

JBoss Enterprise BRMS Platform 0.1 BRMS Userguide

**A guide to BRMS for administrators,
business analysts and developers.**



Michael Neale

Fernando Meyer

JBoss Enterprise BRMS Platform 0.1 BRMS Userguide

A guide to BRMS for administrators, business analysts and developers.

Edition 0

Author Michael Neale
Author Fernando Meyer
Editor Darrin Mison dmison@redhat.com
Copyright © 2008 Red Hat, Inc.

Copyright © 2008 Red Hat, Inc.. This material may only be distributed subject to the terms and conditions set forth in the Open Publication License, V1.0 or later (the latest version of the OPL is presently available at <http://www.opencontent.org/openpub/>).

Red Hat and the Red Hat "Shadow Man" logo are registered trademarks of Red Hat, Inc. in the United States and other countries.

All other trademarks referenced herein are the property of their respective owners.

1801 Varsity Drive
Raleigh, NC 27606-2072 USA
Phone: +1 919 754 3700
Phone: 888 733 4281
Fax: +1 919 754 3701
PO Box 13588 Research Triangle Park, NC 27709 USA

This book contains everything you need to know to make the most of the JBoss BRMS product.

Preface	v
1. Document Conventions	v
1.1. Typographic Conventions	v
1.2. Pull-quote Conventions	vi
1.3. Notes and Warnings	vii
2. We Need Feedback!	viii
1. Introduction	1
1.1. What is a BRMS?	1
1.2. When to use a BRMS	1
1.3. Who uses a BRMS	2
1.4. Features outline	2
2. BRMS Installation & Administration	3
2.1. Supported and recommended platforms	3
2.2. Installation	3
2.3. BRMS Repository Configuration	5
2.3.1. Changing the location of the Repository	5
2.3.2. BRMS Database Configuration	7
2.3.3. Searching and Indexing	10
2.4. Security - Authentication	11
2.4.1. Authentication Example - UserRolesLoginModule	11
2.4.2. Authentication Example - LDAP	13
2.5. Security - Authorization	14
2.5.1. Admin Permissions	17
2.5.2. Analyst Permissions	17
2.5.3. Package Permissions	17
2.6. Data management	18
2.6.1. Backups	18
2.6.2. Asset list customization	18
2.6.3. Customized selectors for package building	18
2.6.4. Customizing the BRMS Platform User Interface	19
2.6.5. Import and Export	19
3. Architecture	21
3.1. Re-usable components	22
3.2. Versioning and Storage	22
4. Quick start guide	23
4.1. Quick start guide	23
4.1.1. Supported browsers	23
4.1.2. Initial configuration	24
4.1.3. Writing some rules	24
4.1.4. Finding stuff	24
4.1.5. Deployment	25
4.2. BRMS concepts	25
4.2.1. Rules are assets	25
4.2.2. Categorization	25
4.2.3. The asset editor	27
4.2.4. Rule authoring	28
4.2.5. Templates of assets/rules	37
4.2.6. Status management	37
4.2.7. Package management	38

4.2.8. Version management	41
4.2.9. Deployment management	41
4.2.10. Navigating and finding rules	42
4.3. Creating a business user view	43
4.4. The fact model (object model)	44
4.5. The business user perspective	46
4.6. Deployment: Integrating rules with your applications	46
4.6.1. The Rule Agent	46
4.6.2. JMS, SOAP/WSDL integration	49
4.6.3. Manual deployment	49
4.7. WebDAV and HTTP	49
4.8. URLs	50
5. Eclipse Integration	51
5.1. Source Code and Plug-in Details	51
5.2. Functionality Overview	51
5.3. Guvnor Connection Wizard	52
5.4. Guvnor Repository Explorer	55
5.5. Local Copies of Guvnor Files	57
5.6. Actions for Local Guvnor Resources	59
5.7. Importing Guvnor Repository Resources	64
5.8. Guvnor plugin Preferences	68
A. Example Persistence Manager configurations	69
B. Revision History	71

Preface

1. Document Conventions

This manual uses several conventions to highlight certain words and phrases and draw attention to specific pieces of information.

In PDF and paper editions, this manual uses typefaces drawn from the *Liberation Fonts*¹ set. The Liberation Fonts set is also used in HTML editions if the set is installed on your system. If not, alternative but equivalent typefaces are displayed. Note: Red Hat Enterprise Linux 5 and later includes the Liberation Fonts set by default.

1.1. Typographic Conventions

Four typographic conventions are used to call attention to specific words and phrases. These conventions, and the circumstances they apply to, are as follows.

Mono-spaced Bold

Used to highlight system input, including shell commands, file names and paths. Also used to highlight key caps and key-combinations. For example:

To see the contents of the file `my_next_bestselling_novel` in your current working directory, enter the `cat my_next_bestselling_novel` command at the shell prompt and press **Enter** to execute the command.

The above includes a file name, a shell command and a key cap, all presented in Mono-spaced Bold and all distinguishable thanks to context.

Key-combinations can be distinguished from key caps by the hyphen connecting each part of a key-combination. For example:

Press **Enter** to execute the command.

Press **Ctrl-Alt-F1** to switch to the first virtual terminal. Press **Ctrl-Alt-F7** to return to your X-Windows session.

The first sentence highlights the particular key cap to press. The second highlights two sets of three key caps, each set pressed simultaneously.

If source code is discussed, class names, methods, functions, variable names and returned values mentioned within a paragraph will be presented as above, in **Mono-spaced Bold**. For example:

File-related classes include `filesystem` for file systems, `file` for files, and `dir` for directories. Each class has its own associated set of permissions.

Proportional Bold

This denotes words or phrases encountered on a system, including application names; dialogue box text; labelled buttons; check-box and radio button labels; menu titles and sub-menu titles. For example:

¹ <https://fedorahosted.org/liberation-fonts/>

Choose **System > Preferences > Mouse** from the main menu bar to launch **Mouse Preferences**. In the **Buttons** tab, click the **Left-handed mouse** check box and click **Close** to switch the primary mouse button from the left to the right (making the mouse suitable for use in the left hand).

To insert a special character into a **gedit** file, choose **Applications > Accessories > Character Map** from the main menu bar. Next, choose **Search > Find...** from the **Character Map** menu bar, type the name of the character in the **Search** field and click **Next**. The character you sought will be highlighted in the **Character Table**. Double-click this highlighted character to place it in the **Text to copy** field and then click the **Copy** button. Now switch back to your document and choose **Edit > Paste** from the **gedit** menu bar.

The above text includes application names; system-wide menu names and items; application-specific menu names; and buttons and text found within a GUI interface, all presented in Proportional Bold and all distinguishable by context.

Note the **>** shorthand used to indicate traversal through a menu and its sub-menus. This is to avoid the difficult-to-follow 'Select **Mouse** from the **Preferences** sub-menu in the **System** menu of the main menu bar' approach.

Mono-spaced Bold Italic or ***Proportional Bold Italic***

Whether Mono-spaced Bold or Proportional Bold, the addition of Italics indicates replaceable or variable text. Italics denotes text you do not input literally or displayed text that changes depending on circumstance. For example:

To connect to a remote machine using ssh, type **ssh *username@domain.name*** at a shell prompt. If the remote machine is **example.com** and your username on that machine is john, type **ssh *john@example.com***.

The **mount -o remount *file-system*** command remounts the named file system. For example, to remount the **/home** file system, the command is **mount -o remount */home***.

To see the version of a currently installed package, use the **rpm -q *package*** command. It will return a result as follows: ***package-version-release***.

Note the words in bold italics above — *username*, *domain.name*, *file-system*, *package*, *version* and *release*. Each word is a placeholder, either for text you enter when issuing a command or for text displayed by the system.

Aside from standard usage for presenting the title of a work, italics denotes the first use of a new and important term. For example:

When the Apache HTTP Server accepts requests, it dispatches child processes or threads to handle them. This group of child processes or threads is known as a *server-pool*. Under Apache HTTP Server 2.0, the responsibility for creating and maintaining these server-pools has been abstracted to a group of modules called *Multi-Processing Modules (MPMs)*. Unlike other modules, only one module from the MPM group can be loaded by the Apache HTTP Server.

1.2. Pull-quote Conventions

Two, commonly multi-line, data types are set off visually from the surrounding text.

Output sent to a terminal is set in Mono-spaced Roman and presented thus:

```
books      Desktop  documentation  drafts  mss    photos  stuff  svn
books_tests Desktop1  downloads      images  notes  scripts svgs
```

Source-code listings are also set in Mono-spaced Roman but are presented and highlighted as follows:

```
package org.jboss.book.jca.ex1;

import javax.naming.InitialContext;

public class ExClient
{
    public static void main(String args[])
        throws Exception
    {
        InitialContext iniCtx = new InitialContext();
        Object          ref    = iniCtx.lookup("EchoBean");
        EchoHome        home   = (EchoHome) ref;
        Echo             echo   = home.create();

        System.out.println("Created Echo");

        System.out.println("Echo.echo('Hello') = " + echo.echo("Hello"));
    }
}
```

1.3. Notes and Warnings

Finally, we use three visual styles to draw attention to information that might otherwise be overlooked.



Note

A Note is a tip or shortcut or alternative approach to the task at hand. Ignoring a note should have no negative consequences, but you might miss out on a trick that makes your life easier.



Important

Important boxes detail things that are easily missed: configuration changes that only apply to the current session, or services that need restarting before an update will apply. Ignoring Important boxes won't cause data loss but may cause irritation and frustration.



Warning

A Warning should not be ignored. Ignoring warnings will most likely cause data loss.

2. We Need Feedback!

If you find a typographical error in this manual, or if you have thought of a way to make this manual better, we would love to hear from you! Please submit a report in Bugzilla: <http://bugzilla.redhat.com/bugzilla/> against the product **Documentation**.

When submitting a bug report, be sure to mention the manual's identifier: *BRMS_Userguide*

If you have a suggestion for improving the documentation, try to be as specific as possible when describing it. If you have found an error, please include the section number and some of the surrounding text so we can find it easily.

Introduction

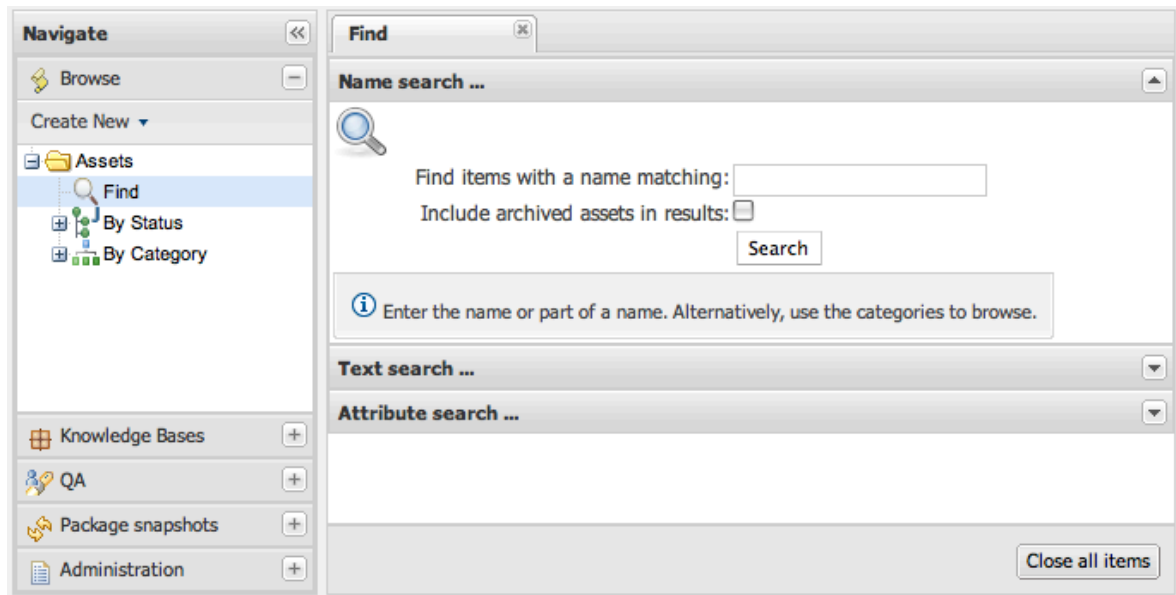


Figure 1.1. The BRMS main screen

1.1. What is a BRMS?

BRMS stands for Business Rules Management System.

The JBoss Enterprise BRMS Platform is a JBoss Rules based server-side solution for the management, storage, editing and deployment of rules and other JBoss Rules assets. A web-based user interface is provided as well as integration for JBoss Developer Studio and other Eclipse based IDEs.

A BRMS 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.

1.2. When to use a BRMS

You should consider a BRMS if any of the following apply:

1. You need to manage versions/deployment of rules
2. You need to let multiple users of different skill levels access and edit rules
3. You don't have any existing infrastructure to manage rules
4. You have lots of "business" rules (as opposed to technical rules as part of an application).

The JBoss Enterprise BRMS Platform can be "branded" for integration into another application, or it can be used as a central rule repository.

1.3. Who uses a BRMS

The main roles of people who would use a BRMS are: Business Analyst, Rule expert, Developer and Administrators (rule administrators etc). The BRMS allows different roles to be assigned to different users to control what assets and functionality is exposed to different users.

1.4. 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

BRMS Installation & Administration

This chapter covers installation and administration of the BRMS environment.

The BRMS 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.

2.1. Supported and recommended platforms

The BRMS is capable of running in any application server that supports Java SE5. JEE 5 is not required.

It is actively tested on JBoss platforms, and these are recommended if you are able to use them, or don't have any existing infrastructure. However, it is possible to use any container/app server, in some cases with minor configuration tweaks (consult the wiki for specific tips).

The following are a list of recommended platforms (all freely available), in all cases, you can substitute newer versions of the same and it should work (as time may have passed since this was written):

- JBoss Application Server 4.2.x

This is recommended as a general application server solution, if you need to run other applications alongside the BRMS.

- JBoss Web 1.0.1

This is an ideal solution if you need a "lighter" server to run just the BRMS (perhaps stand alone).

You can of course download these from www.jboss.com for every operating system.

Deployment into JBoss platforms: If you are installing a new JBoss platform, the WAR can be copied to [app server directory]/server/default/deploy. You then start up the server by running run.sh or run.bat in the [app server directory/bin] directory.

supported databases

2.2. Installation

Installation of the JBoss Enterprise BRMS Platform is very simple. The BRMS application is packaged as a web archive (WAR file). This can be deployed to your application servers with no configuration at all if you wish to use the defaults. However in a production environment some configuration will be required for security.

Procedure 2.1. Installing JBoss Enterprise BRMS Platform

1. Download the package

The JBoss Enterprise BRMS Platform package, **jboss-brms-50.zip** can be downloaded from the Red Hat JBoss Customer Support Portal at <https://support.redhat.com/jbossnetwork/>

2. Extract Files

The JBoss Enterprise BRMS Platform package contains two files, **jboss-brms.war** and **jboss-brms-engine.zip**.

jboss-brms.war is a Java web archive containing the BRMS application.

jboss-brms-engine.zip contains the jar files for the JBoss Rules 5 rules engine. These files are a subset of **jboss-brms.war**.

Extract the file **jboss-brms.war** from the package zip file to a temporary directory. You can simply deploy this web archive to your application server but you will be unable to configure it.

3. **"explode" the web archive**

Create a directory called **jboss-brms.war** and use a file archive tool to unpack the web archive into this directory.

4. **Configure**

Now that the web archive is in "exploded" form you can configure it. In a production environment you would want to configure your authentication method and database details at the bare minimum. Refer to [Section 2.3, "BRMS Repository Configuration"](#) and for configuration details.

5. **Deploy the "exploded" web archive**

Ensure that your application server is fully shutdown and then copy the "exploded" web archive directory to the deploy directory of your application server. Once all the files have been copied successfully you can restart your application server.

6. **Confirm installation**

Open a web browser and navigate to <http://localhost:8080/jboss-brms/>. You should be presented with the BRMS Platform login page.

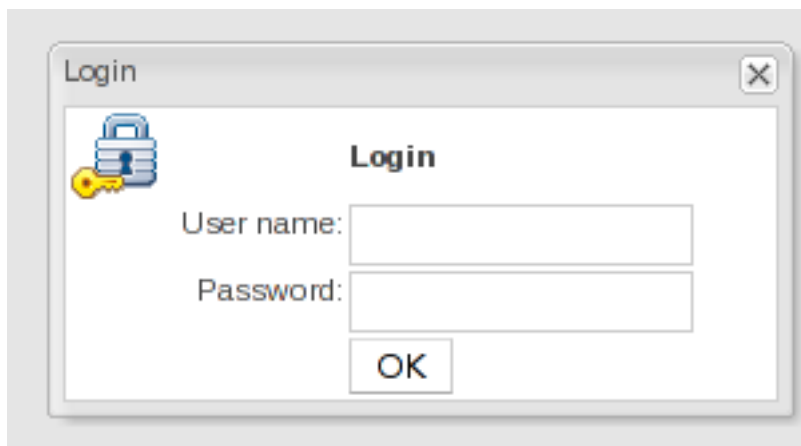


Figure 2.1. BRMS Login

If you cannot get the login page to display then you should check that you have used the correct host name and port number for your application server in the URL. If you are still unable to reach the page you should ensure that your application server has fully started, and then check your server logs for an indication of what is wrong.

By default the BRMS Platform is configured to use the JBoss authentication policy of "**jmx-console**". Assuming that this policy exists you will be able to login to the BRMS Platform with a username and password that is valid for your JMX Console.



Important

It is recommended that you define an authentication policy specifically for use by your BRMS Platform instance. *Section 2.4, "Security - Authentication"* contains details on configuring authentication.

If a new registry was created you will also be prompted to install a set of sample rules and other assets. These samples can be useful for training, testing, and demonstration purposes.

Once you successfully login you will be presented with the **Find** page and your JBoss Enterprise BRMS Platform installation is complete.

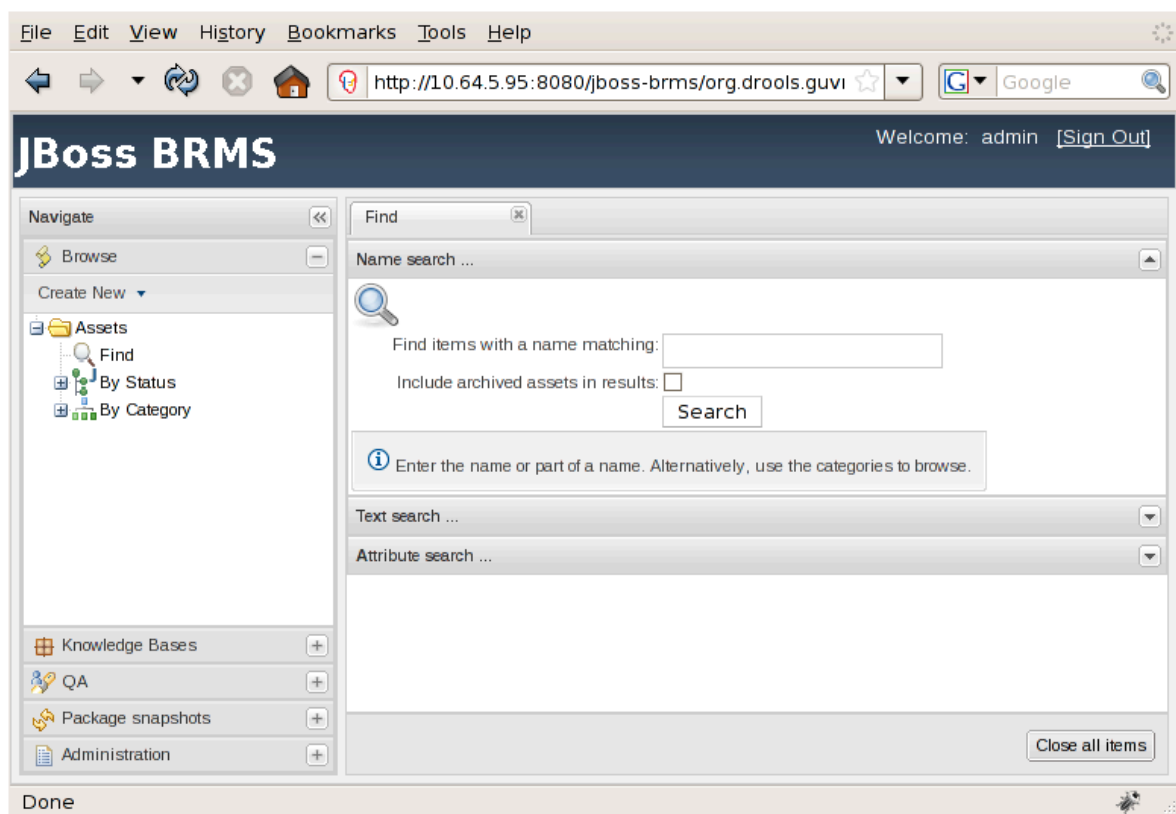


Figure 2.2. BRMS first screen after login

2.3. BRMS Repository Configuration

The BRMS Platform uses the Content Repository API for Java (JCR) standard for storing and tracking assets such as rules. The included implementation is Apache Jackrabbit, <http://jackrabbit.apache.org>, and includes an embedded database. It can also be configured to use a different database system of your choice.

2.3.1. Changing the location of the Repository

When BRMS is started for the first time it will create a repository in the **bin** directory of your application server.

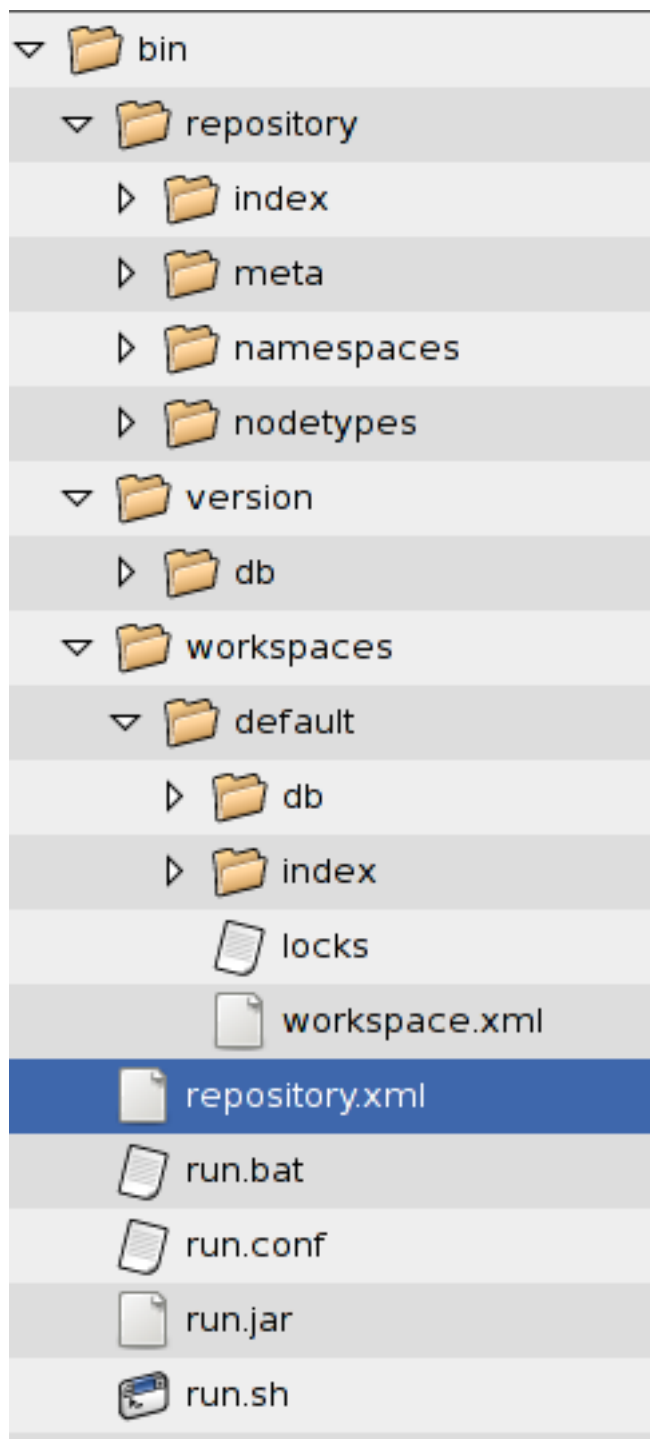


Figure 2.3. Default repository file layout

You can specify a different location for it in the JBoss Seam configuration file for the BRMS application. The location of the data store should be a secure location that is regularly backed up.

Procedure 2.2. Changing the repository location

1. **Shutdown BRMS**

You need to shutdown the BRMS application before performing this configuration. You can do this by either halting your application server or by un-deploying the BRMS application.

2. Locate `components.xml`

The file `components.xml` is located in the **WEB-INF** directory of the BRMS web archive. This is a standard JBoss Seam configuration file which allows various parts of the application to be customized.

3. Locate the `homeDirectory` property

Open the file `components.xml` in a text editor and find the `homeDirectory` property. It is contained in the `repositoryConfiguration` component and is commented out by default.

```
<component name="repositoryConfiguration">
<!--
*** This is for configuring the "home" directory for the repository
storage. the directory must exist. ***
<property name="homeDirectory">
    /home/michael/RulesRepository_001
</property>
-->
</component>
```

4. Update `homeDirectory`

Uncomment the `homeDirectory` element and change its value to the directory where you want your data store to be located. This directory must already exist.

```
<component name="repositoryConfiguration">
    <property name="homeDirectory">
        /opt/jboss-soa-platform/BRMSRulesRepository
    </property>
</component>
```

5. Move existing data store if required

A new data store will be created by BRMS at the new location if there isn't one there already.

If you wish to keep your existing data store then you can copy your existing files to the new location.

6. Restart BRMS

Depending on how you shutdown BRMS in step 1, restart the BRMS application. If you didn't move an existing data store to the new location then a new data store is created.

2.3.2. BRMS Database Configuration

The JBoss Enterprise BRMS Platform maintains two databases. One is used for data storage, referred to as a workspace. The other is used to maintain a history of all changes to the workspace, referred to as versions.

The default configuration uses embedded Derby databases for storage. While this is suitable for evaluation and testing, Derby is not supported by the JBoss Enterprise BRMS Platform in production systems. For a production system you will need to configure the repository to use one of the supported databases.

The means by which the BRMS Platform manages access to these databases is handled by a Persistence Manager. There is a generic Persistence Manager for use with any JDBC compliant database as well as several specific to particular database implementations, such as Oracle or MySQL. [Appendix A, Example Persistence Manager configurations](#) contains example Persistence Manager configurations for each of the supported database servers.

For additional details about Persistence Managers you should refer to <http://wiki.apache.org/jackrabbit/PersistenceManagerFAQ>



Warning

If you already have assets stored using the existing configuration that you wish to keep then you should export those before performing these steps. You can then import them into your new database afterwards.

Before performing this configuration you must already have:

1. an empty database created for use by BRMS,
2. a database server user account for use by BRMS with permissions to create tables in the database,



Note

BRMS will create its tables in the specified database if they do not already exist. Once this has occurred you can safely remove the ability to create tables from the database user.

3. and the appropriate JDBC driver JAR for the database server

It is possible to use either a single database for both Workspace and Versions or you can use different databases for each. It is recommended to use a single database for ease of management.

2.3.2.1. Configuring your Workspace Database

The existing Workspace database configuration is found in the file **Workspaces/Default/workspace.xml**. The file **repository.xml** contains the default settings which are used for new workspaces. Instead of updating the existing Default workspace it is simpler to delete it and update the template configuration in **repository.xml**.

Procedure 2.3. Changing the repository database

1. **Shutdown**
Ensure that the BRMS Platform is not running, either by undeploying it or halting your application server.
2. **Install JDBC driver JAR**
Copy the JAR file that contains the JDBC driver for the database server that you are using to the **lib** directory of your server configuration. Eg. `${INSTALL}/server/production/lib/`

3. open repository.xml in a text-editor

Open the file **repository.xml**. If you didn't specify a different homeDirectory as described in [Section 2.3.1, "Changing the location of the Repository"](#) then **repository.xml** will be in your application server's **bin** directory. Otherwise it will be in the directory that you specified as the homeDirectory property.

4. update the PersistenceManager configuration

Within the <Workspace> element you will find a <PersistenceManager> element. The default configuration looks like:

```
<PersistenceManager class=
  "org.apache.jackrabbit.core.persistence.bundle.DerbyPersistenceManager">
  <param name="url" value="jdbc:derby:${wsp.home}/db;create=true"/>
  <param name="schemaObjectPrefix" value="${wsp.name}_"/>
</PersistenceManager>
```

Replace this with the appropriate configuration for your database. [Appendix A, Example Persistence Manager configurations](#) contains several example configurations for different databases.

5. Set the schemaObjectPrefix parameter

This value is prefixed on the name of each database table managed by this Persistence Manager. This must be unique in each database. The default configuration uses the variable **`\${wsp.name}`** which contains the name of the current workspace.

6. Delete the old workspace

Delete the **workspaces/Default/** directory.

7. Restart

Restart the BRMS application.

2.3.2.2. Configuring your Versions Database

The BRMS Platform also maintains a history of all changes to the database, also referred to as versions. This is kept in a separate storage area to prevent performance degradation as the number of versions increases.

The version storage has its own persistence management configuration in **repository.xml**. Unlike the workspace configuration the version storage configuration here is not a template, it is the actual configuration that is in use.

Procedure 2.4. Changing the Versions database

1. Shutdown

Ensure that the BRMS application is not running, either by undeploying it or halting your application server.

2. Install JDBC driver JAR

Copy the JAR file that contains the JDBC driver for the database server that you are using to the **lib** directory of your server configuration. Eg. **`\${INSTALL}/server/production/lib/**

3. open repository.xml in a text-editor

Open the file **repository.xml**. If you didn't specify a different homeDirectory as described in [Section 2.3.1, "Changing the location of the Repository"](#) then **repository.xml** will be in your application server's **bin** directory. Otherwise it will be in the directory that you specified as the homeDirectory property.

4. update the PersistenceManager configuration

Within the <Versioning> element you will find a <PersistenceManager> element. The default configuration looks like:

```
<PersistenceManager class=
  "org.apache.jackrabbit.core.persistence.bundle.DerbyPersistenceManager">
  <param name="url" value="jdbc:derby:${rep.home}/version/
db;create=true"/>
  <param name="schemaObjectPrefix" value="version_"/>
</PersistenceManager>
```

Replace this with the appropriate configuration for your database. [Appendix A, Example Persistence Manager configurations](#) contains several example configurations for different databases.

5. Set the schemaObjectPrefix property

This value is prefixed on the name of each database table managed by this Persistence Manager. This must be unique in each database. If you are using the same database for Versions as you are for the datastore then you need to ensure that you do not use the same schemaObjectPrefix for both of them. The default configuration uses **version_**.

6. Restart

Restart the BRMS application.

2.3.3. Searching and Indexing

The BRMS Platform uses Apache Lucene to provide indexing and search functionality for both the workspace and versions databases. The default configuration stores this index on the local filesystem for performance reasons. It is recommended to leave this configuration with the default settings unless you have different requirements.

This configuration is found in **repository.xml** within the <SearchIndex> element.

```
<SearchIndex class="org.apache.jackrabbit.core.query.lucene.SearchIndex">
  <param name="path" value="${wsp.home}/index"/>
  <param name="textFilterClasses"
    value="org.apache.jackrabbit.extractor.MsWordTextExtractor,
org.apache.jackrabbit.extractor.MsExcelTextExtractor,
org.apache.jackrabbit.extractor.MsPowerPointTextExtractor,
org.apache.jackrabbit.extractor.PdfTextExtractor,
org.apache.jackrabbit.extractor.OpenOfficeTextExtractor,
org.apache.jackrabbit.extractor.RTFTextExtractor,
org.apache.jackrabbit.extractor.HTMLTextExtractor,
org.apache.jackrabbit.extractor.XMLTextExtractor" />
  <param name="extractorPoolSize" value="2"/>
  <param name="supportHighlighting" value="true"/>
```

```
</SearchIndex>
```

You can find additional information about Apache Lucene at its website: <http://lucene.apache.org/>.

2.4. Security - Authentication

The JBoss Enterprise BRMS Platform does not store authentication information (usernames and passwords) for users. Instead it makes use of the Java Authentication and Authorization Service (JAAS) supplied by the JBoss Application Server to access a separate authentication system. This can be a system that you already have in place on your network. This can be an LDAP or Active Directory Server, a JDBC database, or any other resource that you can access via a JAAS JBoss login module.



Important

By granting users access to your BRMS Platform installation you are potentially giving them the ability to affect the business logic of how other applications run. Role-based authorization can be used to define what each user is allowed to do. *Section 2.5, "Security - Authorization"* contains more details on this topic.

The method by which authentication is performed is configured in the `jboss-brms.war/WEB-INF/components.xml` file. The default configuration has many commented out options but the actual configuration looks like this:

```
<security:identity authenticate-
method="#{authenticator.authenticate}" jaas-config-name="jmx-console"/>
<security:role-based-permission-resolver enable-role-based-
authorization="false"/>
```

This default configuration uses usernames, passwords and roles defined in the JBoss authentication policy "jmx-console". It is recommended that you update this policy to one that you configure specifically for your BRMS Platform installation.

Configuring Authentication for the BRMS Platform involves configuring the appropriate JBoss login module on your application Server and then configuring the BRMS Platform to use that module.

Many JBoss login modules provide a means of specifying one or more roles for each user. The BRMS Platform only makes use of this information if *role-based-authorization* is disabled. The BRMS Platform provides its own mechanism for managing user roles and it is recommended that role-based authorization is enabled and the BRMS Platform is used to manage user roles. It is important that before you enable role-based authorization that you assigned at least one user to the Admin role in the BRMS permissions interface. Once role-based authorization is enabled only users with the Admin role can perform many administrative tasks including managing user roles. This is detailed further in [Section 2.5, "Security - Authorization"](#).

2.4.1. Authentication Example - UserRolesLoginModule

This is a very simple example here to illustrate the process using the `org.jboss.security.auth.spi.UsersRolesLoginModule` login module to access a set of user accounts stored in the files `props/brms-users.properties` and `props/brms-roles.properties`.

Procedure 2.5. Authentication Example - **UserRolesLoginModule**

1. **Ensure your authentication system is correctly setup**

This login module uses two simple files to store the usernames, passwords and roles assigned to each user. Create the files, **brms-users.properties** and **brms-roles.properties** in the directory `${INSTALL}/jboss-as/server/${CONFIG}/conf/props/`. Then specify at least one user in `brms-users.properties` using the format **username=password**. You can leave the file `brms-roles.properties` blank.

2. **Shutdown**

Ensure that the BRMS application is not running, either by undeploying it or halting your application server.

3. **Configure the JBoss Login Module**

JBoss Login Modules are configured in the file `${INSTALL}/jboss-as/server/${CONFIG}/conf/login-config.xml`. Open this file in an editor. It is an XML file containing a `<policy>` element with several `<application-policy>` child elements. Each `<application-policy>` element defines a different authentication scheme. Add the following `<application-policy>` XML snippet as a new child of the `<policy>` element.

```
<!--BRMS Platform Security Domain-->
<application-policy name="brms">
  <authentication>
    <login-module
      code="org.jboss.security.auth.spi.UsersRolesLoginModule"
      flag="required">
      <module-option name="usersProperties">
        props/brms-users.properties
      </module-option>
      <module-option name="rolesProperties">
        props/brms-roles.properties
      </module-option>
    </login-module>
  </authentication>
</application-policy>
```

4. **Configure the BRMS Platform to use the Login Module**

Open the file `${INSTALL}/jboss-as/server/${CONFIG}/deploy/jboss-brms.war/WEB-INF/components.xml`. It contains one `<components>` element with several child elements, including `<security:identity>` and `<security:role-based-permission-resolver>`.

Comment out the existing `<security:identity>` elements so you don't have any conflicts. Add the following `<security:identity>` element.

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

If you changed the application-policy name in the previous step you also need to update the `jaas-config-name` here as well.

5. **Restart**

Restart the BRMS application.

2.4.2. Authentication Example - LDAP

LDAP is perhaps the most popular choice for larger enterprises. The basic configuration steps are the same as the previous example although the details of the configuration will differ.

Procedure 2.6. Authentication Example 2 - LDAP

1. Ensure your LDAP server is correctly setup

You need to ensure that no firewall or network configuration is preventing communication between your application server and your LDAP server.

2. Shutdown

Ensure that the BRMS application is not running, either by un-deploying it or halting your application server.

3. Configure JBoss Login Module

JBoss Login Modules are configured in the file `${INSTALL}/jboss-as/server/${CONFIG}/conf/login-config.xml`. Open this file in an editor. It is an XML file containing a `<policy>` element with several `<application-policy>` child elements. Each `<application-policy>` element defines a different authentication scheme. Add the following `<application-policy>` XML snippet as a new child of the `<policy>` element.

```
<application-policy name="brms">
  <authentication>
    <login-module code="org.jboss.security.auth.spi.LdapExtLoginModule"
      flag="required" >
      <module-option name="java.naming.provider.url">
        ldap://ldap.company.com:389
      </module-option>
      <module-option name="bindDN">DEPARTMENT\someadmin</module-option>
      <module-option name="bindCredential">password</module-option>
      <module-option name="baseCtxDN">cn=Users,dc=company,dc=com
      </module-option>
      <module-option name="baseFilter">(SAMAccountName={0})</module-
option>
      <module-option name="rolesCtxDN">cn=Users,dc=company,dc=com
      </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>
```

You need to update the values in this configuration with those from your own LDAP server.

4. **Configure the BRMS Platform to use the Login Module**

Open the file `${INSTALL}/jboss-as/server/${CONFIG}/deploy/jboss-brms.war/WEB-INF/components.xml`. It contains one `<components>` element with several child elements, including `<security:identity>` and `<security:role-based-permission-resolver>`.

Comment out the existing `<security:identity>` elements so you don't have any conflicts. Add the following `<security:identity>` element.

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

If you changed the application-policy name in the previous step you also need to update the `jaas-config-name` here as well.

5. **Restart**

Restart the BRMS application.

You can get additional information about different LDAP configuration scenarios at <http://wiki.jboss.org/wiki/Wiki.jsp?page=LdapLoginModule> and <http://wiki.jboss.org/wiki/Wiki.jsp?page=LdapExtLoginModule>.

2.5. Security - Authorization

The JBoss Enterprise BRMS Platform allows you to assign different levels of privilege to different users as well as control what rule resources those users have access to. This is done by assigning *roles* to users.

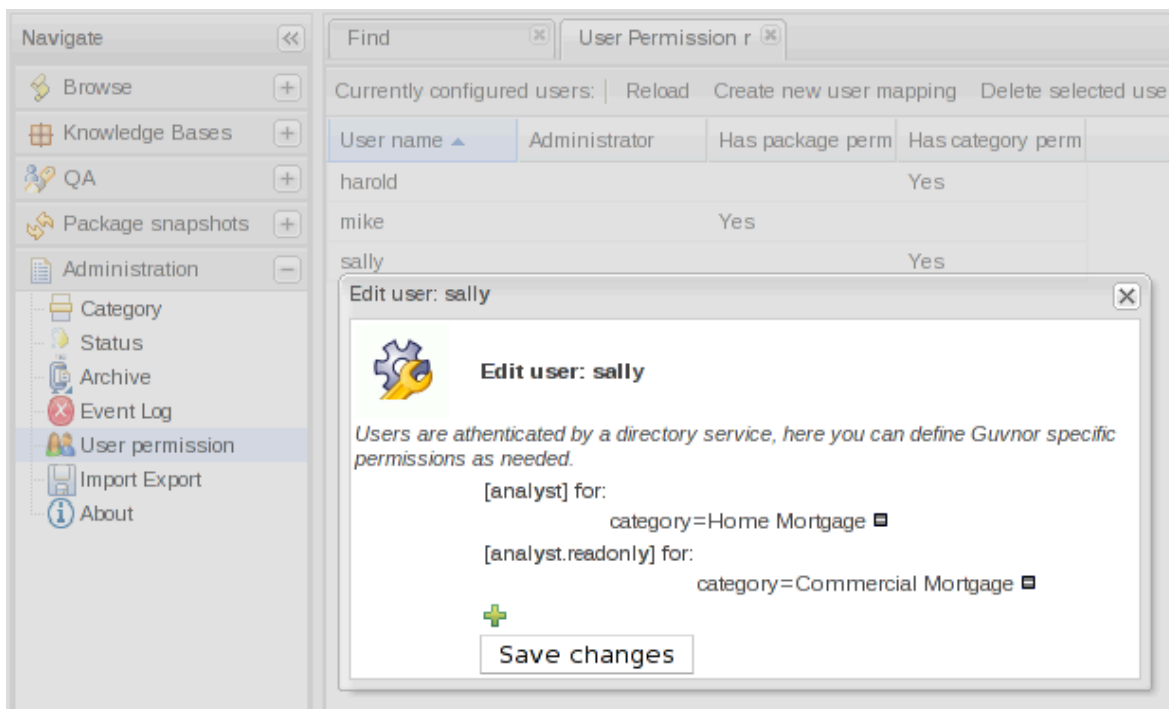


Figure 2.4. Administering User Permissions

By default authorization is not enabled and any username can be used to login with full privileges. [Section 2.4, "Security - Authentication"](#) describes the steps you need to take to enable it. Once this is

done you can enable role-based authorization by following the steps described in [Enabling Role-based Authorization](#).



Important

Before enabling role-based authorization you should login and assign at least one trusted user the role of **admin**. With this role that user will have unlimited access to the BRMS Platform and be able to assign roles to other users.

Procedure 2.7. Enabling Role-based Authorization

1. **Shutdown**

Ensure that the BRMS application is not running, either by un-deploying it or halting your application server.

2. **open components.xml**

Open the file `${INSTALL}/jboss-as/server/${CONFIG}/deploy/jboss-brms.war/WEB-INF/components.xml` in a text-editor.

3. **locate the <security:role-based-permission-resolver> element**

The default `components.xml` file has this XML element as the last child element of `<components>`.

```
<security:role-based-permission-resolver
    enable-role-based-authorization="false"/>
```

4. **Update the attribute value to "true"**

Update the value of the `enable-role-based-authorization` to **"true"** and save the changes to this file.

```
<security:role-based-permission-resolver
    enable-role-based-authorization="true"/>
```

5. **Restart**

Restart the BRMS application.

A user with the **admin** role is able to modify the roles and permissions of other users. This is done by using the **User Permission** panel. This panel can be found by going to the **Navigate** panel, selecting the **Administration** group and then selecting **User Permission**.

User Permission ✕			
Currently configured users: Reload Create new user mapping Delete selected user			
User name ▲	Administrator	Has package permissions	Has category permissions
harold			Yes
mike		Yes	
sally		Yes	Yes
admin	Yes		

Figure 2.5. User Listing

The BRMS Platform does not manage the identity of users. You will have to manually add each user whose role you wish to define.

You can remove user role mappings by selecting the user from the list and clicking the **Delete selected user** button.

Procedure 2.8. Adding a new User to BRMS

1. **Ensure the user exists**

The username that you will be specifying for the role has to match a username that is defined in the service that you have configured for authentication or it will have no effect.

2. **Add the user mapping**

Clicking on the **Create new user mapping** button. Type the username into the dialog that appears and click **OK**. The user will be added and the **Edit User** dialog will appear.

3. **Add permissions**

You can now add the required permissions by clicking on the green "plus" icon. The permissions are covered in detail below. Permissions can also be removed by clicking on the "-" button for the specific permission.

You can save the assigned permissions at anytime by clicking the **Save Changes** button. Once you have finished you can close the dialog by clicking on the **"X"** button on the upper right corner of the dialog.

The permissions assigned to a use can be updated by double-clicking on the user in the list of users on the **User Permissions** panel. You can not currently edit a specific permission assigned to a user. If you wish to change a permission you need to remove it and add a new permission with the changes.

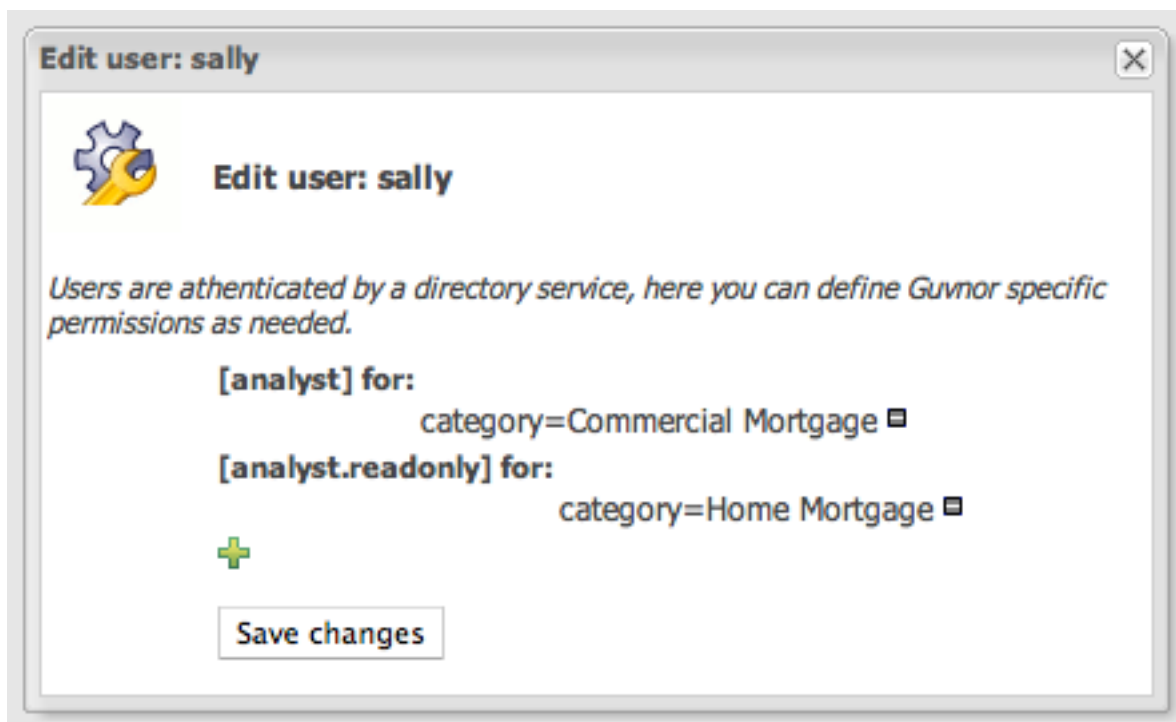


Figure 2.6. Editing

There are basically three types of permissions that can be assigned to a user - admin, analyst and package.

2.5.1. Admin Permissions

Users of the Admin role have complete access to areas of the BRMS Platform. This includes being able to modify the permissions of other users.

2.5.2. Analyst Permissions

Analyst permissions are intended for those users who will be responsible for maintaining the rule resources hosted by the BRMS Platform. This includes developers as well as business analysts.

When you assign Analyst permissions to a user you will be prompted for a category. Categories are a way of grouping rules independent of the package that each rule belongs to.

There is also a read-only analyst permission for situations such as when an analyst is responsible for maintaining rules in one category but needs to be able to examine rules in another category.



Important

A user that only has category permissions will not be shown any package views or details, and will only see the simple categories view.

2.5.3. Package Permissions

Package permissions are limited in scope to one package which you will be prompted for when you assign the permission. There are three different package permissions that you can assign to a user.

Package Administrator

The Package Administrator permission grants full control over the specified package, including the right to deploy it. The Package Administrator permission does not grant any administrative rights to any other part of the BRMS Platform.

Package Developer

The Package Developer permission allows users to create and edit items within the specified package. This includes being able to create and run tests but not the right to deploy the package.

Package Read-only

The Package Read-only permission is similar to the Analyst Read-only permission but grants access to a package instead of a category.

2.6. Data management

2.6.1. Backups

Like all mission critical applications, you need to create backups of your BRMS Platform database on a regular basis. You should consult your documentation for your database server for details on how to configure your backups.

When restoring a backup it is recommended to clear the database indexes so they are created fresh from the data and are thus guaranteed to be optimal.

If you are using the default Derby database creating a backup is a simple manner of creating a copy of your repository directory. You will need to ensure that the BRMS Platform is not being used when then backup is performed. Derby is not a supported database for the BRMS Platform and should not be used in a production environment.

2.6.2. Asset list customization

In a few places in the BRMS Platform there is an asset list. This list can be customized by looking for the **AssetListTable.properties** file. You can then set the header names and the getter methods that are used to populate the columns, e.g. you could add in `getCreator`, or `getExternalSource` as extra fields if you were using them.

2.6.3. Customized selectors for package building

When building packages (from the "Packages" feature) you have the option to specify the name of a "selector". This selector will filter the list of rules that are built into the package. What you enter in the selector text box, is the name of a selector as configured on the server.

To configure a selector you must have deployed the BRMS WAR as an "exploded archive". This is necessary in order to perform any customization or configuration tasks. You can find additional details in [Section 2.2, "Installation"](#). Locate the **selectors.properties** file. In this file, you will find details on how you can configure a custom selector. You can also add your own **selectors.properties** file in the system classpath. The options are to use a drl file, or the name of a class that implements the `AssetSelector` interface. DRL files can also be used and there is an example of one in the **selectors.properties** file. Each selector you configure must have a unique name in this properties file which is the name that you use when building packages.

2.6.4. Customizing the BRMS Platform User Interface

The BRMS user interface is produced dynamically using the GWT framework. Its appearance can be customized - for branding or integration - by editing the images and CSS style sheets used.

The `.css` files can be found in `jboss-brms.war/org.drools.guvnor.Guvnor/` and the images are located in the `images` sub-directory. Of course this assumes that you have deployed the BRMS WAR as an "exploded archive". This is necessary in order to perform any customization or configuration tasks. You can find additional details in [Section 2.2, "Installation"](#).

You can simply edit or replace the images and CSS files and leave the filenames unchanged. If you encounter problems the files can be easily be replaced with the originals from the BRMS WAR file. It is recommended that you add any modified files into a version control system of some variety so the changes can be tracked.

The most common branding change is to replace the images `jbosrules_hdrbkg_blue.gif` and `drools.gif`. Respectively these images are the logo at the top of the interface and the site "favorites icon".¹

`header.css` controls the styling elements of the site header and `Guvnor.css` controls the styling of the remaining elements. There are several other CSS files which are used by the GWT components. It is not recommended to change these.

The URLs used by the BRMS Platform can also be customized by editing the deployment descriptor - `jboss-brms.war/WEB-INF/web.xml` - the same as you would with any other Java web application.

2.6.5. Import and Export

A JCR standard export/import feature is accessible from the Administration interface.

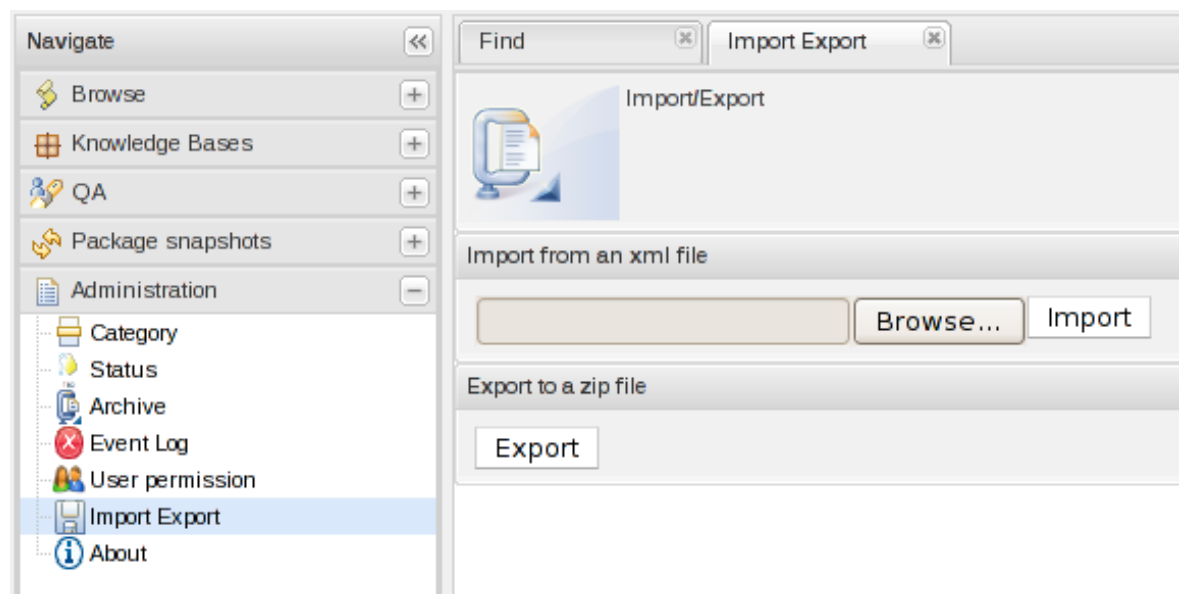


Figure 2.7. Import and Export

¹ The "favorites icon" (favicon) is displayed in different places depending on browser and operating system. Commonly it is seen in the browser address bar, the bookmarks or favorites menu, the window title bar and as the icon used for a desktop URL shortcut.

This will export the entire repository to an XML format as defined by the JCR standard. When importing all existing content in the database is removed. Export and import should only be performed while the BRMS Platform database is not in use.

This should not be used as a substitute for the backup system provided by your database vendor, but can be used migrating from one database to another. However the performance of this feature is extremely dependent on both the size of the database and the available memory on the server.



Warning

The version history of your repository is not included in the export.

Architecture

This section covers the technical aspects of the Business Rules Management System (BRMS), it is not necessary to use this if you are integrating or an end user of the BRMS application.

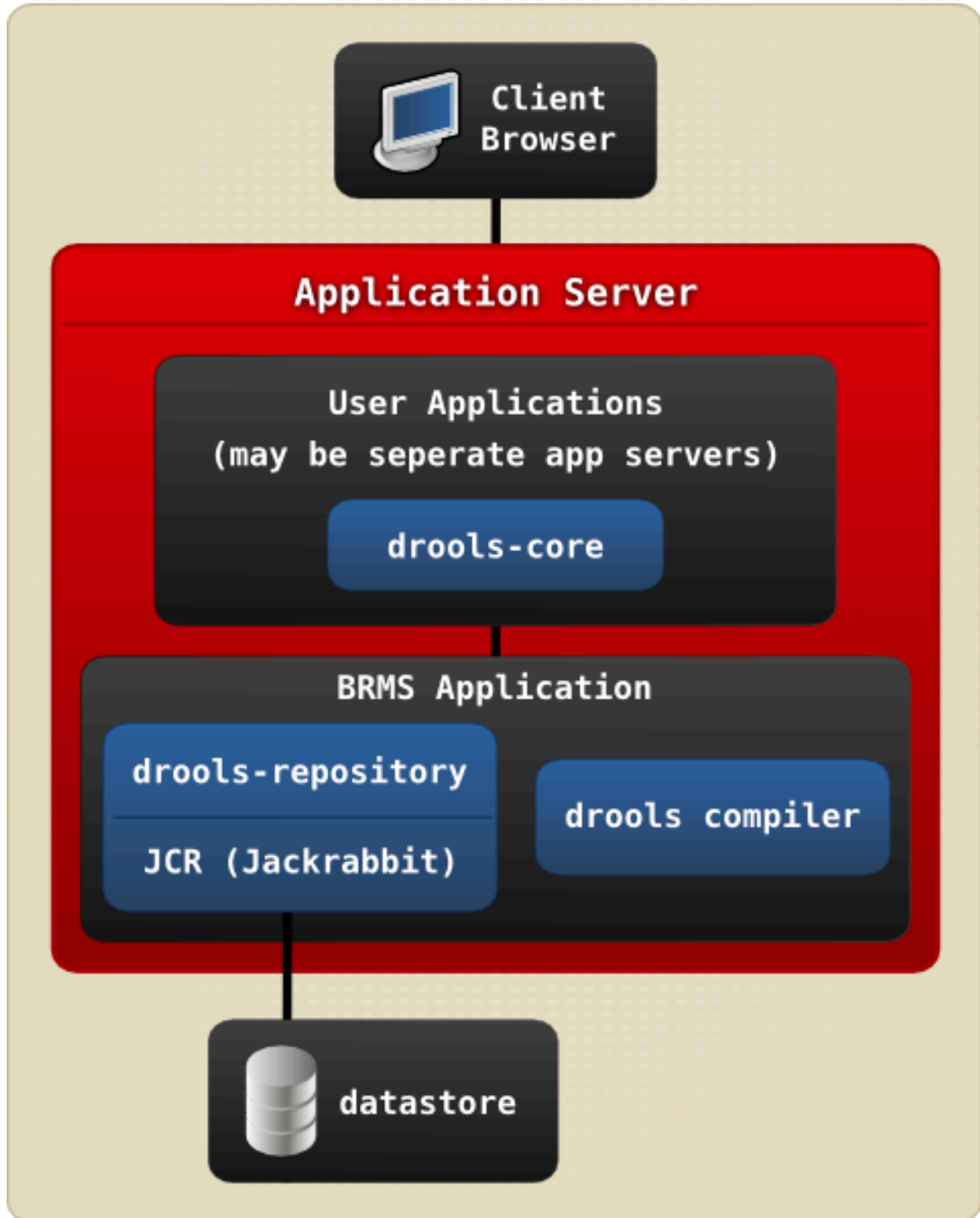


Figure 3.1. Architectural diagram

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 components that configurable, such as the database.

The BRMS is deployed as a war, which provides user interfaces over the web and binary packages via URLs. It utilizes 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.

3.1. Re-usable components

The BRMS uses a service interface to separate the GUI from the back end functionality. In this case the back end both includes the asset repository as well as the compiler specifics for dealing with rules.

The main interface is `RepositoryService`, which is implemented in `ServiceImplementation`. The GWT ajax front end talks to this interface using GWT's asynchronous callback mechanism. The Seam configuration file is `components.xml`.

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`. For those familiar with GWT, each of the *features* can be used separately. The `JBRMSFeature` class and the classes that implement it (they can in theory be stand alone) contain additional information.

[Section 2.6.4, "Customizing the BRMS Platform User Interface"](#) contains information regarding customizing the user interface.

3.2. Versioning and Storage

[Section 2.3, "BRMS Repository Configuration"](#) covers configuration options in some detail 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 jackrabbit, the `.cnd` files are in the source for the node type definitions as some wish to view these. A package is a *folder* and each asset is a file: an asset can either be textual or have a binary attachment.

Quick start guide

4.1. Quick start guide

This section will provide a quick tour of the features of the JBoss Enterprise BRMS Platform. It is assumed that the BRMS Platform and your repository is correctly installed and configured. [Chapter 2, BRMS Installation & Administration](#) contains directions for the installation and configuration of the BRMS Platform.

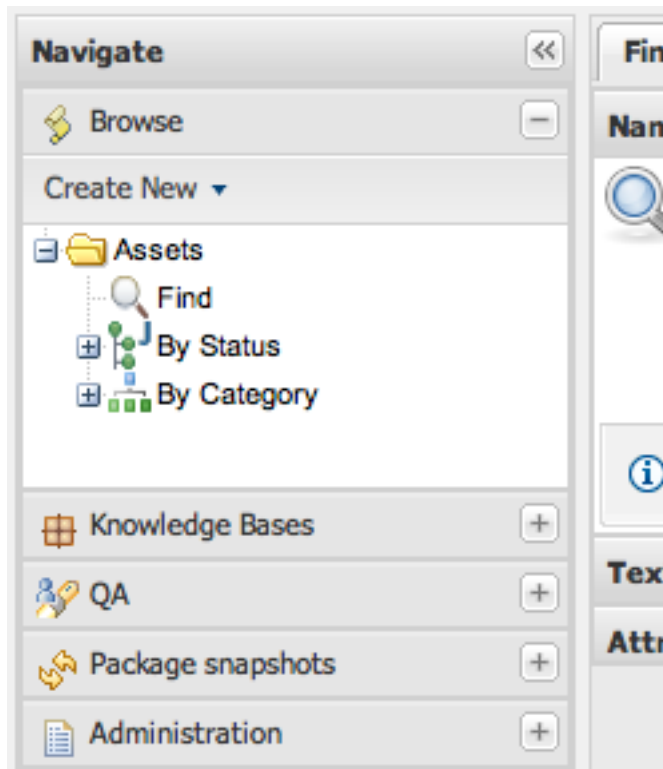


Figure 4.1. Main feature areas of the BRMS Platform

The above picture shows the main feature areas of the BRMS Platform.

- 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)

4.1.1. Supported browsers

The current versions of the major web browsers are supported. This includes Firefox (version 1.5 and greater), Internet Explorer (version 7 and greater), Opera, Safari and Google Chrome. The preferred browser for most platforms is Firefox, it is widely available and free.

4.1.2. 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:

- 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 with. From the Package management feature you can create a new package. Packages should have meaningful names 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. 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).

4.1.3. 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 BRMS 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)

4.1.4. 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).

4.1.5. 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 do not affect the package. It also makes the package available on a URL of the form: "http://<your server>/drools-guvnor/org.drools.guvnor.Guvnor/packages/<packageName>/<snapshotName>" (where you can use that URL and downloads will be covered in the section on deployment).

4.2. BRMS concepts

4.2.1. Rules are assets

As the BRMS 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.

4.2.2. Categorization

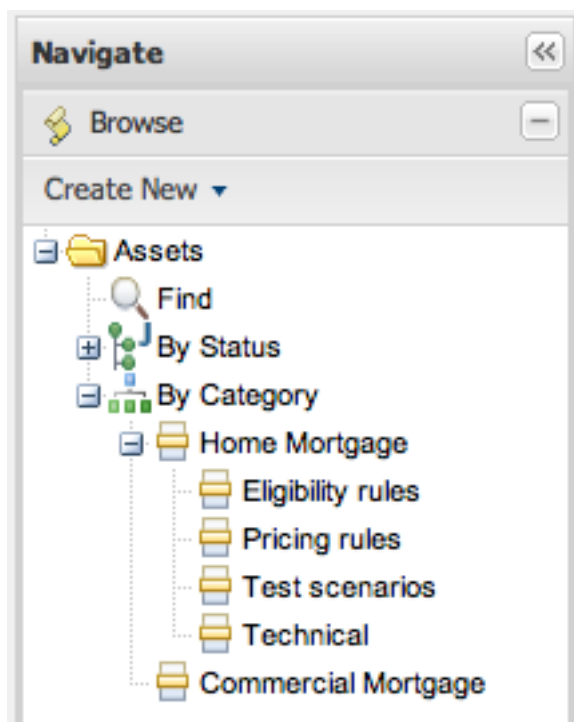


Figure 4.2. Categories

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 BRMS 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".

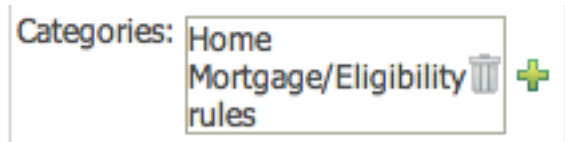


Figure 4.3. Assets can have multiple categories

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 a still a single category, but its 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" - its 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.

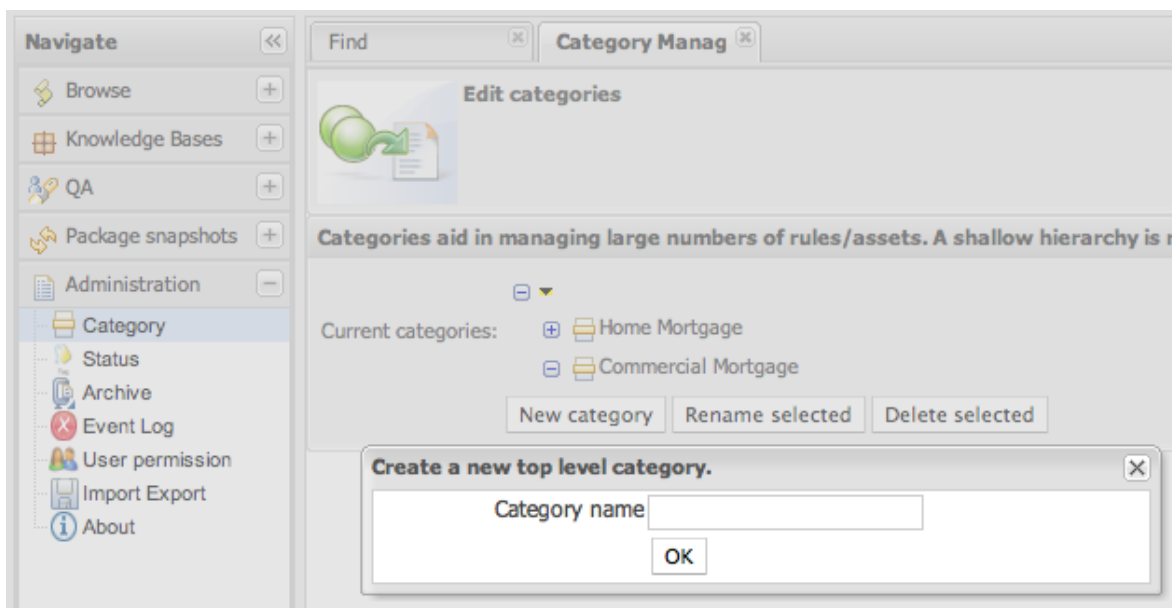


Figure 4.4. Creating categories

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 chose 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. Its OK if its not clear at first, you are free to change categories as you go.

4.2.3. The asset editor

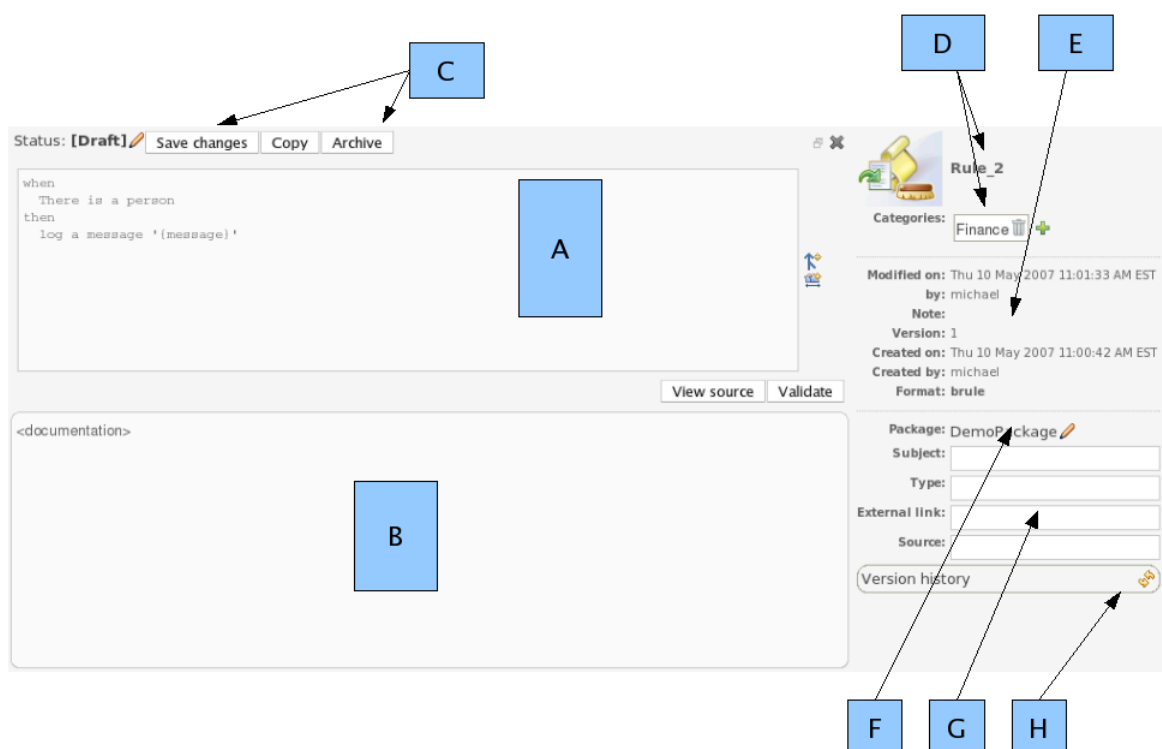


Figure 4.5. The Asset editor view

The above diagram shows the "asset editor" with some annotated areas. The asset editor is where all rule changes are made. Below is a list which describes the different parts of the editor.

- A

This is where the "editor widget" lives - exactly what form the editor takes depends on the asset or rule type.
- B

This is the documentation area - a free text area where descriptions of the rule can live. It is encouraged to write a plain description in the rule here before editing.
- C

These are the actions - for saving, archiving, changing status etc. Archiving is the equivalent of deleting an asset.

- D

This has the asset name, as well as the list of categories that the asset belongs to.

- E

This section contains read-only meta data, including when changes were made, and by whom.

"Modified on:" - this is the last modified date.

"By:" - who made the last change.

"Note:" - this is the comment made when the asset was last updated (i.e. why a change was made)

"Version:" - this is a number which is incremented by 1 each time a change is checked in (saved).

"Created on:" - the date and time the asset was created.

"Created by:" - this initial author of the asset.

"Format:" - the short format name of the type of asset.

- F

This shows what package the asset belongs to. Ownership can also be changed from here.

- G

This is some optional meta data.

- H

This will show the version history list when requested.

4.2.4. Rule authoring

The BRMS Platform supports several formats of assets (rules). Here the key ones are described. Some of these are covered in other parts of the manual, and the detail will not be repeated here.

4.2.4.1. 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.

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

Also note that there is a guided editor in the Eclipse plug in, most of the details in this section can also apply to it.

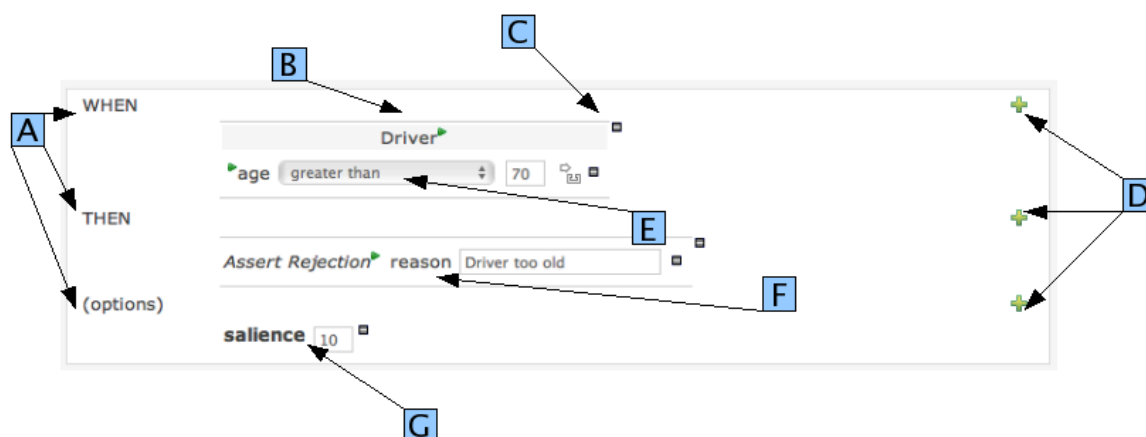


Figure 4.6. The guided BRL editor

The above diagram shows the editor in action. The following description applies to the lettered boxes in the diagram above:

A: The different parts of a rule. The "WHEN" part is the condition, "THEN" action, and "(options)" are 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 "Driver" fact (the fields are listed below, in this case just "age"). Note the green triangle, it will popup a list of options to add to the fact declaration: you can add more fields (eg their "location"), or you can assign a variable name to the fact (which you can use later on if needs be). As well as adding more fields to this pattern - you can add "multiple field" constraints - ie constraints that span across fields (e.g. age > 42 or risk > 2). The popup dialog shows the options.

C: The small "-" icons indicate you can remove something - in this case it would remove the whole Driver fact declaration. If its the one below, it would remove just the age constraint.

D: The "+" symbols allow 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 to add a constraint on a fact (it will give you a list of facts), or you can use another conditional element, the choices which are: "There is no" - which means the fact+constraints must not exist, "There exists" - which means that there exists at least one match (but there only needs to be one - it will not trigger for each match), and "Any of" - which means that any of the patterns can match (you then add patterns to these higher level patterns). 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 "age" field. (Looking from left to right) the green triangle allows you to "assign" a variable name to the "age" field, which you may use later on in the rule. Next is the list of constraint operations - this list changes depending on the data type. After that is the value field - the value field will be one of: a) a literal value (e.g. number, text), b) a "formula" - in which case it is an expression which is calculated (this is for advanced users) or b) a variable (in which case a list will be provided to choose values from). After this there is a horizontal arrow icon, this is for "connective constraints": these are constraints which allow you to have alternative values to check a field against, for example: "age is less than 42 or age is not equal to 39" is possibly this way.

F: This shows an "action" of the rule, a rule consists of a list of actions. In this case, we are asserting/inserting a new fact, which is a rejection (with the "reason" field set to an explanation). There are quite a few other types of actions you can use: you can modify an existing fact (which tells the engine the

fact has changed) - or you can simply set a field on a fact (in which case the engine doesn't know about the change - normally because you are setting a result). You can also retract a fact. In most cases the green arrow will give you a list of fields you can add so you can change the value. The values you enter are "literal" - in the sense that what you type is what the value is. If it needs to be a calculation, then add an "=" at the start of the value - this will be interpreted as a "formula" (for advanced users only) ! and the calculation will be performed (not unlike a spreadsheet).

G: This is where the rule options live. In this case, only salience is used which is a numeric value representing the rules "priority". This would probably be the most common option to use.

4.2.4.1.1. User driven drop down lists

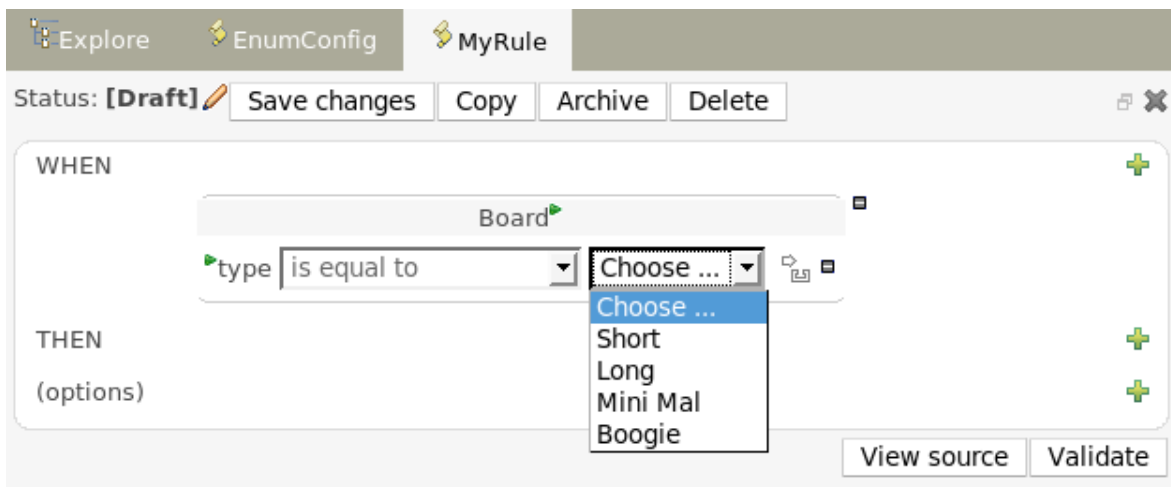


Figure 4.7. Data enumeration showing as a drop down list

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.

4.2.4.1.2. 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:

Figure 4.8. DSL in guided editor

4.2.4.1.3. A more complex example:

Figure 4.9. A more complex BRL example

In the above example, you can see it is using a mixture of literal values, and formulas. The second constraint on the "Person" fact, is a formula (in this case it is doing a silly calculation on the persons age, and checking something against their name - both "age" and "name" are fields of the Person fact in this case. In the 3rd line (which says "age is less than .." - it is also using a formula, although, in this case the formula does a calculation and returns a value (which is used in the comparison) - in the former case, it had to return True or False (in this case, its a value). Obvious formulas are basically pieces of code - so this is for experienced users only.

Looking at the "Board" pattern (the second pattern with the horizontal grey bar): this uses a top level conditional element ("There is no") - this means that the pattern is actually looking for the "non existence" of a fact that matches the pattern. Note the "Any of:" - this means that EITHER the "type" field constraint is matched, or the "name" field is matched (to "myname" in the case above). This is what is termed a Multiple field constraint (you can nest these, and have it as complex as you like, depending on how much you want the next person to hate you: Some paraphrased advice: Write your rules in such as way as if the person who has to read/maintain them is a psychopath, has a gun, and knows where you live).

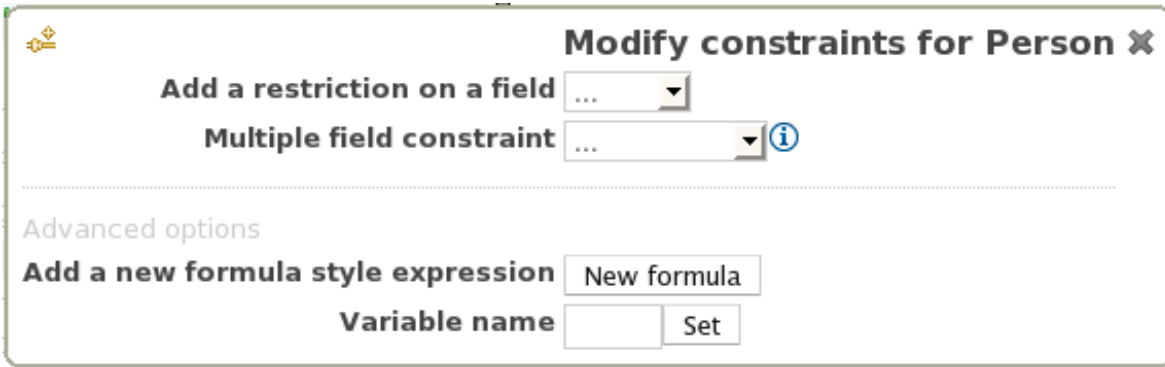


Figure 4.10. Adding constraints

The above dialog is what you will get when you want to add constraints to the Person fact. In the top half are the simple options: you can either add a field straight away (a list of fields of the Person fact will be shown), or you can add a "Multiple field constraint" - of a given type (which is described above). The Advanced options: you can add a formula (which resolves to True or False - this is like in the example above: "age < (age * 2) ..."). You can also assign a Variable name to the Person fact (which means you can then access that variable on the action part of the rule, to set a value etc).

4.2.4.2. DSL rules

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

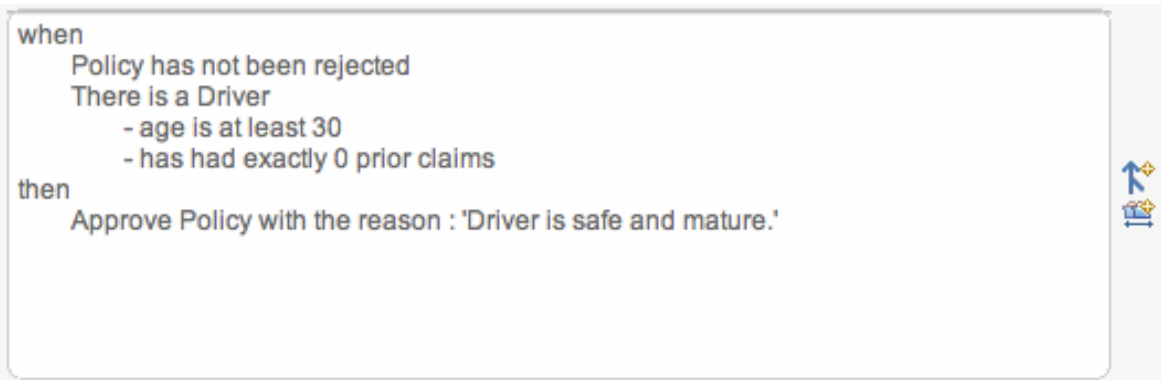


Figure 4.11. DSL rule

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).

4.2.4.3. 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 4.12. Spreadsheet decision table

To use a spreadsheet, you upload an xls (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).

4.2.4.4. 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.

Decision table							
Modify... ▾							
	Description	Advertiser type	age is at least	Postcode gre...	Postcode les...	Set the value...	Set the rea...
Advertiser type: Agency (3 Items)							
1	Good suburbs	Agency	10	4000	4100	→ 42	Loyal
2	Good suburbs	Agency		2000	2100	→ 43	Good region
4	Good suburbs	Agency		2200	2300	→ 43	Good region
Advertiser type: Partner (1 Item)							
3	Partners	Partner	1			→ 49	Other

Figure 4.13. Decision table

At the top right there is a button which shows the configuration area of the guided decision table:

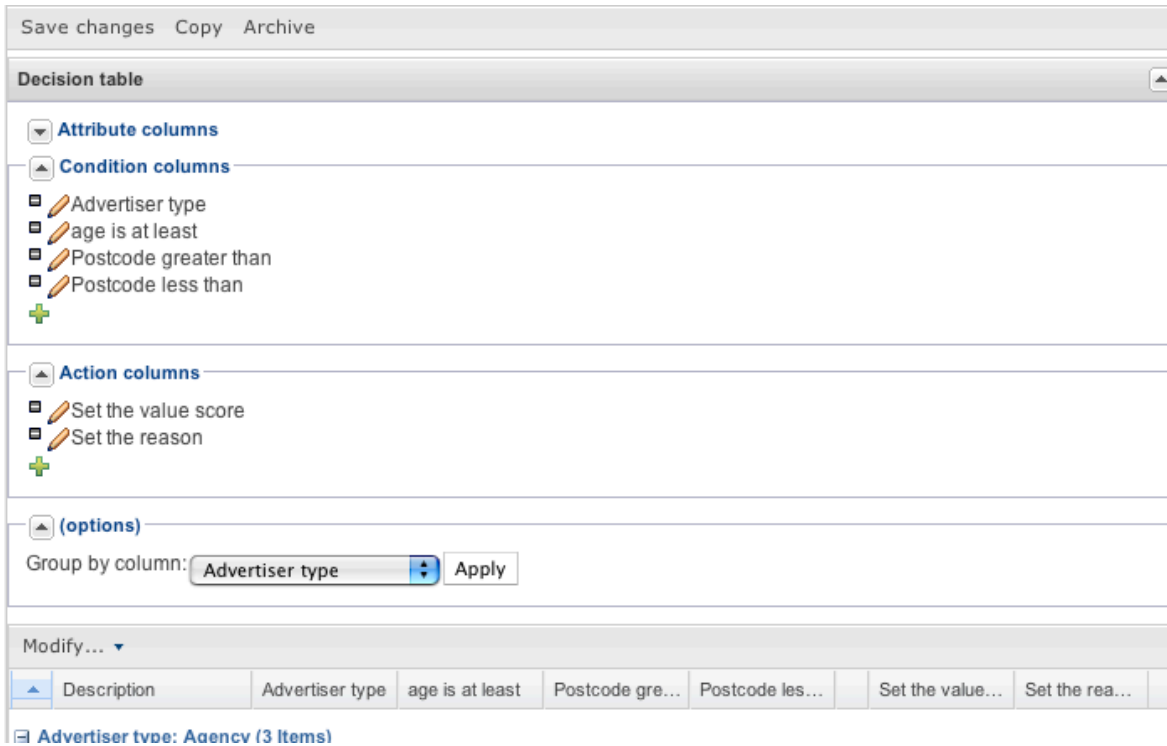


Figure 4.14. Decision table config

It is in this section where condition and action columns are configured. "Attribute columns" are for setting attributes on a per rule (row) basis, such as salience. Web based decision tables are compiled into DRL like all other rule assets.

4.2.4.5. Technical rules (drl)

Technical (drl) rules are stored as text - they can be managed in the BRMS. A DRL can either be a whole chunk of rules, or an individual rule. If it's 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 any typical package of rules, you generally have a been for some "technical rules" - you can mix and match all the rule types together of course.

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

Figure 4.15. DRL technical rule

4.2.4.6. 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 4.16. Function

4.2.4.7. 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 '=').

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

Figure 4.17. 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 BRMS 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 BRMS. Instead of specifying a list of values in the BRMS 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.

4.2.4.8. 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
com.yourco.DataHelper()).getListOfAges("@{dependentField1}",
"@{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. 'depenentField1' 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".

4.2.5. Templates of assets/rules

Tip: As you may have many similar rules, you can create rule templates, which are simply rules which are kept in an inactive package - you can then categories templates accordingly, and copy them as needed (choosing a live package as the target package).

4.2.6. Status management

Each asset (and also package) in the BRMS has a status flag set. The values of the status flag are set in the Administration section of the BRMS. (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 life-cycle of assets (which you can alternatively do with categories if you like).

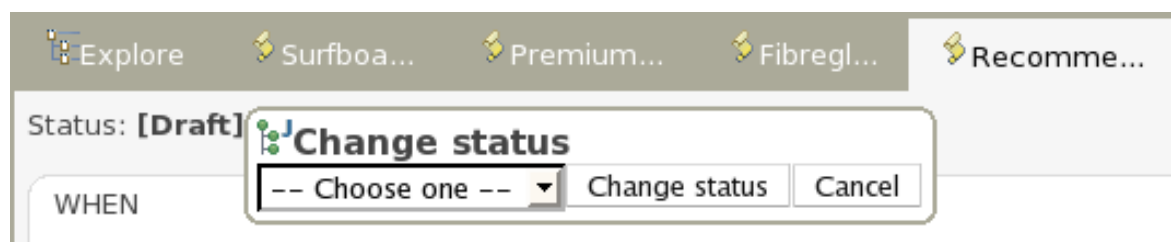


Figure 4.18. Asset status

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

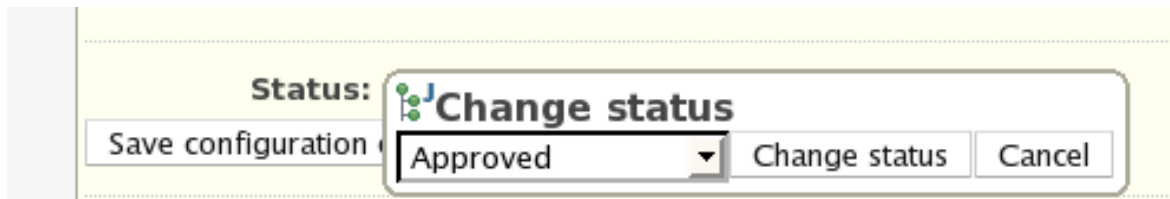


Figure 4.19. Asset status

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).

4.2.7. Package management

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 BRMS - 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.

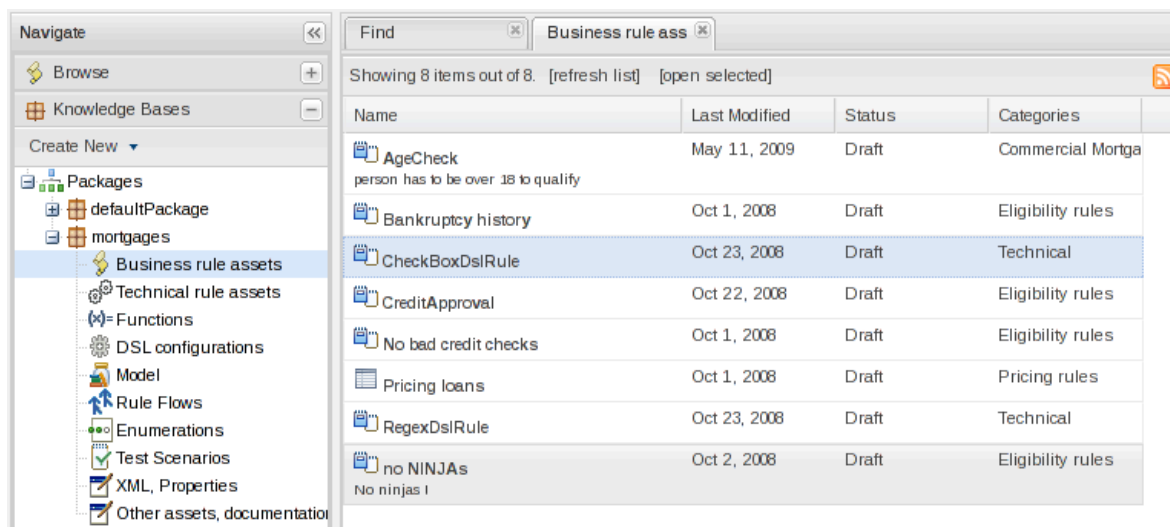


Figure 4.20. The package explorer

The above picture shows the package explorer. Clicking on an asset type will show a list of matches (for packages with thousands of rules, showing the list may take several seconds - hence the importance of using categories to help you find your way around).

So whilst rules (and assets in general) can appear in any number of categories, they only live in one package. If you think of the BRMS 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".

The package management feature allows you to see a list of packages, and then "expand" them, to show lists of each "type" of asset (there are many assets, so some of them are grouped together):

The asset types:

- Business assets: this shows a list of all "business rule" types, which include decision tables, business rules etc. etc.
- Technical assets: this is a list of items that would be considered technical (e.g. DRL rules, data enumerations and rule flows).
- Functions: In the BRMS you can also have functions defined (optionally of course).
- DSL: Domain Specific Languages can also be stored as an asset. If they exist (generally there is only one), then they will be used in the appropriate editor GUIs.
- Model: A package requires at least one model - for the rules.

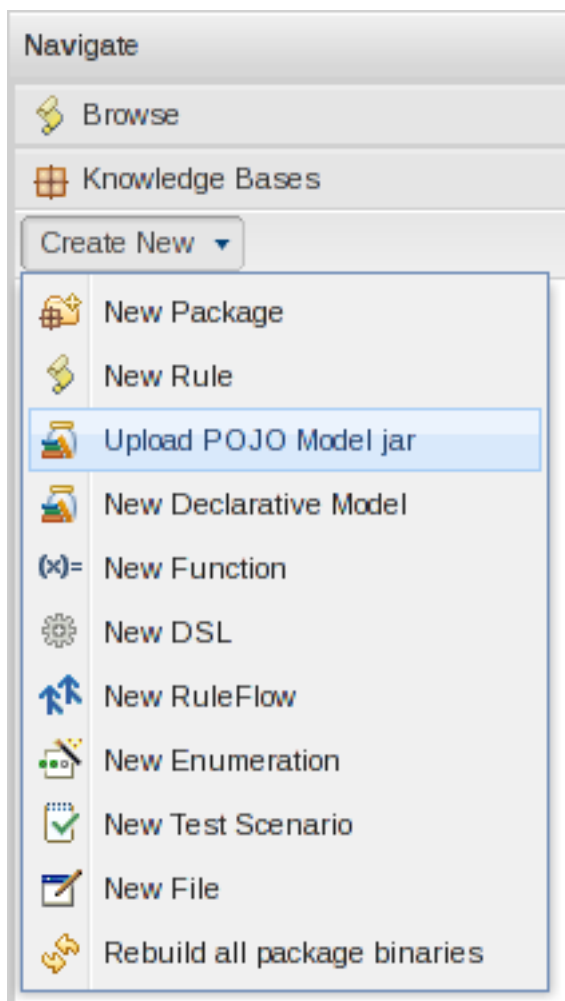


Figure 4.21. Creating new assets

From the package explorer you can create new rules, or new assets. Some assets you can only create from the package explorer. The above picture shows the icons which launch wizards for this purpose. If you hover the mouse over them, a tooltip will tell you what they do.

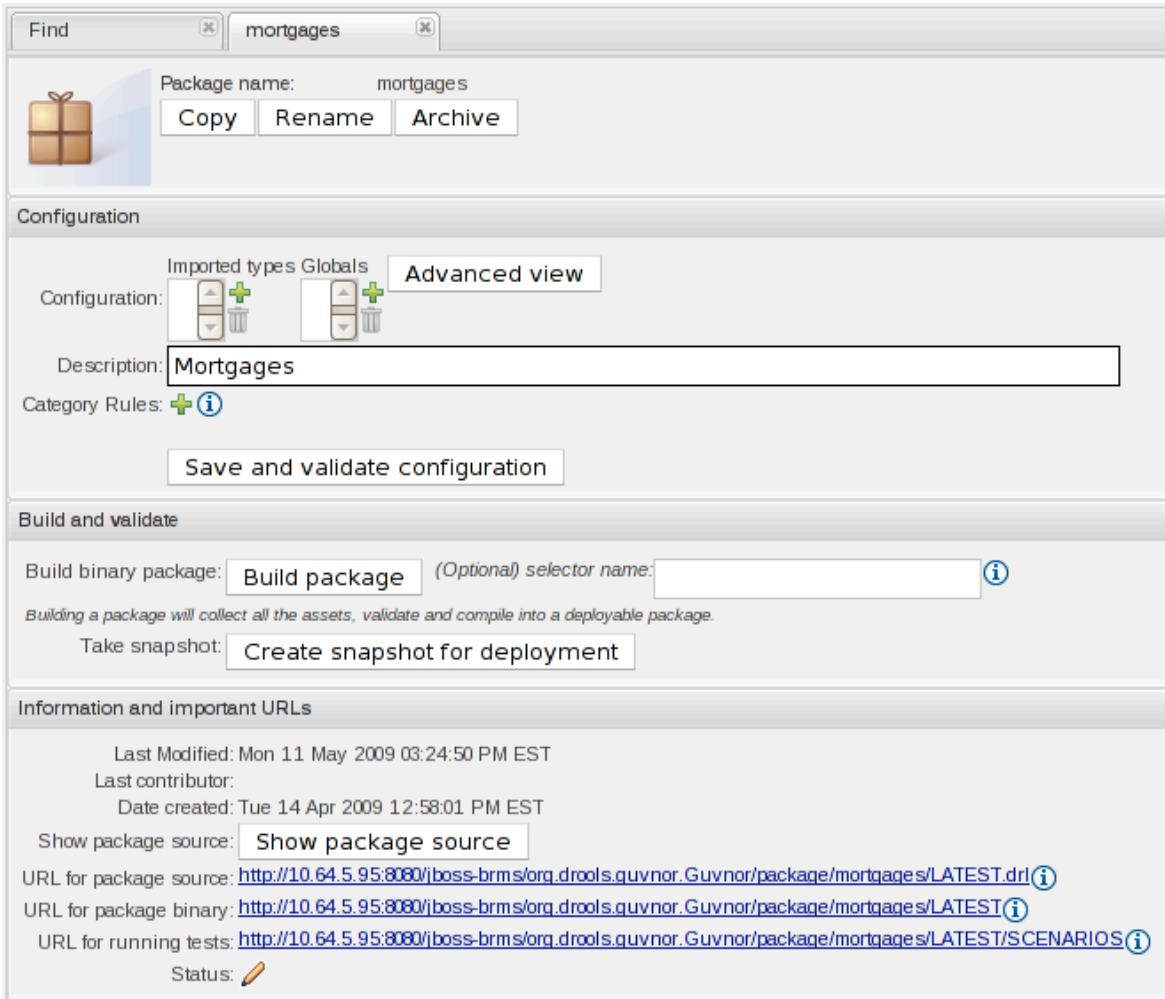


Figure 4.22. Package configuration

One of the most critical things you need to do is configure packages. This is mostly importing the classes used by the rules, and global variables. Once you make a change, you need to save it, and that package is then configured and ready to be built. 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.

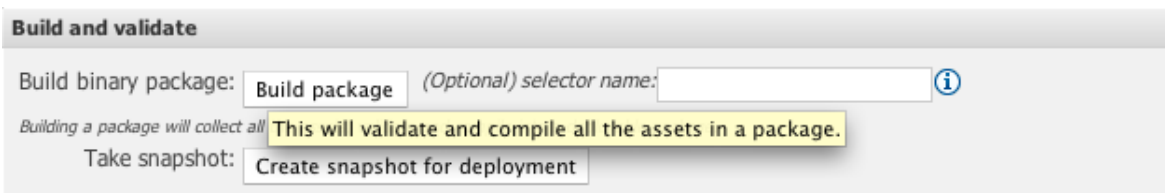


Figure 4.23. Package building

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. WARNING: in cases of large numbers of rules, all these operations can take some time.

It is optional at this stage to enter the name of a "selector" - see the admin section for details on how to configure custom selectors for your system (if you need them - selectors allow you to filter down

what you build into a package - if you don't know what they are for, you probably don't need to use them).

4.2.7.1. 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 BRMS 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.

4.2.8. Version management

Both assets and whole packages of assets are *versioned* in the BRMS, 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 4.24. 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.

4.2.9. Deployment management

Snapshots, URLs and binary packages:

URLs are central to how built packages are provided. The BRMS provides packages via URLs (for download and use by the Rule Agent). These URLs take the form of: `http://<server>/drools-guvnor/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 Rule 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.

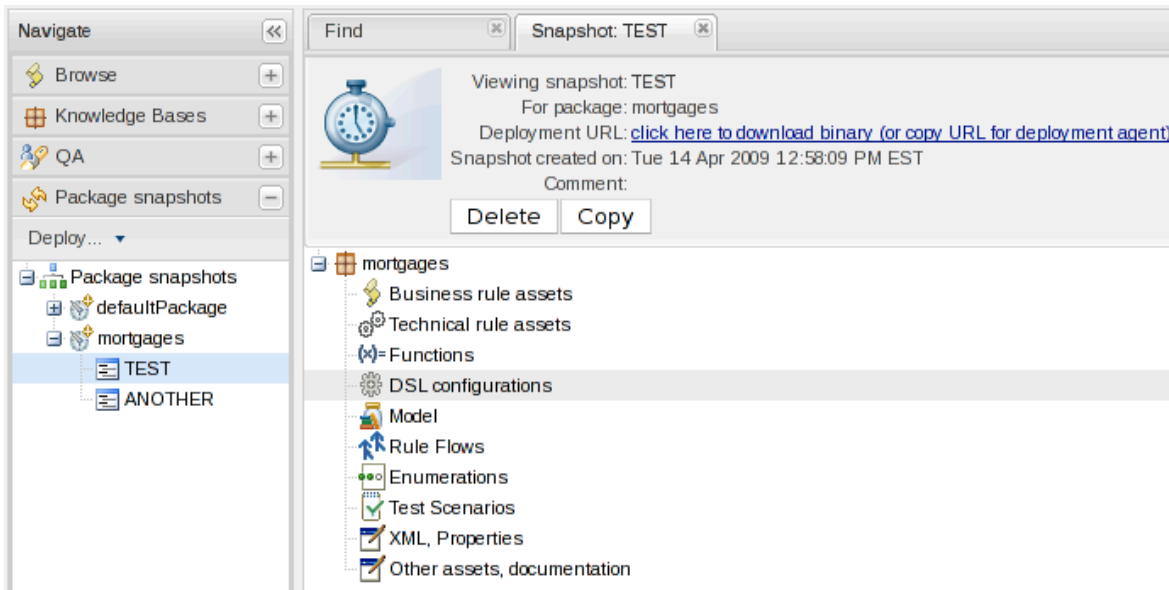


Figure 4.25. Deployment snapshots

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.

4.2.10. Navigating and finding rules

The two main ways of viewing the repository are by using user-driven Categorization (tagging) as outlined above, and the package explorer view.

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

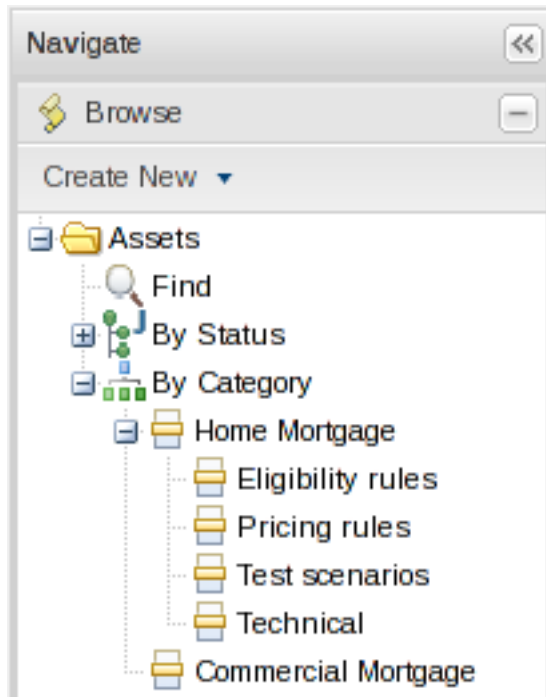


Figure 4.26. Category view

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

The alternative and more technical view is to use the package explorer. This shows the rules (assets) closer to how they are actually stored in the database, and also separates rules into packages (name spaces) and their type (format, as rules can be in many different formats).

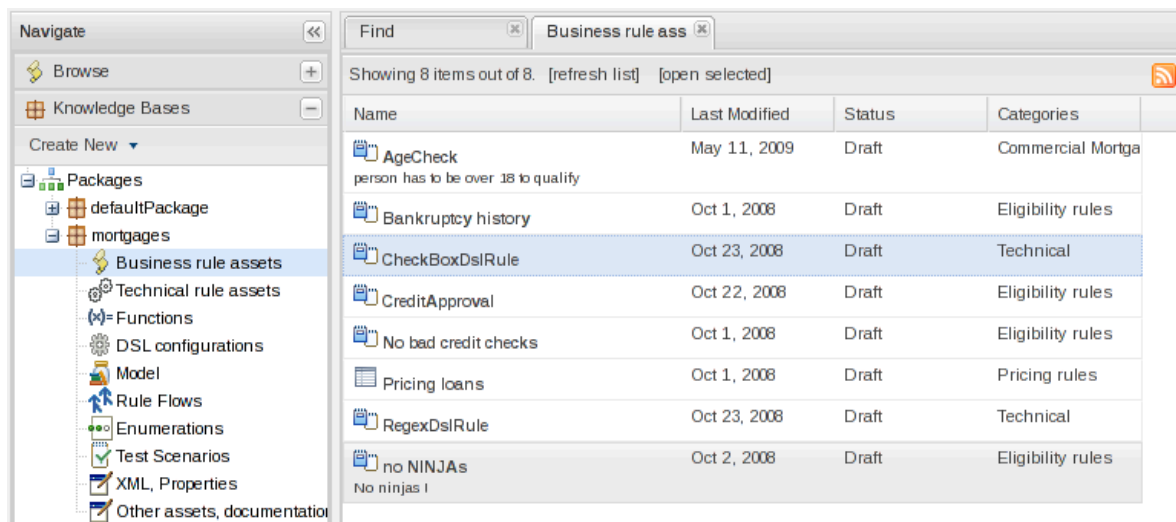


Figure 4.27. Package view

The above shows the alternate way of exploring - using packages.

4.3. 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.

4.4. The fact model (object 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 2 ways to do this: you can upload jar files containing classes which your application and the rules both use, or you can use models that are declared along with the rules.

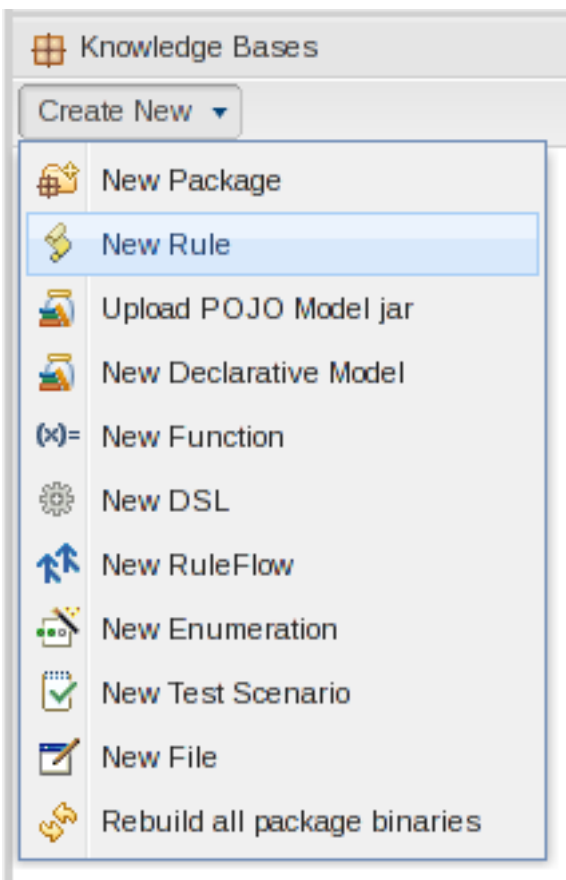


Figure 4.28. Choosing a model type

When a jar is uploaded, it will add import statements to the package configuration (you can then review and change them).

Using declared models, you will see an editor like the following:

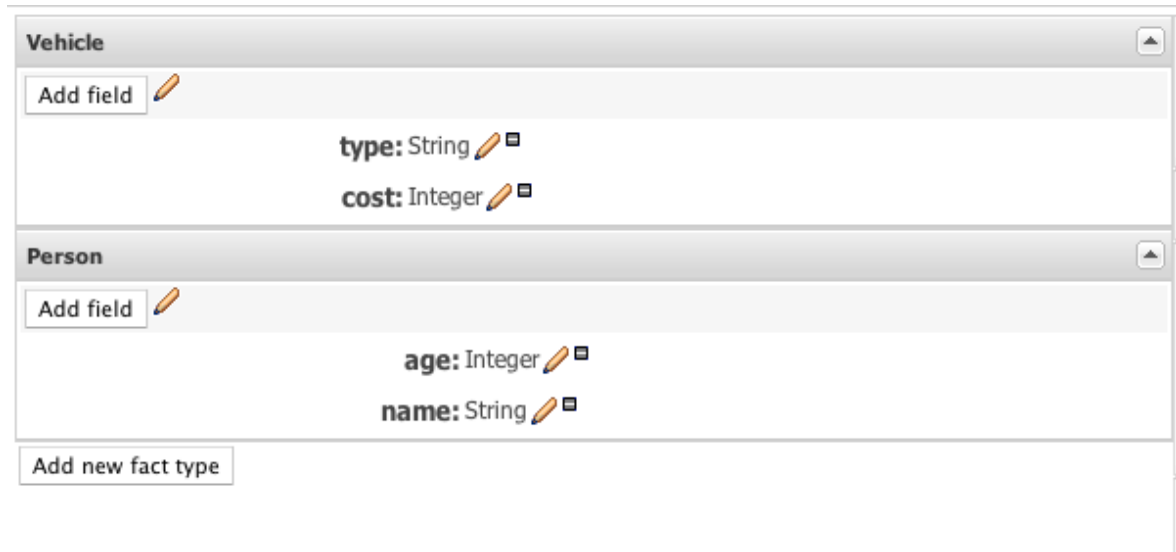


Figure 4.29. Choosing a model type

In here you can define types, and add fields (each field has a type). The type of a field is suggested by a list (but this list is not exhaustive):

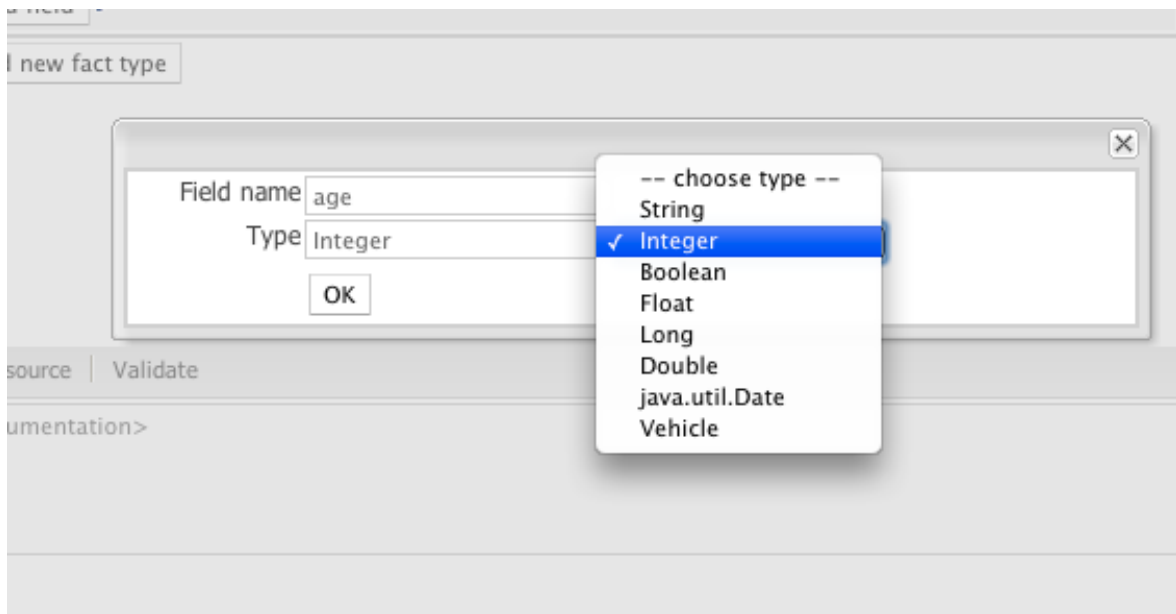


Figure 4.30. Choosing a model type

These fact models can be used like normal fact objects, however the way you create them is different (as they are not on your applications classpath). To create these objects, they are available from the RuleBase instance.

```
// Retrieve the generated fact type
FactType cheeseFact =
    ruleBase.getFactType( "org.drools.generatedbeans.Cheese" );

// Create a new Fact instance
Object cheese = cheeseFact.newInstance();
```

```
cheeseFact.set( cheese, "type", "stilton" );
```

The "cheese" object above can then be inserted into working memory just like a normal POJO based fact.

Note that the namespace of the declared type is the package namespace where it was declared (in the above case "org.drools.generatedbeans").

Why would you chose declared types over jar files: generally this reinforces the fact that the model "belongs" to the rulebase, rather than the application, and allows the model to have a life-cycle separate from the application. It also removed the hassle of keeping jar files in sync between rules and the applications that use the rules.

4.5. The business user perspective

You can see from this manual, that some expertise and practice is required to use the BRMS Platform. 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 the BRMS Platform 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 the BRMS Platform 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).

4.6. Deployment: Integrating rules with your applications

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

4.6.1. The Rule Agent

The rule agent is a component which is embedded in the core runtime of the rules engine. To use this, you don't need any extra components. In fact, if you are using the BRMS Platform, your application should only need to include the 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 the BRMS Platform at all - the drools-ant task will generate .pkg files the same as the BRMS Platform.

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

To use the rule agent, you will use a call in your applications code like:

```
RuleAgent agent = RuleAgent.newRuleAgent("/MyRules.properties");
RuleBase rb = agent.getRuleBase();
rb.newStatefulSession....
//now assert your facts into the session
```

IMPORTANT: You should only have one instance of the RuleAgent per rulebase you are using. This means you should (for example) keep the agent in a singleton, JNDI (or similar). In practice most people are using frameworks like Seam or Spring - in which case they will take care of managing this for you (in fact in Seam - it is already integrated - you can inject rulebases into Seam components). Note that the RuleBase can be used multiple times by multiple threads if needed (no need to have multiple copies of it).

This assumes that there is a MyRules.properties in the root of your classpath. You can also pass in a Properties object with the parameters set up (the parameters are discussed next).

The following shows the content of MyRules.properties:

```
##
## RuleAgent configuration file example
##

newInstance=true
file=/foo/bar/boo.pkg /foo/bar/boo2.pkg
dir=/my/dir
url=http://some.url/here http://some.url/here
localCacheDir=/foo/bar/cache
poll=30

name=MyConfig
```

You can only have one type of key in each configuration (eg only one "file", "dir" etc - even though you can specify multiple items by space separating them). Note also, instead of a discrete properties file, you can construct a java.util.Properties object, and pass it in to the RuleBase methods.

Referring to the above example, the "keys" in the properties are:

newInstance

Setting this to "true" means that the RuleBase instance will be created fresh each time there is a change. this means you need to do agent.getRuleBase() to get the new updated rulebase (any existing ones in use will be untouched). The default is false, which means rulebases are updated "in place" - ie you don't need to keep calling getRuleBase() to make sure you have the latest rules (also any StatefulSessions will be updated automatically with rule changes).

file

This is a space-separated list of files - each file is a binary package as exported by the BRMS Platform. You can have one or many. The name of the file is not important. Each package must be in its own file.

It is also possible to specify .drl files - and it will compile it into the package. However, note that for this to work, you will need the drools-compiler dependencies in your applications classpath (as opposed to just the runtime dependencies).

Please note that if the path has a space in it, you will need to put double quotes around it (as the space is used to separate different items, and it will not work otherwise). Generally spaces in a path name are best to avoid.

dir

This is similar to file, except that instead of specifying a list of files you specify a directory, and it will pick up all the files in there (each one is a package) and add them to the rulebase. Each package must be in its own file.

Please note that if the path has a space in it, you will need to put double quotes around it (as the space is used to separate different items, and it will not work otherwise). Generally spaces in a path name are best to avoid.

url

This is a space separated list of URLs to the BRMS Platform which is exposing the packages (see below for more details).

localCacheDir

This is used in conjunction with the url above, so that if the BRMS Platform is down (the url is not accessible) then if the runtime has to start up, it can start up with the last known "good" versions of the packages.

poll

This is set to the number of seconds to check for changes to the resources (a timer is used).

name

This is used to specify the name of the agent which is used when logging events (as typically you would have multiple agents in a system).

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

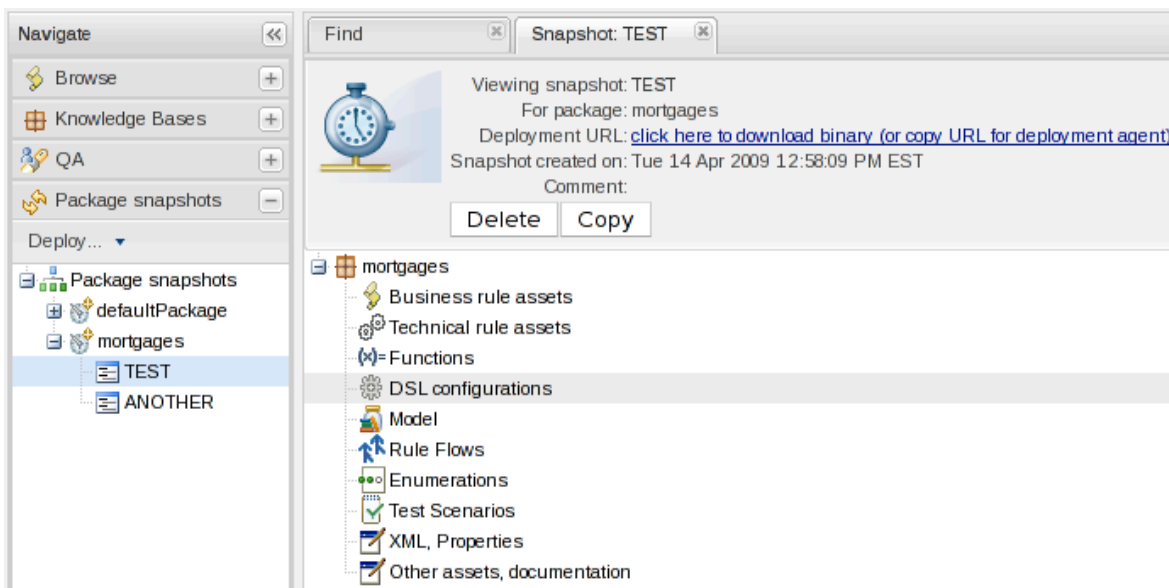


Figure 4.31. Snapshot deployment

You can see the "Package URI" - this is the URL that you would copy and paste into the agent .properties 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 RuleAgent if needed (in some cases people will not want to have the runtime automatically contact the BRMS Platform for updates - but that is generally the easiest way for many people).

4.6.2. JMS, SOAP/WSDL integration

A stateless rule execution server may not be enough for your needs. If you require WSDL+SOAP, JMS or other integration, JBoss ESB (<http://www.jboss.org/JBossESB/>) can be used.

The esb can provide stateful and stateless rule services via multiple transport mechanisms (such as http/soap, or JMS, and many many more) as needed. Load balancing, failover, monitoring and more is also provided. Refer to the ESB site (<http://www.jboss.org/JBossESB/>) for more information on this (it will not be covered in this manual).

4.6.3. Manual deployment

This section is only needed for advanced users who are integrating deployment into their own mechanism. Normally you should use the rule agent.

For those who do not wish to use the automatic deployment of the RuleAgent, "rolling your own" is quite simple. The binary packages emitted by the BRMS Platform are serialized Package objects. You can de-serialize them and add them into any rulebase - essentially that is all you need to do.

From the BRMS Platform, binary packages are provided either from the latest version of a package (once you have successfully validated and built a package) or from the deployment snapshots. The URLs that the BRMS Platform web application exposes provide the binary package via http. You can also issue a "HEAD" command to get the last time a package was updated.

4.7. WebDAV and HTTP

The repository back end can also be accessed via WebDav. WebDAV is a HTTP based file system API. Most modern operating systems provide integrated support for accessing WebDav shares, including Microsoft Windows, MacOS X and Linux. You refer to your operating system vendors documentation for configuration directions. There are also many third-party WebDav clients are available for most platforms.

The URL for accessing your repository using WebDav is the same as the web interface with / **webdav/** at the end instead of **Guvnor.html**, e.g.

```
http://10.64.5.95:8080/jboss-brms/org.drools.guvnor.Guvnor/webdav/
```

Authentication is required as normal. WebDav provides a packages and snapshots directory. The snapshots directory is read only, a view of created snapshots of packages. The packages directory contains a list of packages in the repository as directories which in turn contain the individual assets as files.

4.8. URLs

There are a few other URLs which are handy to know exist. The package deployment URL mentioned in the section about rule 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.

Eclipse Integration

The Eclipse Guvnor Tools (EGT) provides an interface for developers to read, write, add and remove assets from BRMS Platform repository server using the Eclipse IDE. EGT provides developers with an interface similar to traditional source control systems such as Subversion. These tools are also provided with the JBoss Developer Studio (JBDS) product.

The BRMS Platform repository and EGT are not intended to replace a dedicated source control system in your development environment but rather provide a convenient means of access for developers.



Important

Guvnor is the name of the open-source project on which the JBoss Enterprise BRMS Platform has been built. The Eclipse Guvnor Tools were developed with it in mind and so refer to the rules repository as "Guvnor". Any place where EGT refers to "Guvnor" it can be considered equivalent to "BRMS Platform". To avoid confusion the remainder of this chapter will use term "Guvnor".

5.1. Source Code and Plug-in Details

The source code for the EGT is available at: <http://anonsvn.jboss.org/repos/labs/labs/jbosrules/trunk/drools-eclipse/>

EGT consist of two plug-ins: **org.guvnor.tools**, **org.eclipse.webdav** and requires Eclipse 3.3.x. The Eclipse Drools plug-ins are also useful for viewing repository resources such as rule definitions, but not required for operation of the EGT.

5.2. Functionality Overview

The EGT contains two views – Repository Explorer and Version History – that are be the center of most interaction with the Guvnor repository.

The "Guvnor Repository Exploring" perspective is provided as a suggested layout. It can be accessed from the **Open Perspective** dialog (**Window -> Open Perspective -> Other...**).

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 **Guvnor Repository Explorer** provides access to the Guvnor repository resources in a browsable tree format. The **Guvnor Resource History** view shows revisions of specific resources available in the repository.

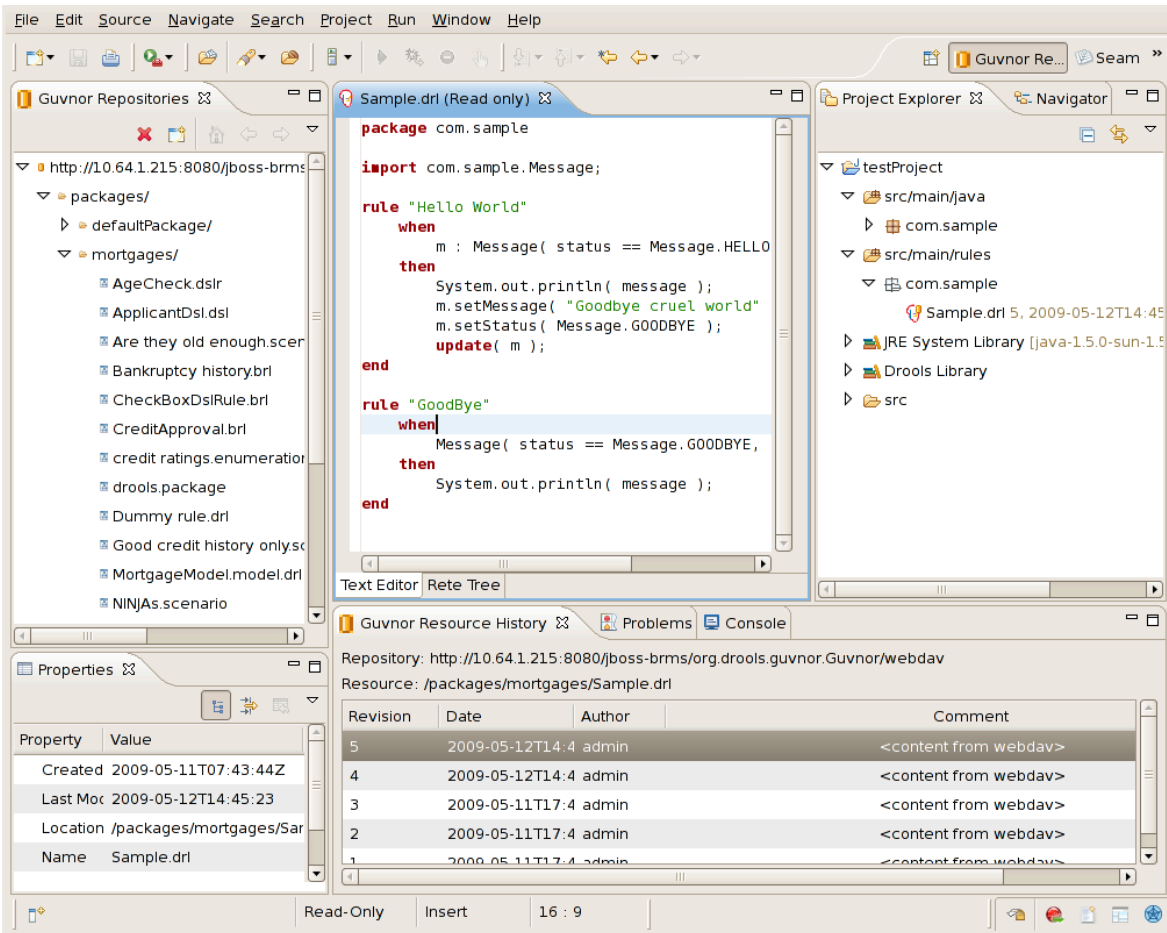


Figure 5.1. "Guvnor Repository Exploring" Perspectives

5.3. 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:

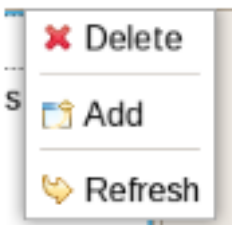


Figure 5.2. Connection wizard

or the menu button:

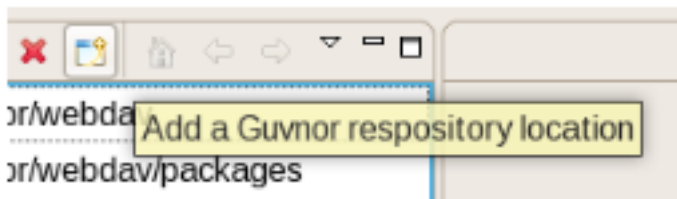



Figure 5.3. Connection wizard

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"/>

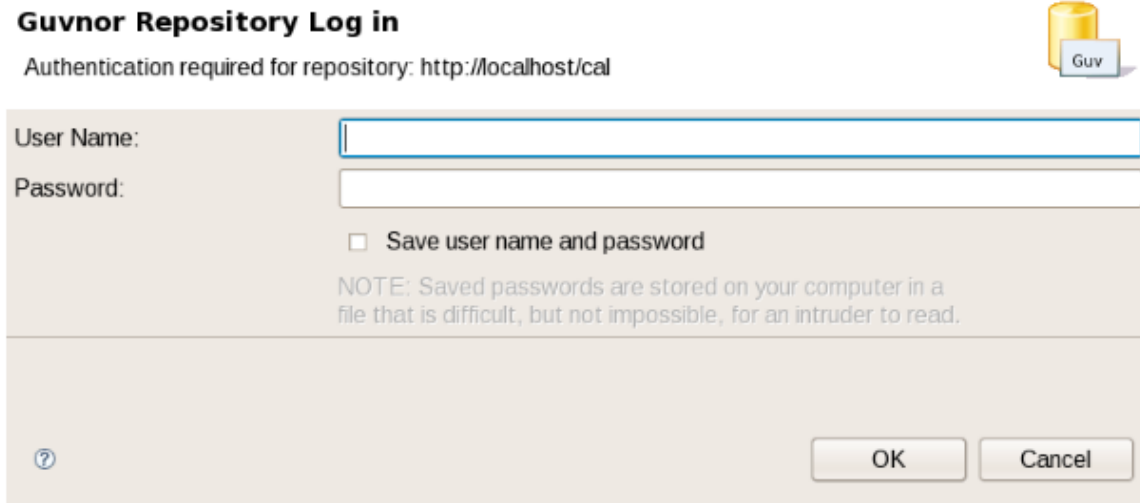
Save user name and password

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

Figure 5.4. Connection wizard

Default values appear in the Location, Port, and Repository fields. (See the “Guvnor Preferences” section below 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/drools-guvnor/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:



The image shows a dialog box titled "Guvnor Repository Log in". At the top right, there is a yellow cylinder icon with a "Guv" label. Below the title, it says "Authentication required for repository: http://localhost/cal". The dialog has two input fields: "User Name:" and "Password:". Below the password field is a checkbox labeled "Save user name and password". A note below the checkbox reads: "NOTE: Saved passwords are stored on your computer in a file that is difficult, but not impossible, for an intruder to read." At the bottom left is a help icon (a question mark in a circle). At the bottom right are "OK" and "Cancel" buttons.

Figure 5.5. 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. 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 un-synchronized 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 repository connection will appear in the **Guvnor Repository Explorer**. You can then expand the tree to view repository contents.

5.4. Guvnor Repository Explorer

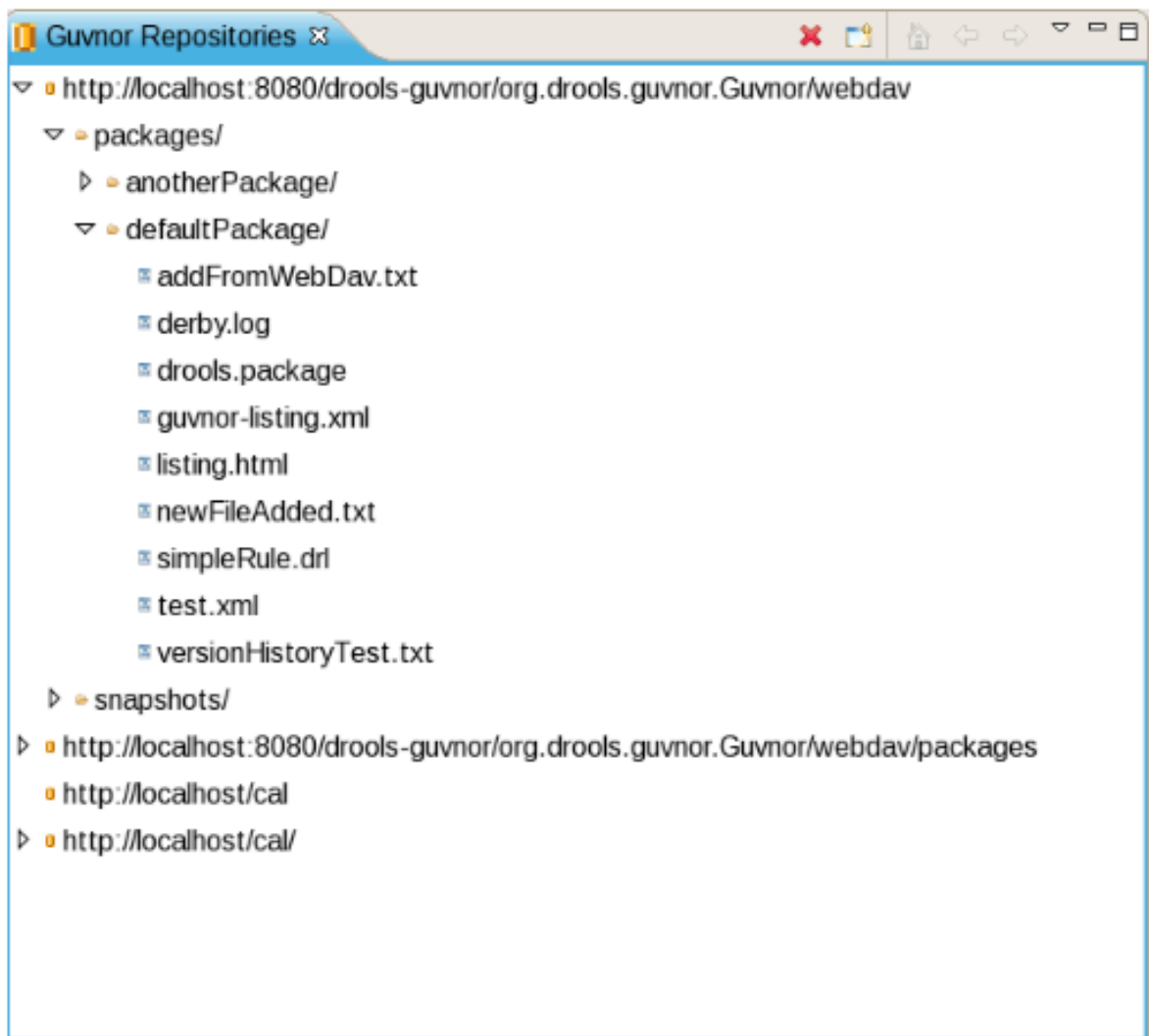


Figure 5.6. Explorer

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:

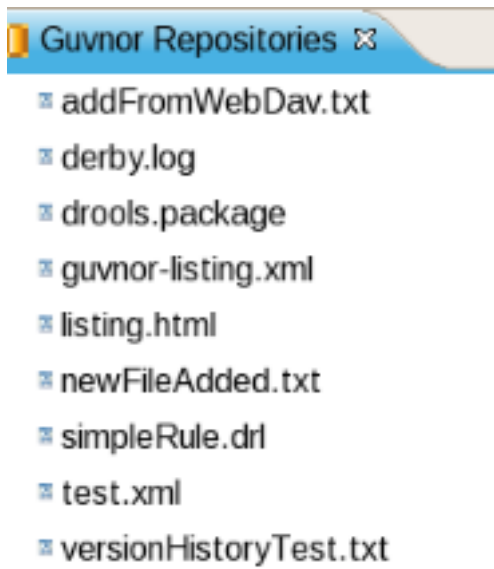


Figure 5.7. Explorer

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:

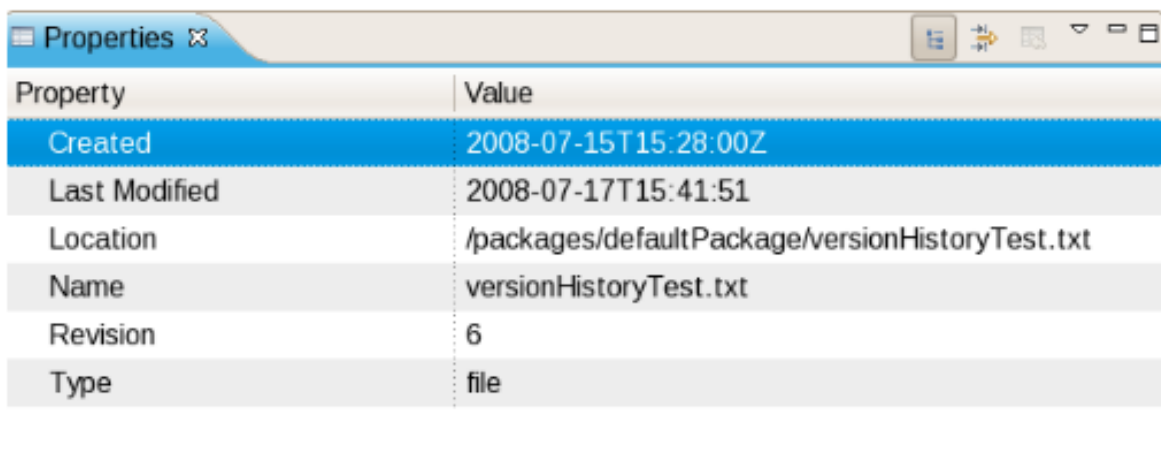


Figure 5.8. Properties

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:

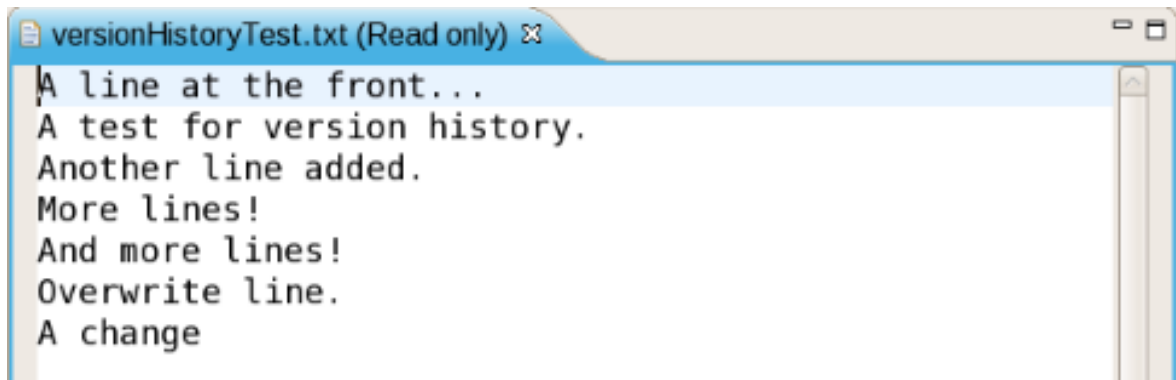


Figure 5.9. Comments

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.)

5.5. 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 methods 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:

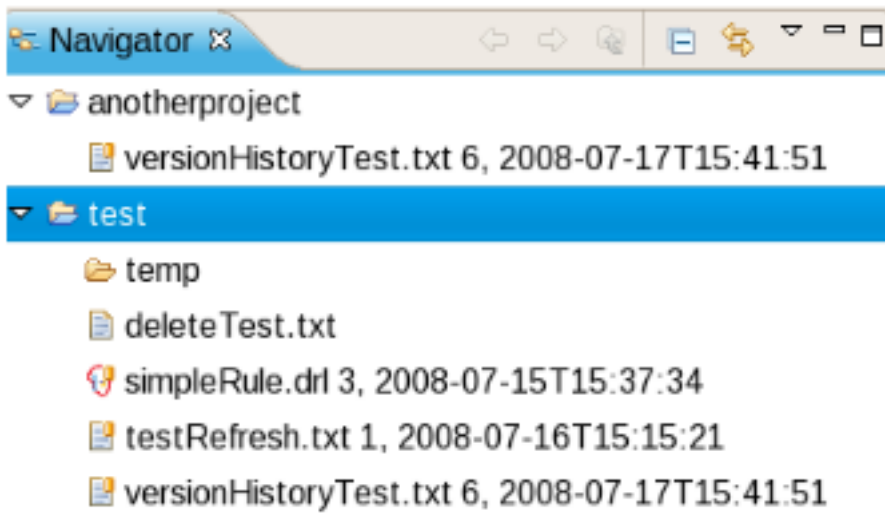


Figure 5.10. Navigator

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 Preferences” below 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:

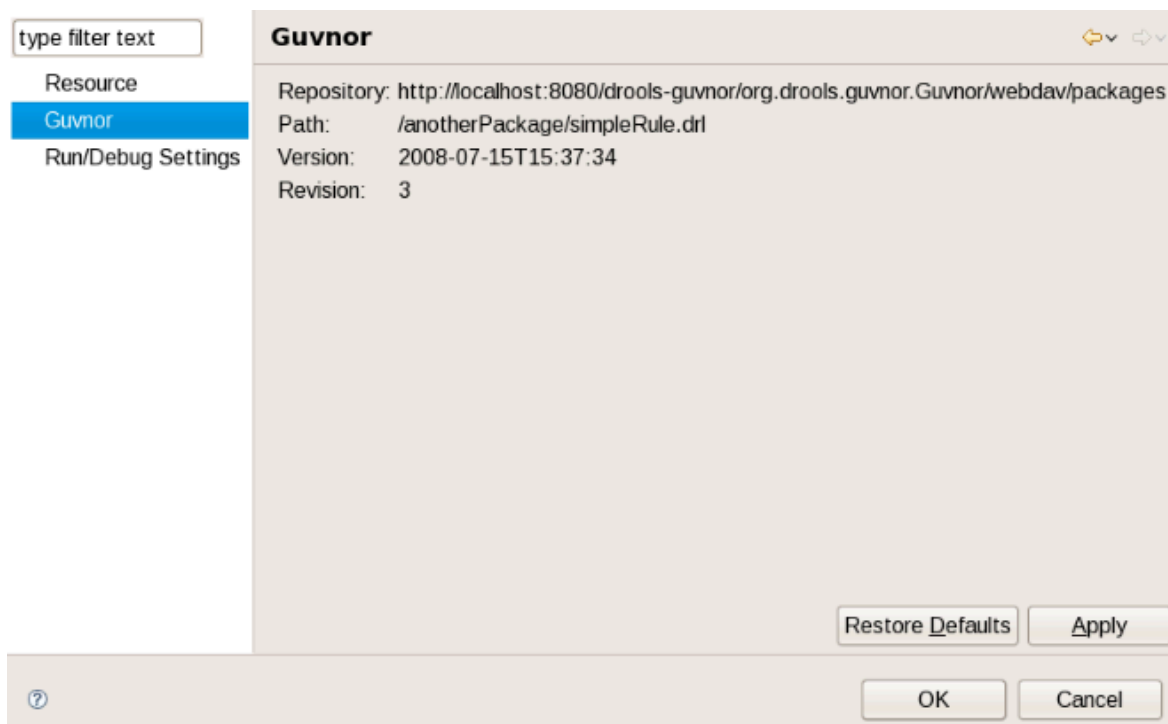


Figure 5.11. Properties

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.

5.6. 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
```



Figure 5.12. 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

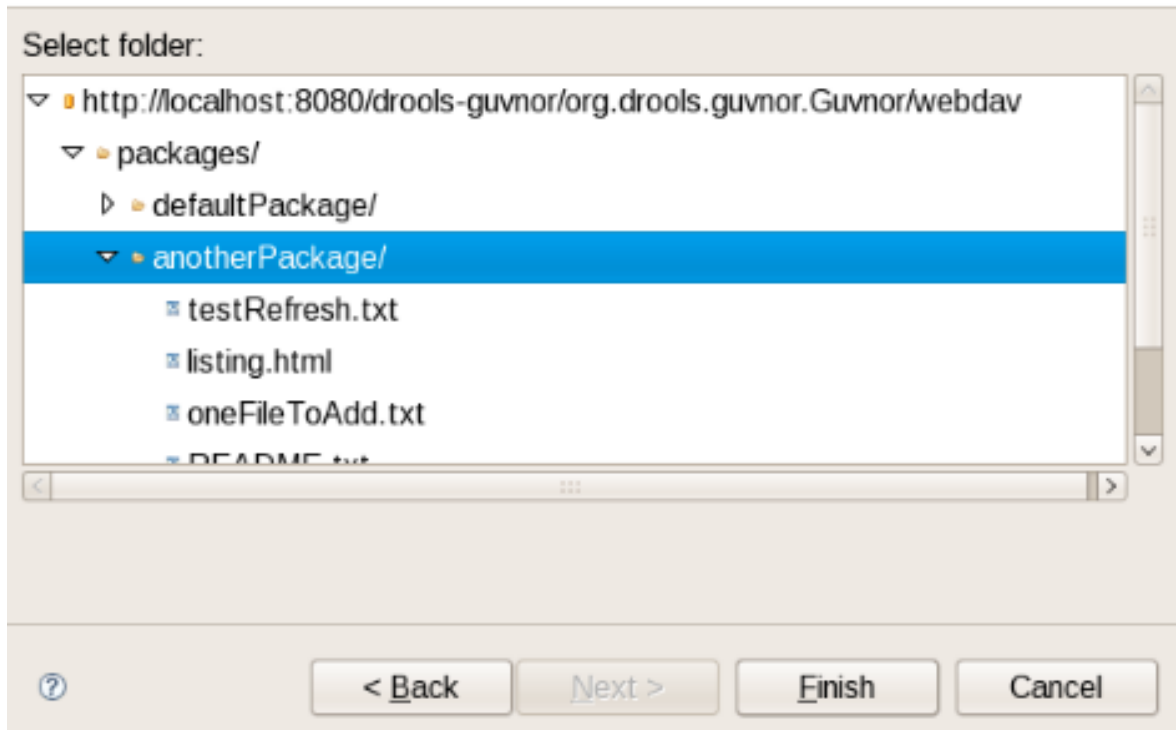


Figure 5.13. Add action

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):

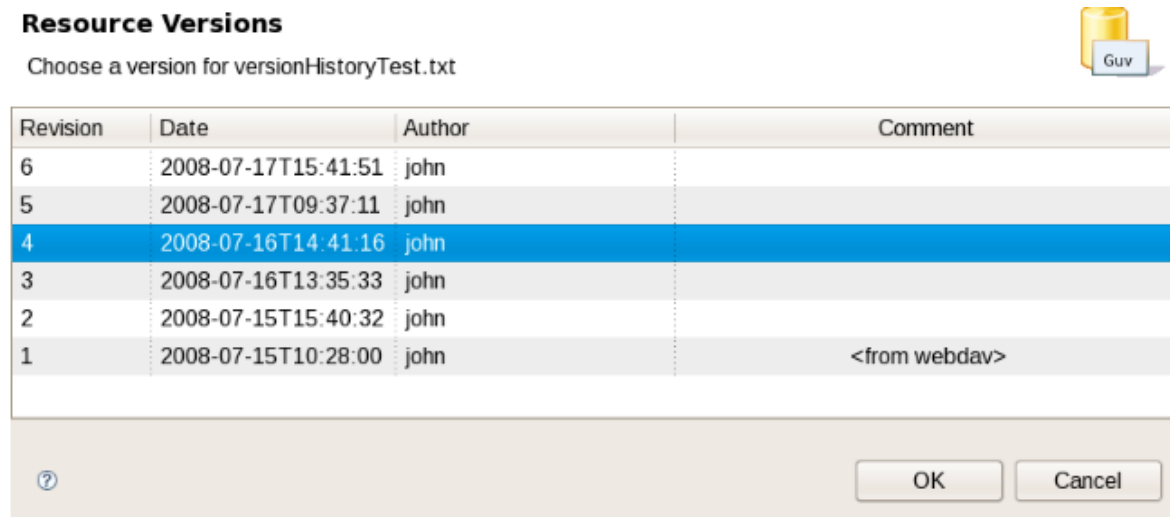


Figure 5.14. Compare

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

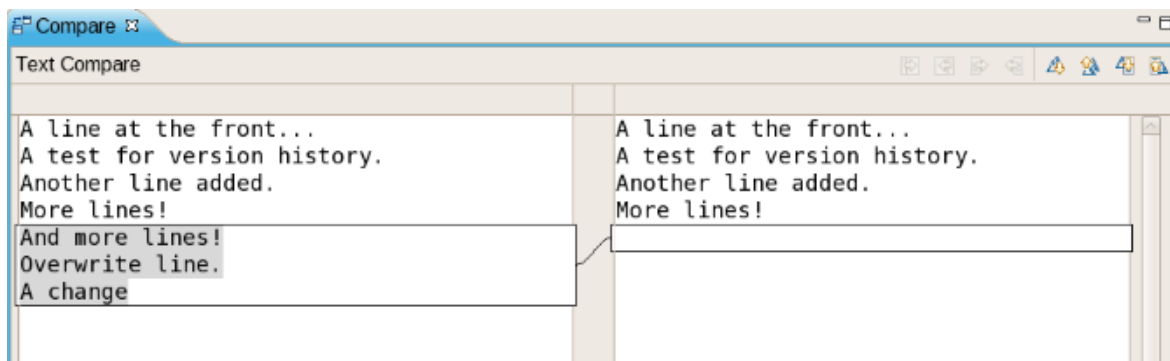


Figure 5.15. Compare

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:

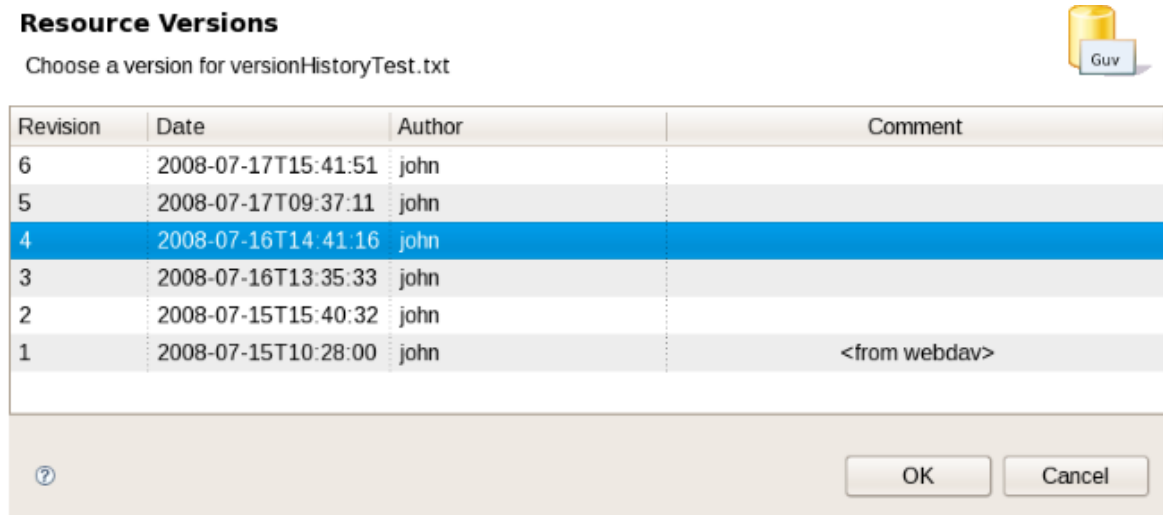


Figure 5.16. Versions

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:

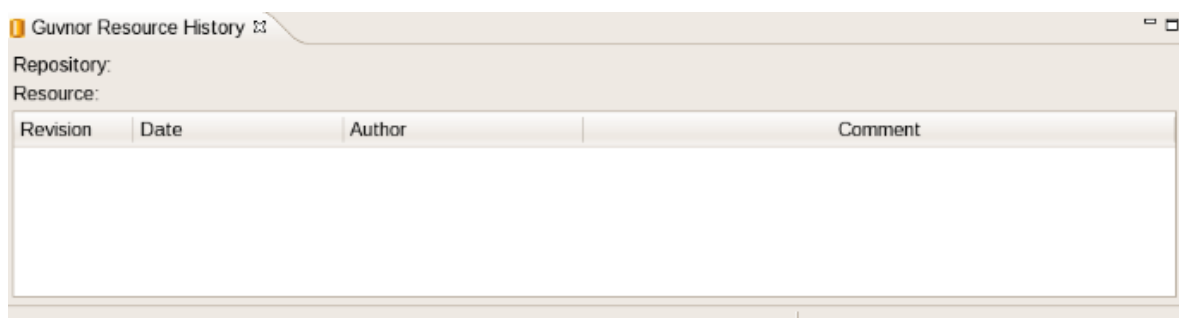
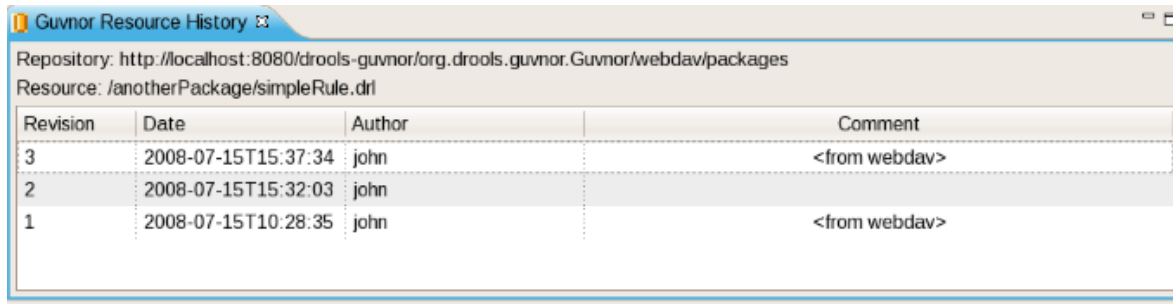


Figure 5.17. History

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 5.18. 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.)

5.7. 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`

Figure 5.19. Import

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

Select resources

Select resources to copy from the Guvnor repository

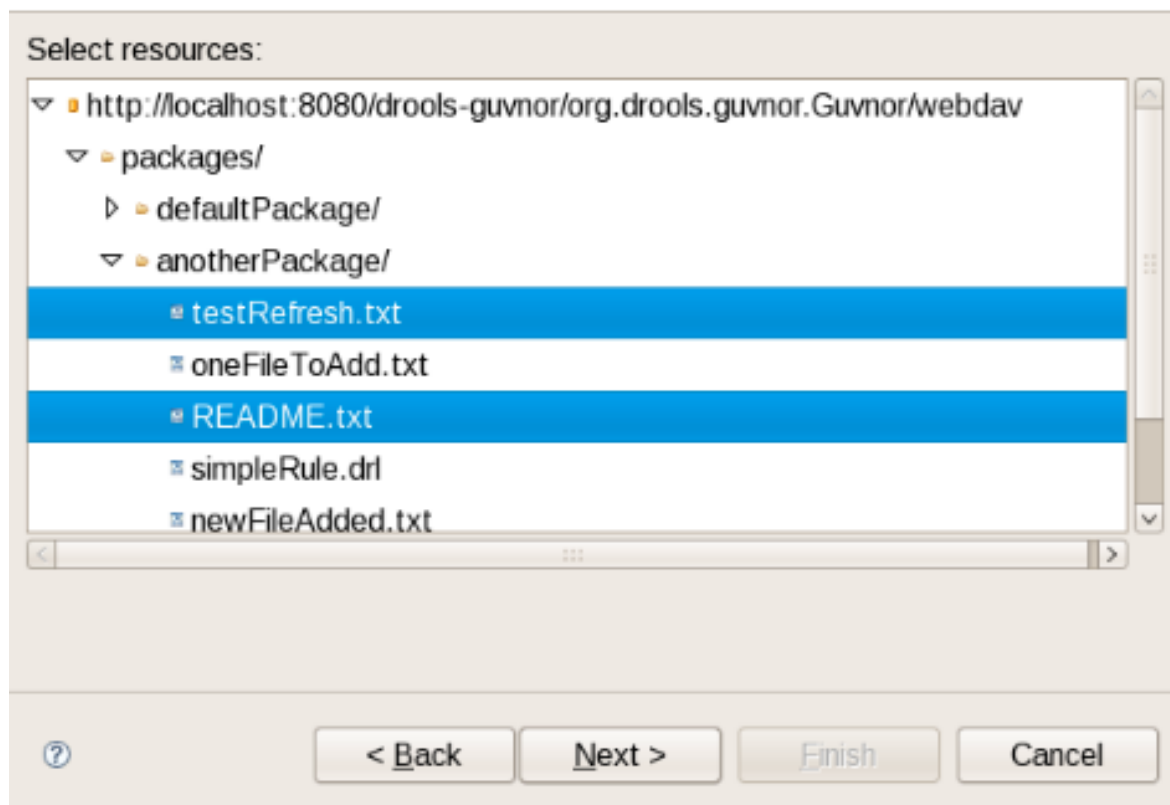


Figure 5.20. Import

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

Select copy location

Select the destination location

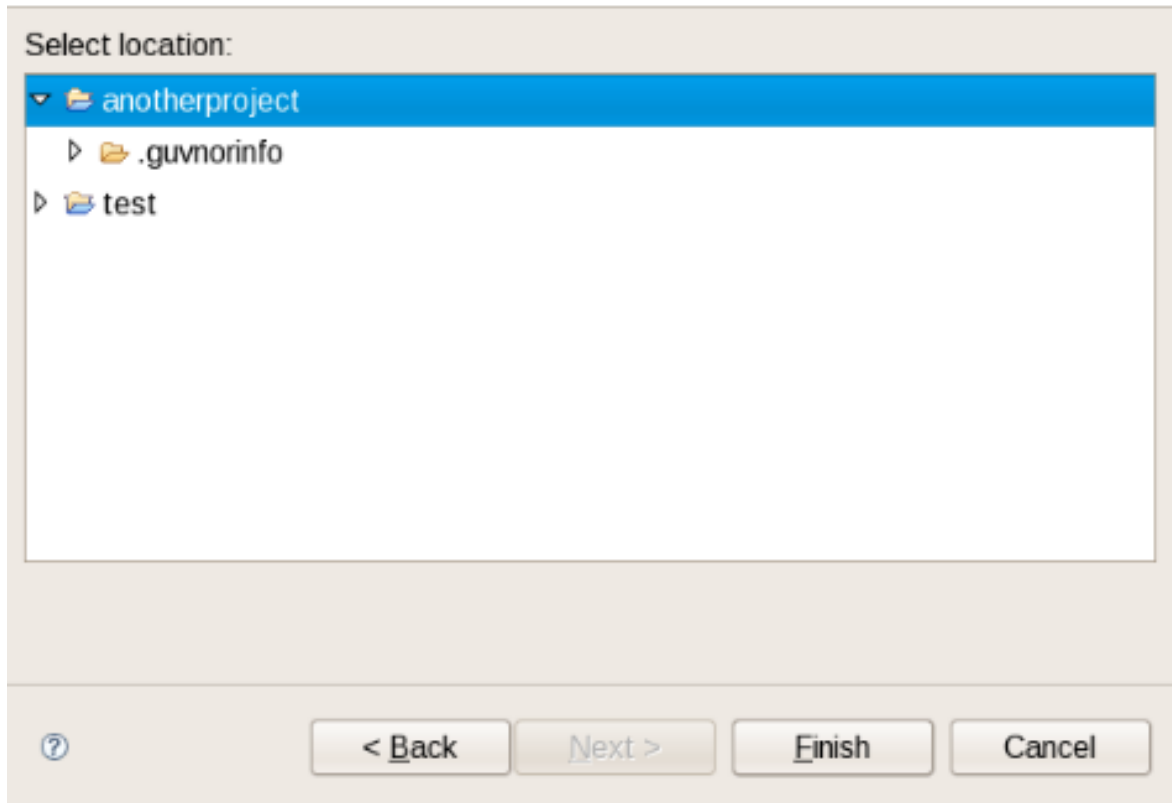


Figure 5.21. Import

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:

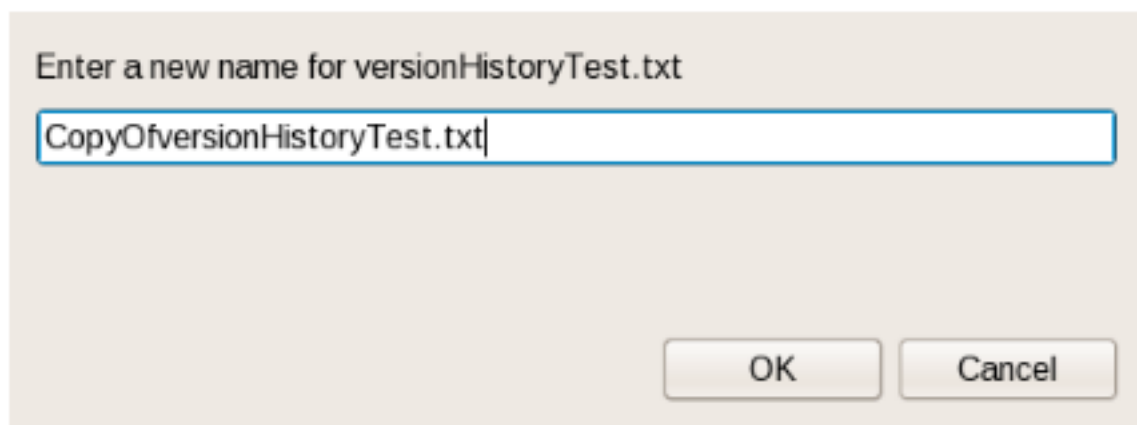


Figure 5.22. Copy

5.8. Guvnor plugin Preferences

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

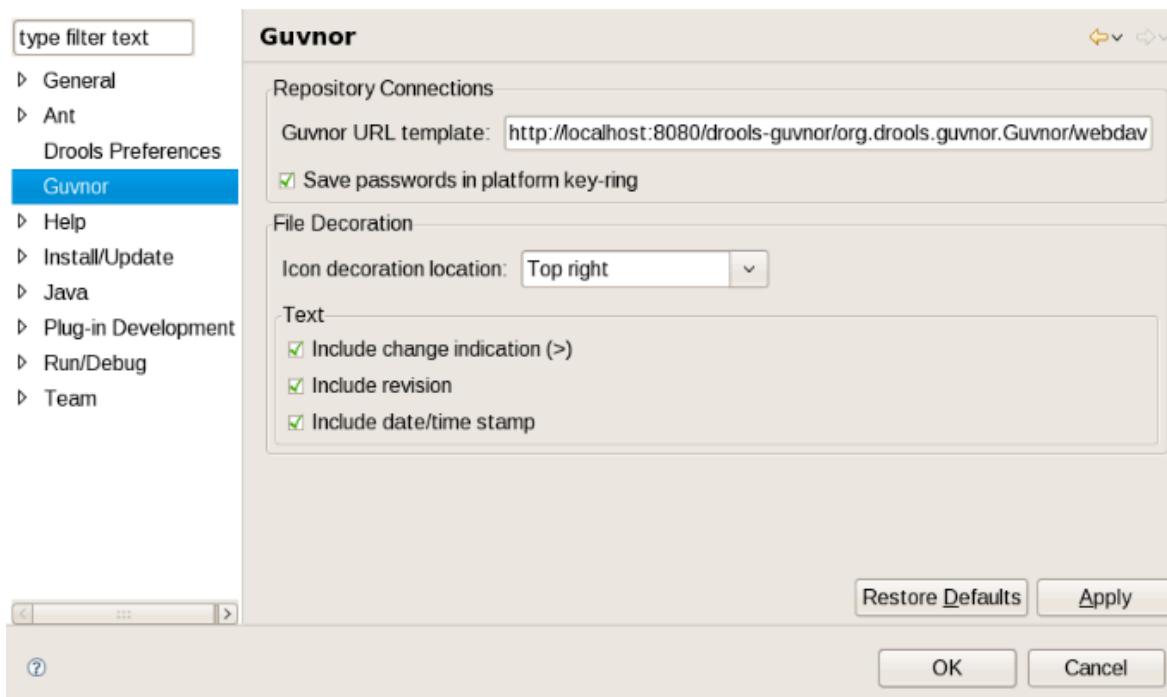


Figure 5.23. Preferences

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.

Appendix A. Example Persistence Manager configurations

Here are several example Persistence Manager configurations for the supported databases. You will, of course, have to update the values in the configurations with those that are correct for your database, such as the JDBC URL and schemaObjectPrefix.

For additional details about Apache Jackrabbit Persistence Managers you should refer to <http://wiki.apache.org/jackrabbit/PersistenceManagerFAQ>

```
<PersistenceManager class=
"org.apache.jackrabbit.core.persistence.bundle.BundleDbPersistenceManager">
  <param name="driver" value="com.mysql.jdbc.Driver"/>
  <param name="url" value="jdbc:mysql://localhost/brms"/>
  <param name="user" value="brms_user"/>
  <param name="password" value="brms_password"/>
  <param name="schema" value="mysql"/>
  <param name="schemaObjectPrefix" value="{wsp.name}_"/>
</PersistenceManager>
```

Example A.1. Generic JDBC Configuration for MySQL using **BundleDbPersistenceManager**

```
<PersistenceManager class=
"org.apache.jackrabbit.core.persistence.bundle.MySqlPersistenceManager">
  <param name="driver" value="com.mysql.jdbc.Driver"/>
  <param name="url" value="jdbc:mysql://localhost:3306/brms"/>
  <param name="user" value="brms_user"/>
  <param name="password" value="brms_password"/>
  <param name="schemaObjectPrefix" value="{wsp.name}_"/>
  <param name="schema" value="mysql"/>
</PersistenceManager>
```

Example A.2. MySQL Configuration using **MySqlPersistenceManager**

```
<PersistenceManager class=
"org.apache.jackrabbit.core.persistence.bundle.OraclePersistenceManager">
  <param name="driver" value="oracle.jdbc.OracleDriver"/>
  <param name="url" value="jdbc:oracle:thin:@localhost:1521:brms" />
  <param name="schema" value="oracle"/>
  <param name="user" value="brms_user" />
  <param name="password" value="brms_password" />
  <param name="schemaObjectPrefix" value="{wsp.name}_" />
</PersistenceManager>
```

Example A.3. Oracle Configuration using **OraclePersistenceManager**

```
<PersistenceManager class="org.apache.jackrabbit.core.persistence.bundle.
PostgreSQLPersistenceManager">
  <param name="driver" value="org.postgresql.Driver"/>
  <param name="url" value="jdbc:postgresql://localhost:5432/brms" />
  <param name="schema" value="postgresql"/>
  <param name="user" value="brms_user" />
  <param name="password" value="brms_password" />
  <param name="schemaObjectPrefix" value="{wsp.name}_" />
</PersistenceManager>
```

Example A.4. PostgreSQL Configuration using **PostgreSQLPersistenceManager**

Appendix B. Revision History

Revision 1.0 Fri 20 Mar 2009

Darrin Mison dmison@redhat.com

Created for BRMS Platform 5.0

