `beans.xml` that can turn the permissions system on and off (so people can manually override if needs be). A administrator can also give other users admin rights, regardless of their roles in the external directory service.

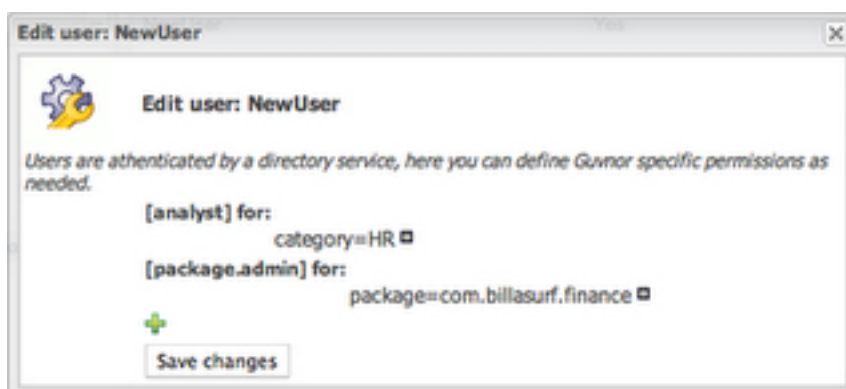


Figure 15.3. Editing

There are several types of permissions: Per package: Package Administrator ("owns" a package - can deploy etc, but has no administrative rights to the system). Package developer - this permissions allows users to create new items, edit etc - but only at the package level (not deploy). They can also run and create tests. Package readonly - well this one is pretty obvious. Per Category: This is the "interesting" one - as assets (rules) can be tagged with multiple categories, you can use these to assign permissions to an "analyst" type of user. A user can be assigned multiple categories. A user can then edit and view any asset that is tagged in that category (regardless of package). A user that only has category permissions will not be shown any package views or details, and will only see the simple categories view. This allows administrators and managers to control exactly what these users can and can't see. Note that per category permissions can also be set as "read only" so a user can view all the assets in a category, but not make changes to them.

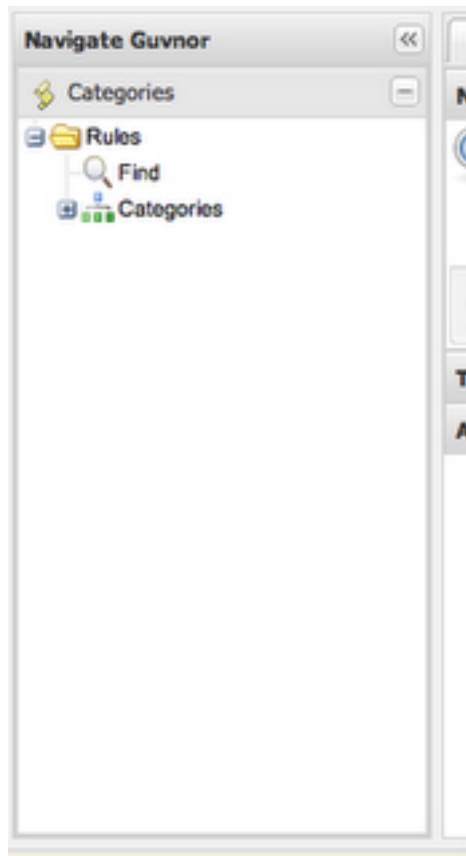


Figure 15.4. The analyst view

The per category "analyst" permissions are quite useful - you can also augment their permissions with a specific package (so on top of their category rights, they can see and play with a particular package - which may be used as a "practice" area, or test area for instance). This provides a few ways to manage permissions in a coarse or fine grained way, as suits the different types of users.

15.1. Enabling fine grained authorization

By default authorization is NOT enabled. To enable it, edit the `beans.xml` file in the `WEB-INF` directory:

```
<component name="org.jboss.seam.security.roleBasedPermissionResolver">;
  <s:modifies/>
  <property name="enableRoleBasedAuthorization">true</property>
</component>
```


Chapter 16. Data management

16.1. Backups

How backups are performed is dependent on what persistence manager scheme you are using. Using the default one - then its a matter of backing up the repository directory (wherever you have it configured to be). Restoring it is simply a matter of copying across the repository directory.

Ideally you will either stop Guvnor application while a file backup is being done, or ensure that no one is using it.

In the case of using an external database (e.g. Oracle, MySQL), then the normal scheme can apply that you would use to backup those database (you do back them up, right?). In this case, when restoring, it is also a good idea to clear the indexes (delete the directory where the indexes are) so they are created fresh from the data (and thus guaranteed to be in sync).

16.2. Repository Data Migration

It is often needed to migrate your existing repository from one persistence manager schema to another. A typical scenario for this case is if you have existing rule assets in a repository using the default file-system configuration and would like to move to storing your existing data to a RDBMS. In these cases you can use the drools-ant JackrabbitMigrationAntTask which can easily convert all your repository data from one repository configuration to another repository configuration. Example configuration for this ant task can be:

```
<project default="migraterepo">
  <path id="migration.classpath">
    <pathelement path="${classpath}" />
    <fileset dir="/Users/tihomir/development/droolsjbpm/jboss-4.2.3.GA/server/
default/deploy/drools-guvnor.war/WEB-INF/lib">
      <include name="**/*.jar"/>
    </fileset>
    <filelist refid="drools-ant" />
    <filelist refid="db-driver-jars" />
  </path>

  <filelist id="db-driver-jars" dir="/Users/tihomir/development/droolsjbpm/
jboss-4.2.3.GA/server/default/lib">
    <file name="mysql-connector-java-5.1.11-bin.jar" />
  </filelist>

  <filelist id="drools-ant" dir="lib">
    <file name="drools-ant-${project.version}.jar" />
  </filelist>
</project>
```

```
<taskdef name="migrate" classname="org.drools.contrib.JackrabbitMigrationAntTask"
    classpathref="migration.classpath" />

<target name="migraterepo">
    <record name="migration-log.txt"/>
    <migrate verbose="true"
        sourcedir="/Users/tihomir/development/droolsjbpm/jboss-4.2.3.GA/bin/
repository/"
        sourceconfig="/Users/tihomir/development/droolsjbpm/jboss-4.2.3.GA/bin/
repository.xml"
        targetdir="/Users/tihomir/demo-jrmigration/targetrepo/"
        targetconfig="/Users/tihomir/demo-jrmigration/targetrepo/
repository.xml" />
    </target>
</project>
```

In the above scenario JackrabbitMigrationAntTask is going to migrate all repository data configured in `repository.xml` defined in the `sourcedir` attribute, to the repository configured in `repository.xml` defined in the `targetconfig` attribute. Note that this data migration is a full migration, which means it migrates the entire repository which also makes it a good tool to use for backups as well. The blog post <http://blog.athico.com/2011/03/using-drools-ant-to-migrate-back-up.html> contains a video showing a full example on how to use the JackrabbitMigrationAntTask for repository migration.

16.3. Adding your own logos or styles to Guvnor web GUI

To achieve, this, you can "explode" the deployment WAR file, and locate the `Guvnor.html` file, which will look something like the following:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN" "http://
www.w3.org/TR/html4/loose.dtd">
<html>
<head>
    <!-- Note you can append #asset=UUID to the end of the URL to preload a
    given asset.

    Also, if you appent #asset=UUID&amp;nochrome it will only show the asset
    without all the GUI "chrome"

    To select a locale, specify &amp;locale=en_US at the end of the URL to
    pick the appropriate bundle.
    -->
    <meta http-equiv="Content-Type" content="text/html; charset=UTF-8">
    <meta http-equiv="X-UA-Compatible" content="IE=EmulateIE7" />
    <title>JBoss Guvnor</title>
```

```
.....  
  
</body>  
</html>
```

Note that the above `Guvnor.html` file is fairly small (as most of the work is done by the GWT - the GUI is built dynamically in the browser). The parts you can customize are the style sheet - you can either edit the `Guvnor.css` (or better yet, take a copy, and change the style to be what you need), the "shortcut icon" (its what shows in the address bar in the browser etc - also change the "icon" link to be the same so it works in IE), and the header logo. The rest should be left as is, to allow the GWT components to be loaded and attached to the page. This html page is loaded only once by the browser when the user accesses Guvnor web GUI.

The best way to customize is to take a copy of the `Guvnor.html` file and then edit. You can also change the URL by editing the `web.xml` via the normal means.

16.4. Import and Export

A JCR standard export/import feature is available from the Admin part of the web interface.

This will export the entire repository to an XML format as defined by the JCR standard.

In the case of import, it will clear any existing content in the database.

This is *not* a substitute for backup but can be useful when migrating. It is important to note that version history is not exported this way, only the current state. Hence it is still recommended that a formal backup regime be used at all times on the repository database itself.

Note that when importing repositories with many thousands of items, extra memory will be required when performing the import.

Chapter 17. Architecture

This section covers the technical aspects of Guvnor, it is not necessary to use this if you are integrating or an end user of the application. However, Drools is open source, so build instructions form part of the manual.

You may want to build from source if you want to re-use components, or embed the application within another.

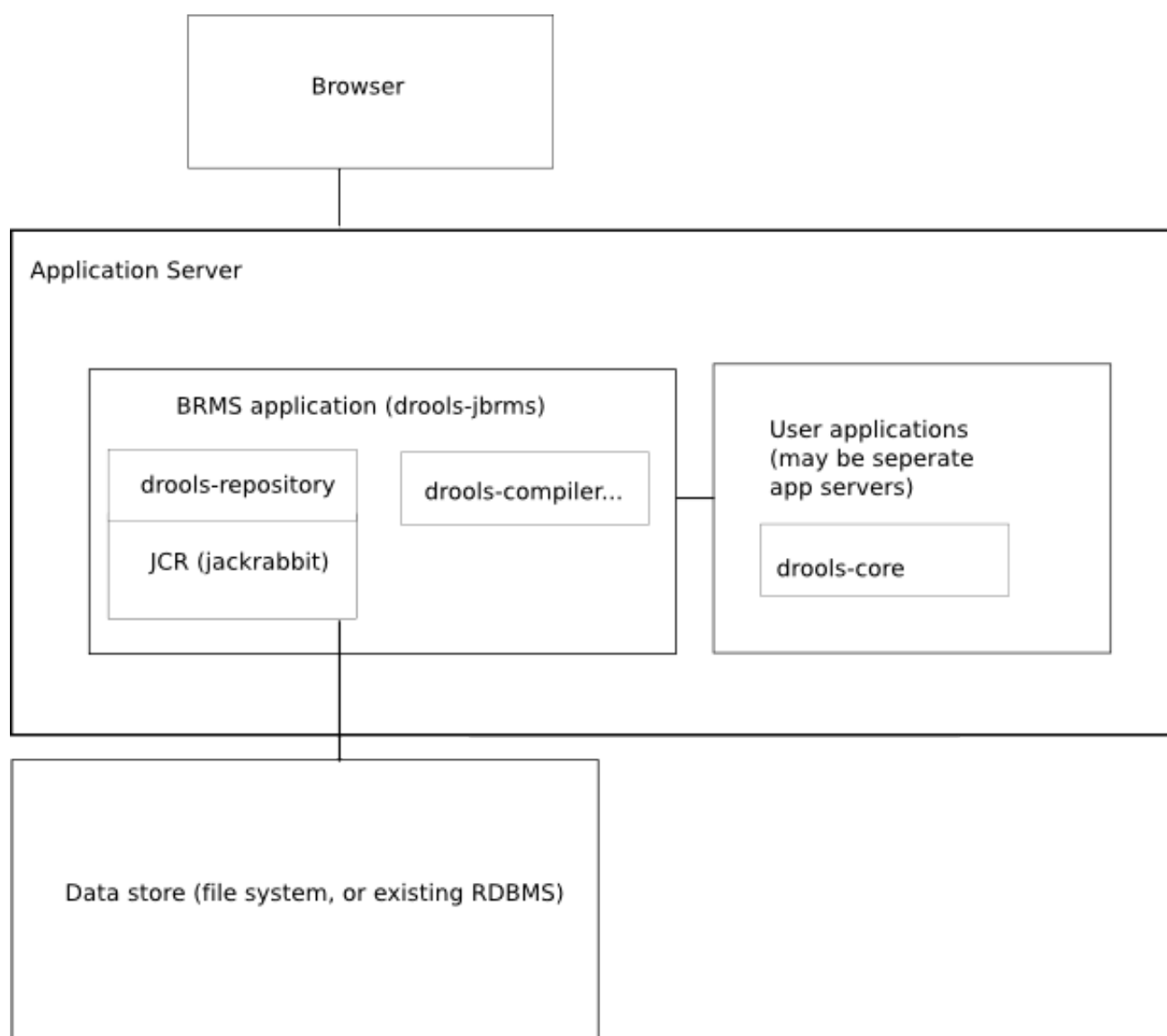


Figure 17.1. Architectural diagram

The above diagram shows the major components of the system and how they integrate and are deployed. The User Guide has more details on the parts that are highly configurable (e.g. database).

Guvnor is deployed as a WAR, which provides user interfaces over the web, and provides binary packages using URLs (or files). It uses the JSR-170 standard for data storage (JCR). JBoss Seam is used as the component framework, and GWT is used as the widget toolkit for constructing the AJAX-based web user interface.

17.1. Building from source

This section will go over the steps necessary to build various components. Mostly this is automated, but the manual process is described for thoroughness.

17.1.1. Modules

There are 2 modules: `guvnor-repository` (back end) and `guvnor-webapp` (front end and rules integration). The `guvnor-webapp` module depends on the `guvnor-repository` module, as well as other components. Guvnor is part of the main build for all of Drools - when building Drools, Guvnor is built alongside it.

17.1.2. Working with Maven 2

Maven 2 is used as the build system. To get started, the *whole* of the source tree for JBoss Rules needs to be checked out. This includes the other modules, and the top level `lib` and `repository` directories (which are needed by the build); as the Guvnor build is part of the main Drools build.

Initially, go into the root of the `jboss-rules` checked out source tree, and run `mvn install` to install all the components for the inter-project dependencies. If the build is broken (all care is taken for this eventuality not to occur), the flag `-DskipTests` can be used to prevent failing unit tests from preventing the build.

When wishing to build Guvnor, go into the `guvnor-webapp` directory, and run `mvn package`. This will run the tests, and then build a deployable WAR. Once the WAR file is in the target directory, the Guvnor is ready to go.

17.1.3. Working with GWT

The GUI widgets for the web front end are developed with GWT (Google Web Toolkit).

17.1.4. Debugging, Editing and running with Eclipse

Each module has a ready to go and up to date eclipse project configuration, so they can merely be imported into the eclipse workspace. These projects are generated by Maven. Use the `mvn eclipse:eclipse` command to refresh them in case they are wrong or outdated. They have been manually modified to have project dependencies which means that the code can be stepped through when debugging.

Some environment variables are required in eclipse (for Window: `>Preferences->Java->Build path-> Classpath variables`): the `M2_REPO`, as normal, to point to where Maven downloads shared dependencies. `GWT_HOME` should point to where you installed GWT. `GWT_DEV` must point to the platform specific "dev" JAR that ships with the version of GWT you have.

How to launch from Eclipse: unit tests can be launched, as normal (in which case only `M2_REPO` setup is needed, GWT does not need to be downloaded separately), or it can be launched in *hosted mode* using the GWT browser, which is great for debugging (from GUI to back end, the code can be stepped through, and changes made on the fly and simply hit refresh). There is a `Guvnor.launch` file in the `guvnor-webapp` directory. To launch Guvnor in debug mode, open the Run dialog (**Run->Run**), and then choose **Guvnor** from the list. Launching this will open a new window, with Guvnor in debug mode, ready to go.

Downloading and debugging Guvnor with GWT is optional, so if there are no GUI issues being worked on then this step can be safely skipped.

17.2. Re-usable components

Guvnor uses a service interface to separate the GUI from the back end functionality. In this case the back end both includes the asset repository (`guvnor-repository` and JCR) as well as the compiler specifics to deal with rules.

The main interface is `RepositoryService`, which is implemented in `ServiceImplmentation`. The GWT ajax front end talks to this interface using the asynchronous callback mechanism that GWT uses. The Seam configuration file is `beans.xml`. Refer to the Seam documentation, and the `beans.xml` file for details.

This service interface may be re-used by alternative components or front ends.

The GWT user interface may be re-used, as it is GWT is only one html page: `Guvnor.html`.

Normally Guvnor is intended to be deployed as its own WAR, however it can be combined with another application (with some care), but it is easier to keep it as a separate WAR. We recommend deploying Guvnor by itself because this will make it easier to upgrade to newer releases as they come out.

The `Guvnor.html` file can be customized. For example to change logos or embed Guvnor in another page. Take a look at the `Guvnor.html` file for details.

17.3. Versioning and Storage

Refer to [Chapter 12, Database configuration](#) for configuration options for database and filesystems.

Versions of assets are stored in the database along with the data.

When *snapshots* are created, copies are made of the entire package into a separate location in the JCR database.

For those familiar with JCR and Apache Jackrabbit, the `*.cnd` files are in the source for the node type definitions as some wish to view these. In a nutshell, a package is a *folder* and each asset is a file: an asset can either be textual or have a binary attachment.

17.4. Contributing

As an open source project, contributions from the wider community are encouraged. In order to contribute consult the wiki and project home pages. A useful way to contribute is via logging issues or feature requests in JIRA. The JIRA link to use is <https://jira.jboss.org/jira/browse/GUVNOR>.