
1. The Guvnor (Business Rule Management System)	1
1.1. Introduction	1
1.1.1. What is a BRMS?	1
1.1.2. Features outline	2
1.2. Administration guide	2
1.2.1. Installation	3
1.2.2. Database configuration	3
1.2.3. Security - Authentication and basic access	5
1.2.4. Fine grained permissions and security	7
1.2.5. Data management	12
1.3. Architecture	14
1.3.1. Building from source	15
1.3.2. Re-usable components	16
1.3.3. Versioning and Storage	17
1.3.4. Contributing	17
1.4. Quick start guide	17
1.4.1. Quick start guide	17
1.4.2. BRMS concepts	21
1.4.3. Creating a business user view	45
1.4.4. The fact model (object model)	45
1.4.5. The business user perspective	48
1.4.6. Advanced config options in a rule package	49
1.4.7. Deployment: Integrating rules with your applications	49
1.4.8. WebDAV and HTTP	57
1.4.9. Eclipse Guvnor integration	58
Index	81

Chapter 1. The Guvnor (Business Rule Management System)

1.1. Introduction

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

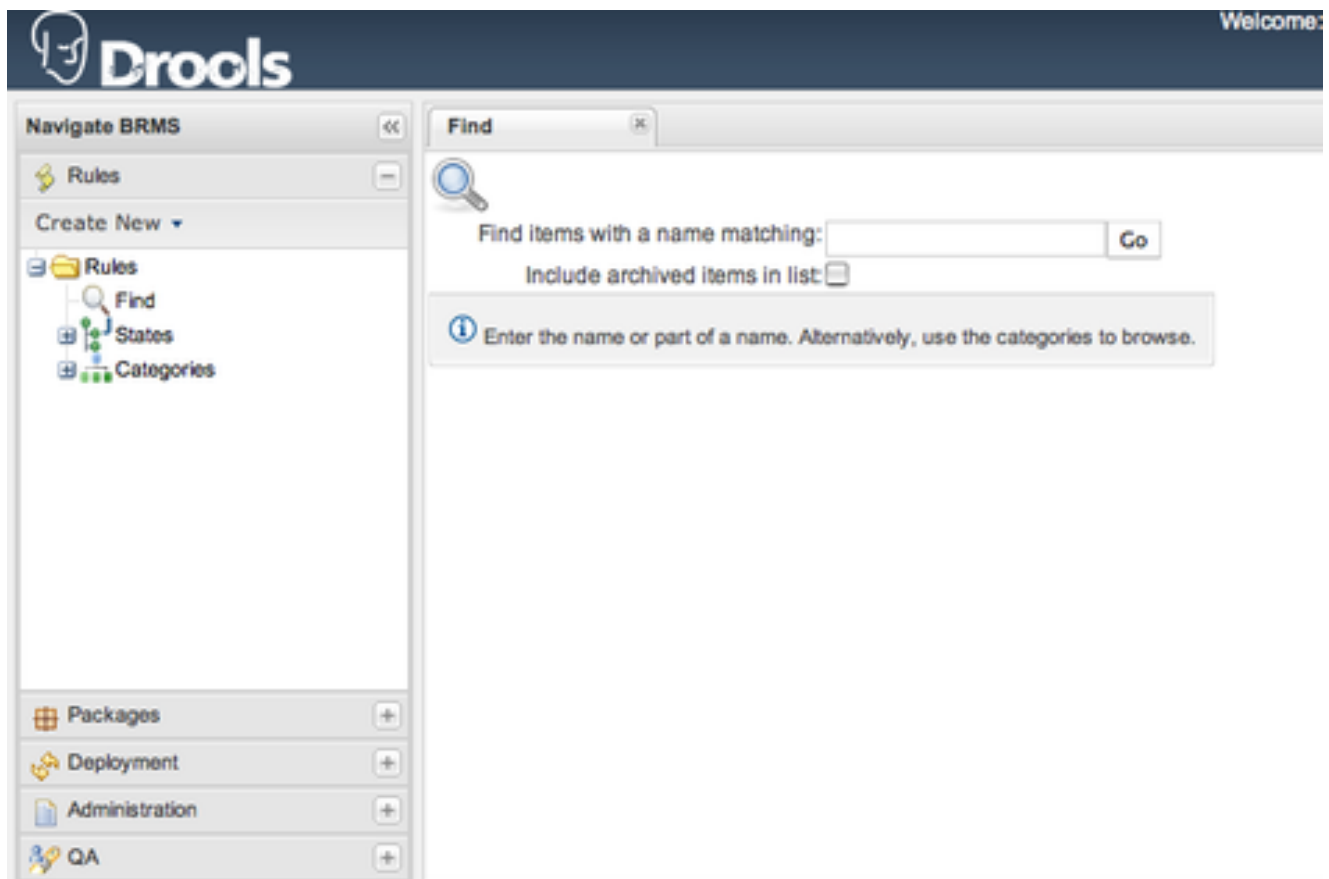


Figure 1.1. The BRMS main screen

1.1.1. What is a BRMS?

BRMS stands for Business Rules Management System.

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.

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

1.1.1.1. When to use Guvnor

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

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

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

1.1.1.1.1. When to not use Guvnor

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

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

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

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

1.1.1.2. Who uses Guvnor

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

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

1.1.2. Features outline

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

1.2. Administration guide

This chapter covers installation and administration issues of the BRMS.

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.

1.2.1. Installation

Installation for most people is very simple. The Guvnor application is deployed as a .war file, which can be deployed in application servers or servlet containers with little or no configuration if you are happy with the defaults.

When you have downloaded the Guvnor distribution (which you can get from <http://www.jboss.org/drools>), you will find the drools-jbrms.war file in the zip file. Copy the WAR file into the deployment directory of your app server, and then start your app server. If you need to customize some settings, you can first "explode" (unzip) the war file, and change any configuration settings, and then either zip it up, or deploy it "exploded". (Note in the JBoss Application Server you will need to make sure the exploded folder name ends in ".war")

Once the drools-guvnor.war has been placed in the deployment directory, and the application server started, you should navigate to <http://localhost/drools-jbrms> and check that Guvnor appears. (Obviously substitute the URL for what your application server is configured to).

Once that shows up, you are deployed and ready to go !

1.2.1.1. Supported and recommended platforms

Guvnor is capable of running in any application server that supports Java SE5 (JEE 5 is not required) - this includes servlet containers like tomcat.

It is actively tested/developed on JBoss app server 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 - many in the community have been able to make it run on various platforms).

You can of course download these app servers 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.

1.2.2. Database configuration

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

1.2.2.1. Changing the location of the data store

When you run Guvnor for the first time (starting up the app server), it will create a database in the [app server directory]/bin/ directory (assuming you used one of the JBoss platforms). There is a repository.xml file, and a repository directory that are automatically created.

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

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

```
<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>
```

Find the component with a name of "repositoryConfiguration" and the property with the name of "homeDirectory".

If you un comment this (as in the example above it is commented out), you can set whatever path you need for the repository data to be stored in. You can also use this to move the repository around. In that case, when you have set the location in the components.xml you can simply move the repository.xml AND the repository directory to the new location that you set in the components.xml.

If there is no repository at the location specified (or in the default location) then Guvnor will create a new empty one.

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

1.2.2.2. Configuring Guvnor to use an external RDBMS

In some cases it may be a requirement that you use an external RDBMS, such as Oracle, MySQL, or Microsoft SQL Server as the data store - this is permitted. In this case, the easiest thing to do is to start up Guvnor with defaults (or with a suitable home directory as specified above) to let it generate the default repository.xml scaffolding.

Locate the repository.xml file that was generated, and open it - it will be annotated with comments describing many of the different options. From here on, you will need to know a little about Jackrabbit Persistence managers: <http://jackrabbit.apache.org/doc/config.html>

There are a few persistence managers, some are database specific (eg Oracle). There is a SimpleDBPersistenceManager which works with any database that supports JDBC - you also

specify the database type, so it uses the specific DDL to create the table structure (all major databases are supported).

Guvnor will create the tables the first time it is started up if it is running against a fresh (empty) RDBMS - so its important to note that the user credentials supplied have permissions to create tables (at least initially, on first run, after that they could be locked down).

1.2.2.3. Searching and indexing, Version storage

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

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

1.2.3. Security - Authentication and basic access

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

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

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

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

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

```
<!-- SECURITY CONFIGURATION -->

<!-- default (will take any username, useful if you want to keep track of
users but not authenticate -->
<security:identity
  authenticate-method="#{defaultAuthenticator.authenticate}"/>
```

```
<!-- NO authentication. This will bypass the login screen when you hit the
app. Everyone is "guest" -->
<!-- <security:identity
authenticate-method="#{nilAuthenticator.authenticate}"/> -->
```

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

1.2.3.1. Using your containers security and LDAP

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

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

This will use the "other" jaas config in JBoss AS. If you look in [jboss install dir]/server/default/conf you will see a login-config.xml file. This file contains various configs. If you use "other" like the one above, then it will look for users.properties and roles.properties in the conf directory for usernames and passwords to authenticate against (this is fine for a fixed small number of users).

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

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

      <module-option
name="rolesCtxDN">cn=Users,dc=jboss,dc=org</module-option>
```

```
        <module-option
name="roleFilter">({sAMAccountName={0}})</module-option>
        <module-option name="roleAttributeID">memberOf</module-option>
        <module-option name="roleAttributeIsDN">true</module-option>
        <module-option name="roleNameAttributeID">cn</module-option>

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

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

Similar configuration examples can be found for other directory services.

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

1.2.4. Fine grained permissions and security

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

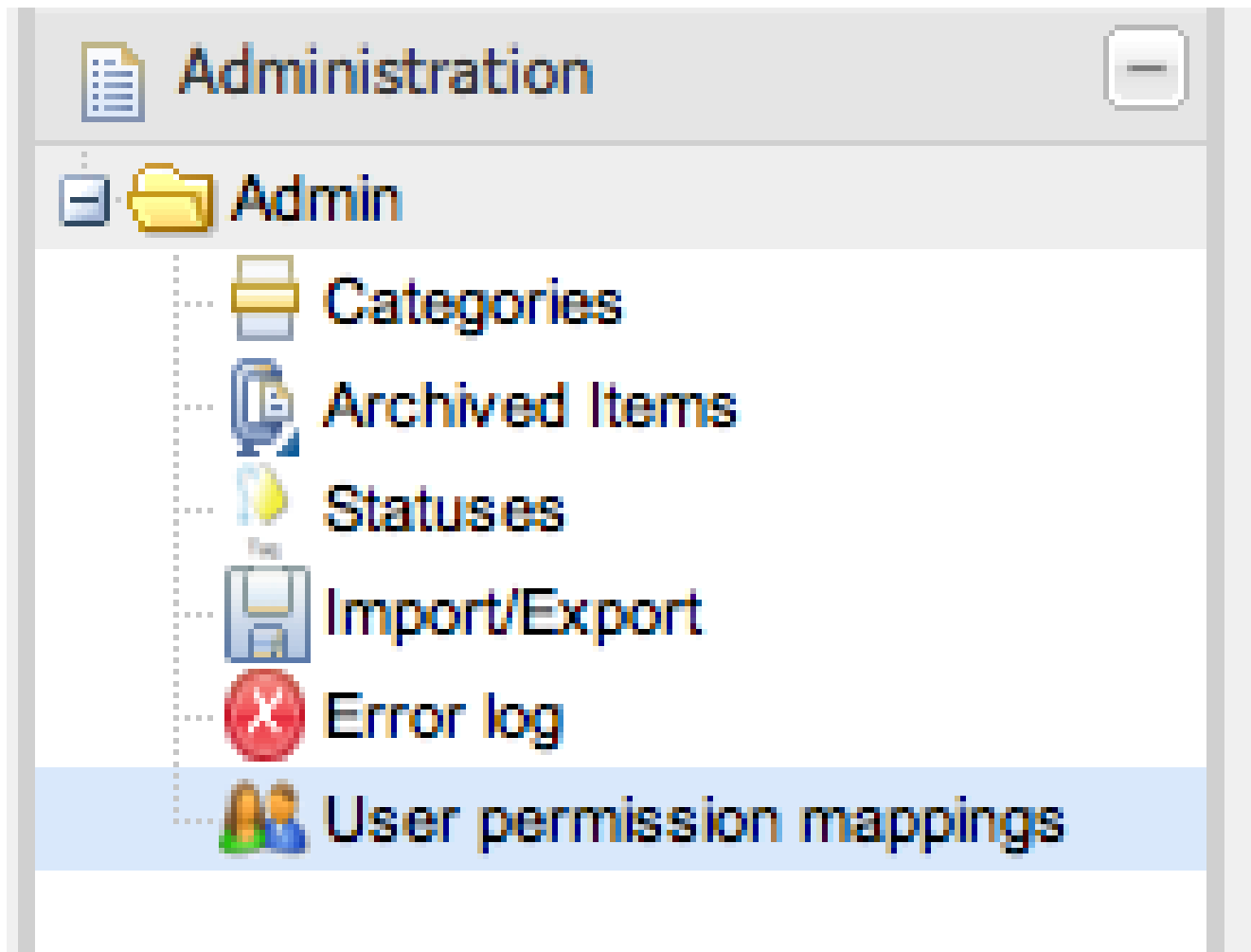


Figure 1.2. Administer user permissions

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

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


Find ✕ User Permission ✕			
Currently configured users: Reload Create new user mapping Delete selected user			
User name ▲	Administrator	Has package permissi...	Has category permissi...
⊟ Administrator: (None) (2 Items)			
NewUser		Yes	
woozle2			Yes
⊟ Administrator: Yes (1 Item)			
woozle1	Yes		

Figure 1.3. User listing

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

There are really 2 system wide roles: Users who are Administrators and users who are not. Easy ! Administrators can see and do anything. Out of the box, the permission system is turned off, and every user is an administrator (this is pretty much how things used to work). There is also a system setting in components.xml that can turn the permissions system on and off (so people can manually override if needs be). A administrator can also give other users admin rights, regardless of their roles in the external directory service.

Edit user: NewUser
Yes
✕



Edit user: NewUser

Users are authenticated by a directory service, here you can define Guvnor specific permissions as needed.

[analyst] for:

category=HR ☐

[package.admin] for:

package=com.billasurf.finance ☐




Figure 1.4. Editing

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

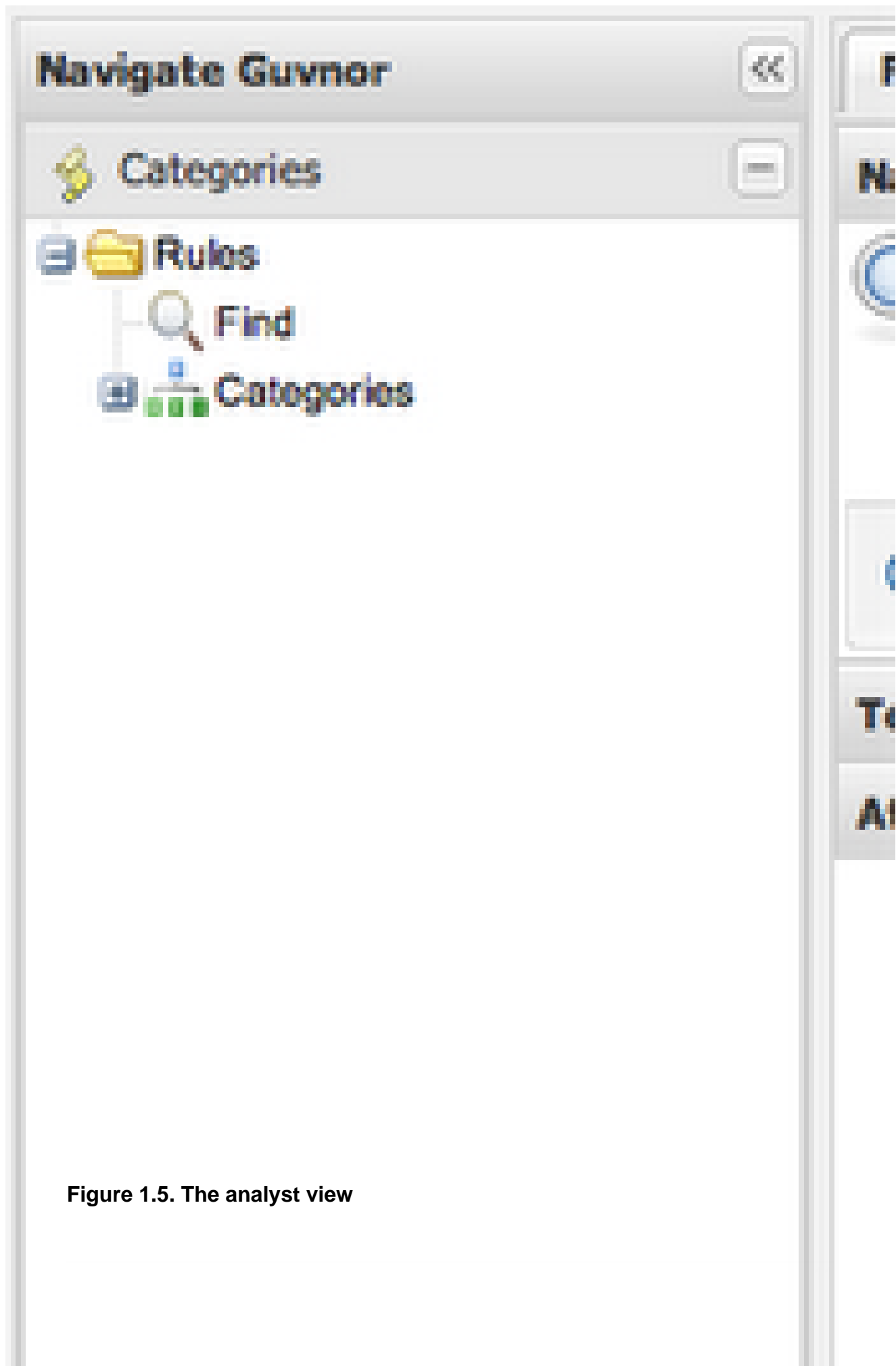


Figure 1.5. The analyst view

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

1.2.4.1. Enabling fine grained authorization

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

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

1.2.5. Data management

1.2.5.1. Backups

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

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

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

1.2.5.2. Asset list customization

In a few places in Guvnor 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. eg you could add in getCreator, or getExternalSource as extra fields if you were using them.

1.2.5.3. Customised 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 will need to "explode" the war file for Guvnor, and locate the selectors.properties file (note you can also put your own selectors.properties file in the system classpath if you like). In this file, you will find details on how you can configure a custom selector. The options are to use a drl file, or the name of a class that you have written (and which is available

on the classpath). Classes must implement the AssetSelector interface. DRL files can also be used (there is an example one in the selectors.properties file). Each selector you configure has a unique name in this properties file - and this is the name that you can use when building packages.

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

Everyone loves having their own logo on screen - this is to ensure that the people using the application don't forget who they work for or what product they are using for more than a nanosecond (the consequences of them forgetting are too terrible to contemplate).

To achieve, this, you can "explode" the deployment war file, and locate the JBRMS.html file.

```
<html>
<head>
  <meta name='gwt:module' content='org.drools.guvnor.Guvnor'>
  <link rel='stylesheet' href='JBRMS.css'>
  <title>JBoss Business Rules Management System</title>
  <link rel="shortcut icon" href="images/drools.gif" type="image/gif">
  <link rel="icon" href="images/drools.gif" type="image/gif">
</head>
<body>
  <div class="headerBarblue"></div>
<!-- This script is the bootstrap stuff that simply must be there; it is
sent down uncompressed -->
  <script language='javascript' src='gwt.js'></script>
  <iframe id='__gwt_historyFrame'
style='width:0;height:0;border:0'></iframe>
</body>
</html>
```

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

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

1.2.5.5. Import and Export

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

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

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

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

1.3. Architecture

This section covers the technical aspects of Guvnor (previously known as the Business Rules Management System or BRMS), it is not necessary to use this if you are integrating or an end user of the BRMS application. However, Drools is open source, so build instructions form part of the manual.

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

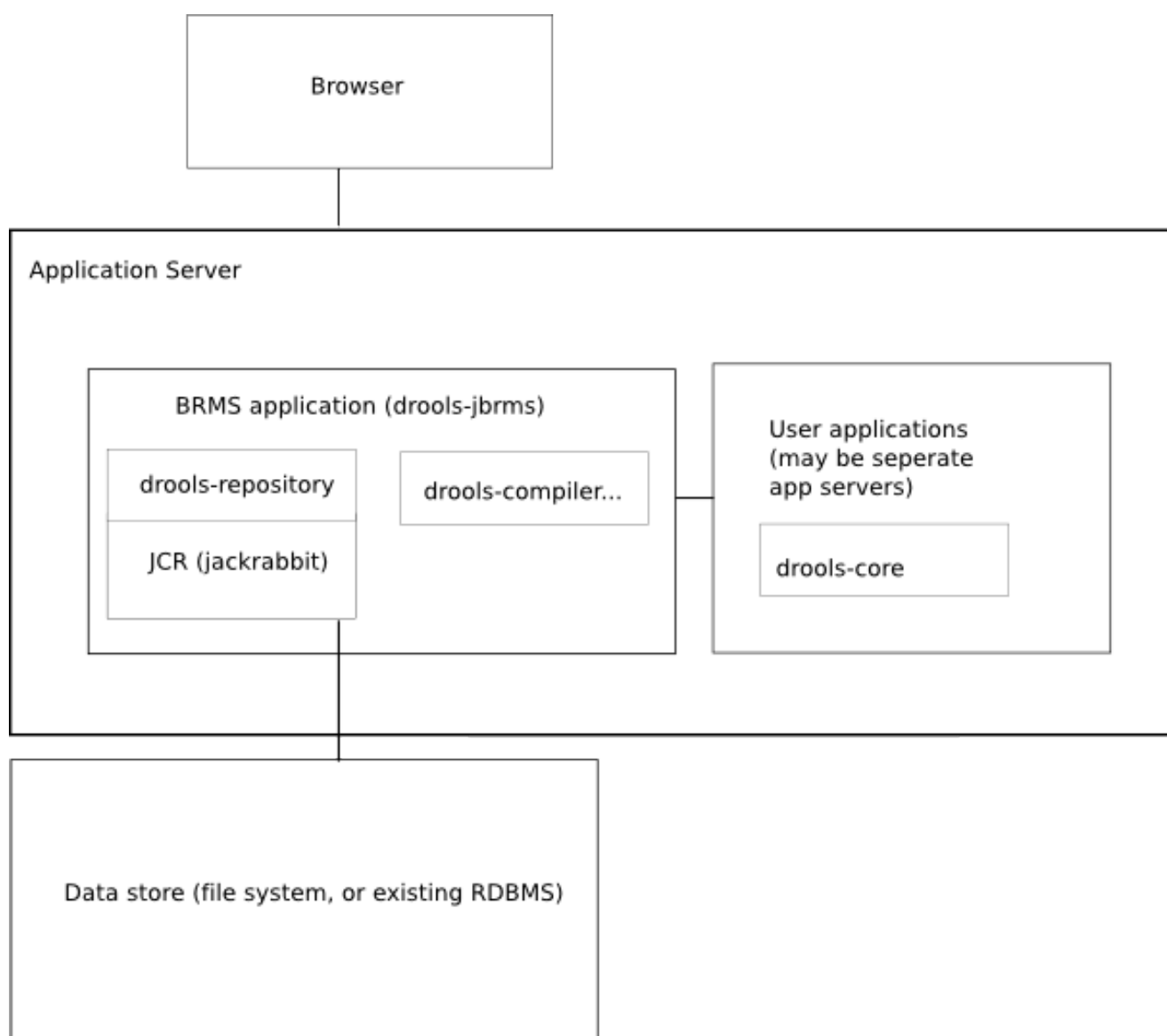


Figure 1.6. Architectural diagram

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

The BRMS is deployed as a war, which provides user interfaces over the web, and provides binary packages via URLs (or files). It utilized 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.

1.3.1. Building from source

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

1.3.1.1. Modules

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

1.3.1.2. Working with Maven 2

Maven 2 is used as the build system. To get started, the WHOLE of the source tree for JBoss Rules will need to be checked out. This includes the other modules, and the top level lib and repository directories (which are needed by the build); as the BRMS build is part of the main drools build.

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

When wishing to build the BRMS, go into the drools-jbrms directory, and run `mvn package`. This will run the tests, and then build a deployable war. The only thing this will not do is rebuild the GWT front end (see the next section for details on that). Once the war file is in the target directory, the BRMS is ready to go.

1.3.1.3. Working with GWT

The GUI widgets for the web front end are developed with GWT (Google Web Toolkit). If there is a need to make changes to or build the GUI, GWT will need to be downloaded separately. Once GWT is downloaded, the `build.properties` file in the drools-jbrms directory to point to where GWT is installed. Once this is completed, the ant tasks can be used to build the GWT components, as well as launch GWT in debug/hosted mode if it is desired. If running the build, it will update the webapp directory in the project with the new *compiled* artifacts (GWT does not use JSP, only html and javascript at runtime).

1.3.1.4. Debugging, Editing and running with Eclipse

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

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

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

Normally

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

1.3.2. 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 (drools-repository and JCR) as well as the compiler specifics to deal with rules.

The main interface is `RepositoryService`, which is implemented in `ServiceImplementation`. The GWT ajax front end talks to this interface (via the asynchrony callback mechanism that GWT uses). The seam configuration file is `components.xml` (consult Seam documentation, and the `components.xml` file for details).

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

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

Normally Guvnor is intended to be deployed as its own war, however it can be combined with another application (with some care), but it is easier to keep it as a separate war. Deploying Guvnor by itself will also make it easier to upgrade to newer releases as they come out (and is recommended).

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

1.3.3. Versioning and Storage

The User Guide, Admin Section 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. In a nutshell, a package is a *folder* and each asset is a file: an asset can either be textual or have a binary attachment.

1.3.4. Contributing

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

1.4. Quick start guide

1.4.1. Quick start guide

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

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

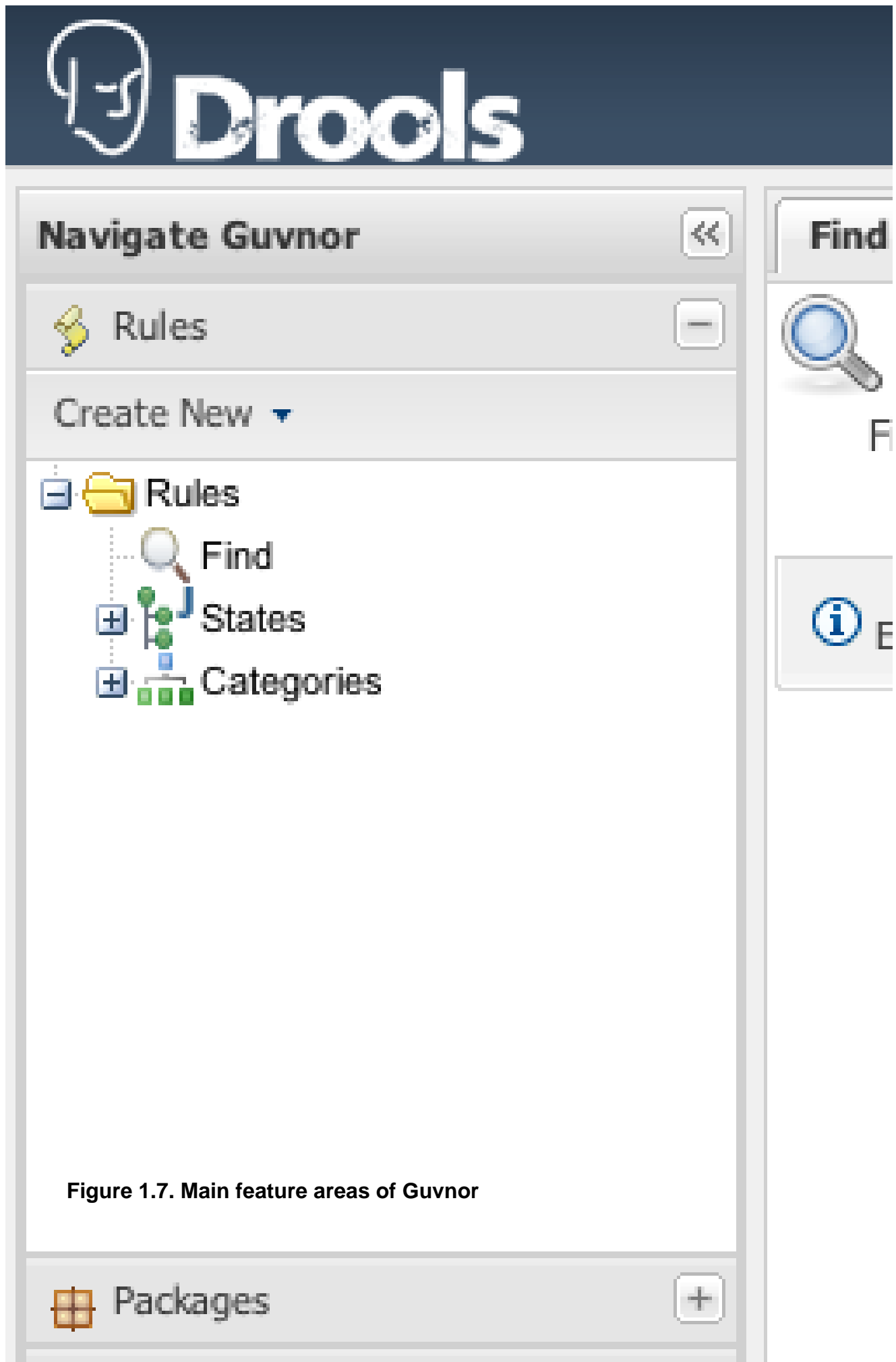


Figure 1.7. Main feature areas of Guvnor

The above picture shows the main feature areas of Guvnor.

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

1.4.1.1. Supported browser platforms

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

1.4.1.2. BRMS or Guvnor?

In previous versions of Drools "BRMS" was often used to refer to the web interface to the drools management features. Nowadays we use BRMS to refer to the "whole package" - the runtime, the web tools and so on - but in some cases you can read "BRMS" as meaning the Guvnor web console and associated tools.

1.4.1.3. Initial configuration

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

- Once deployed, go to "<http://<your server>/drools-guvnor/>" (This will show the initial info screen - or login screen depending on the configuration).
- If it is a brand new repository, you will want to go to "Admin", and choose "Manage Categories"
(Add a few categories of your choosing, categories are only for classification, not for execution or anything else.)
- Rules need a fact model (object model) to work off, so next you will want to go to the Package management feature. From here you can click on the icon to create a new package (give it a meaningful name, with no spaces).
- To upload a model, use a jar which has the fact model (API) that you will be using in your rules and your code (go and make one now if you need to !). When you are in the model editor screen, you can upload a jar file, choose the package name from the list that you created in the previous step.

- Now edit your package configuration (you just created) to import the fact types you just uploaded (add import statements), and save the changes.
- At this point, the package is configured and ready to go (you generally won't have to go through that step very often).

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

1.4.1.4. 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)

1.4.1.5. 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).

1.4.1.6. 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 effect 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).

1.4.2. BRMS concepts

1.4.2.1. 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" (ie 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.

1.4.2.2. Categorisation

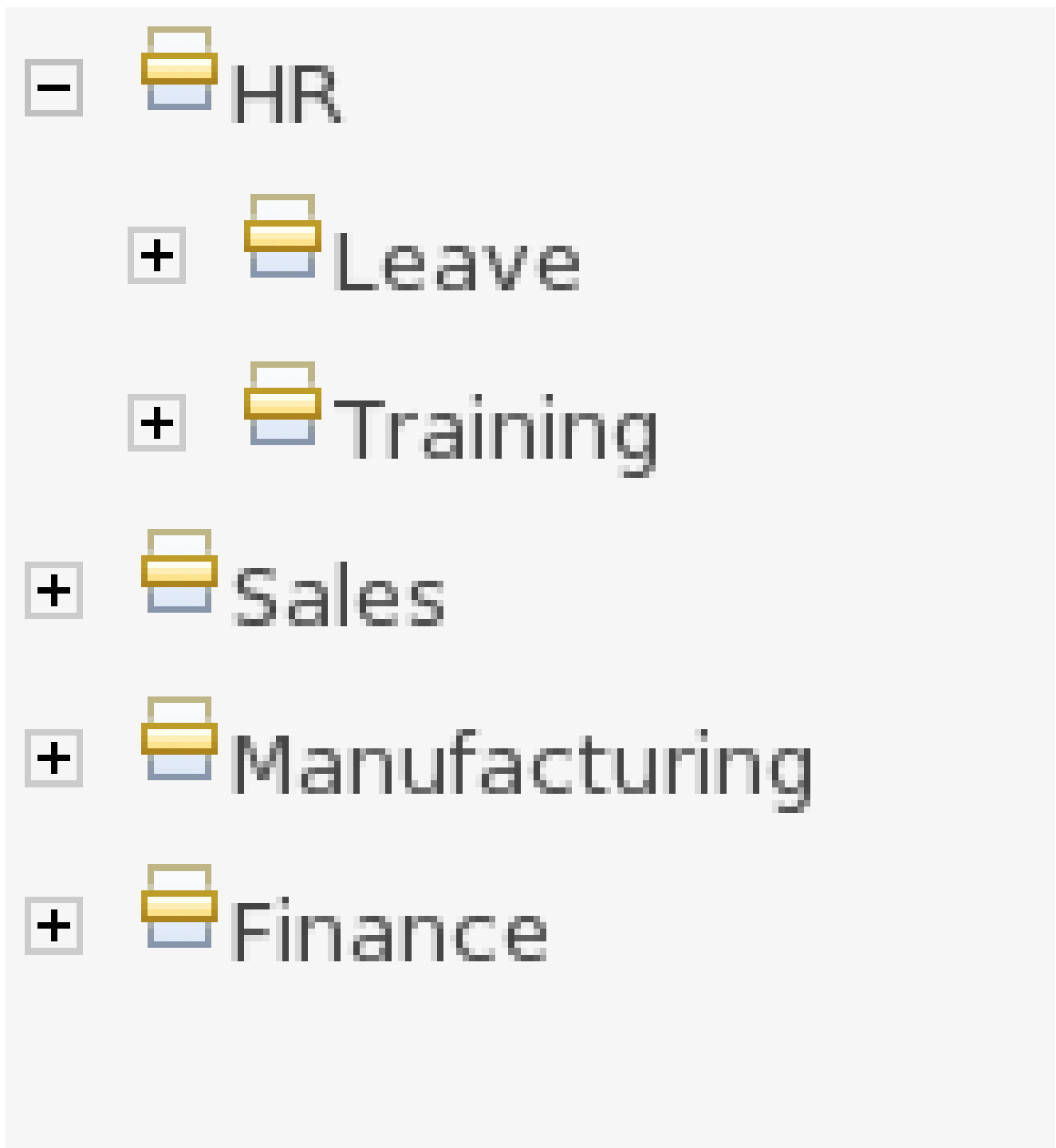


Figure 1.8. Categories

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 1.9. Assets can have multiple categories

The view above shows the category editor/viewer that is seen when you open an asset. In this example you can see the asset belongs to 2 categories, with a "+" button to add additional items (use the trash can item to remove them). This means that when either category is used to show a list of assets, you will see that asset.

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

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

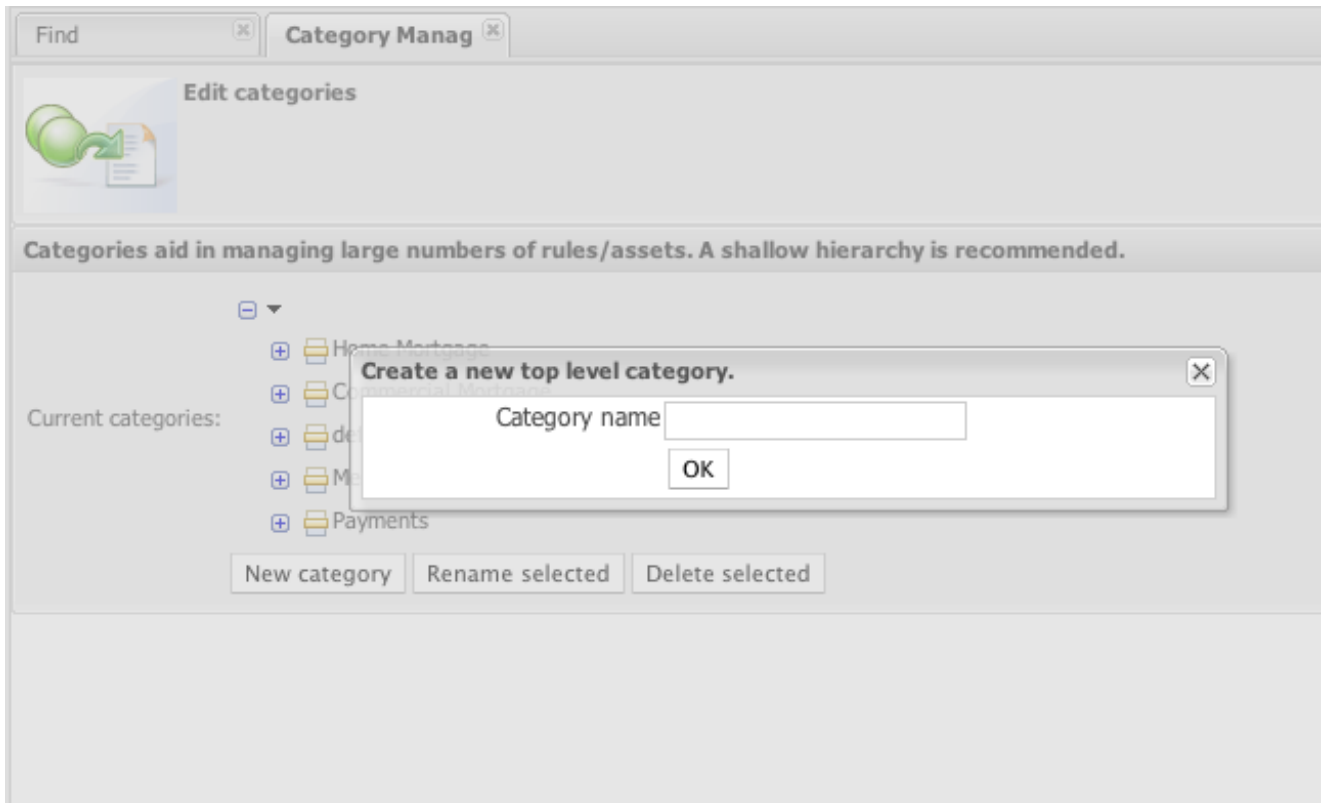


Figure 1.10. Creating categories

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.

1.4.2.3. The asset editor

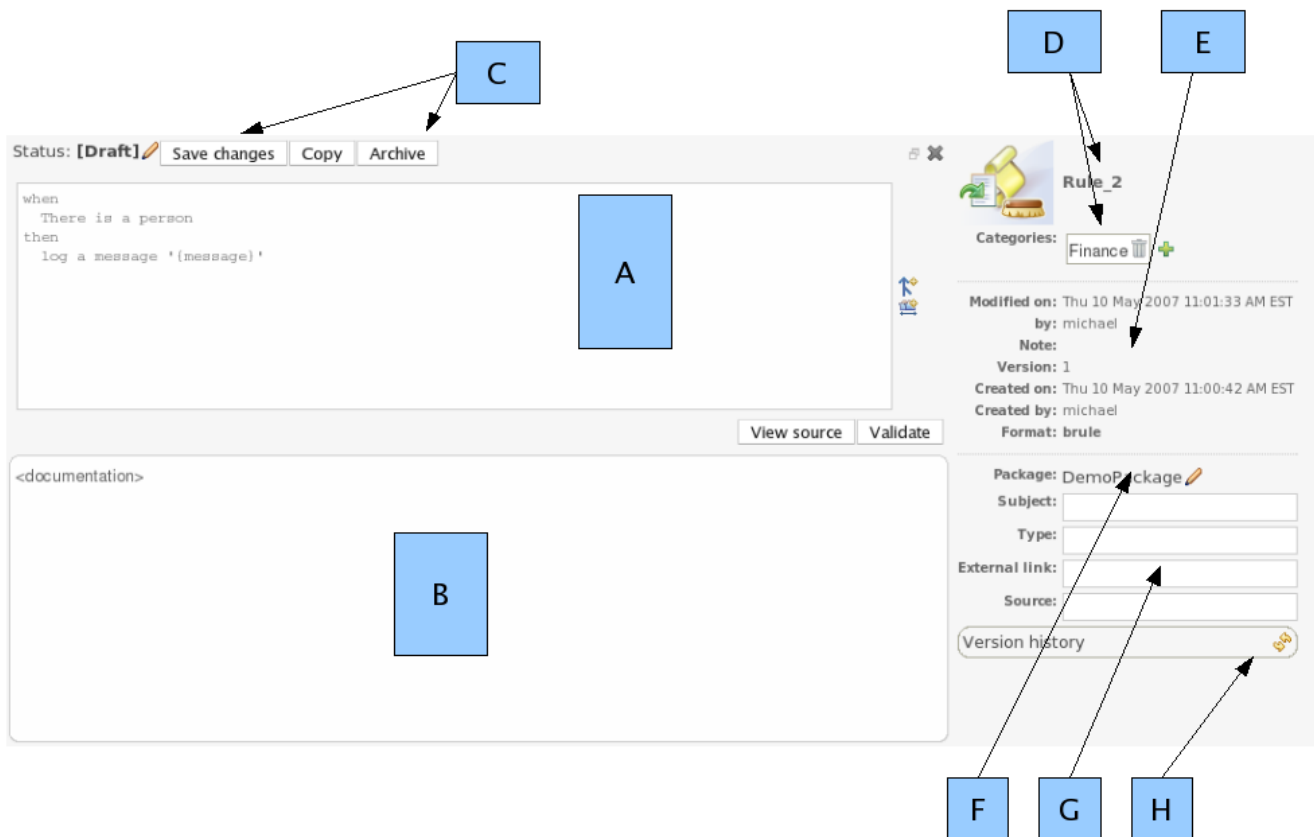


Figure 1.11. The Asset editor view

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 (ie 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 belong to (you can also change it from here).

- G

This is some more (optional) meta data (taken from the Dublin Core meta data standard)

- H

This will show the version history list when requested.

1.4.2.4. Rule authoring

The BRMS supports a (growing) list of 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.

1.4.2.4.1. 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.

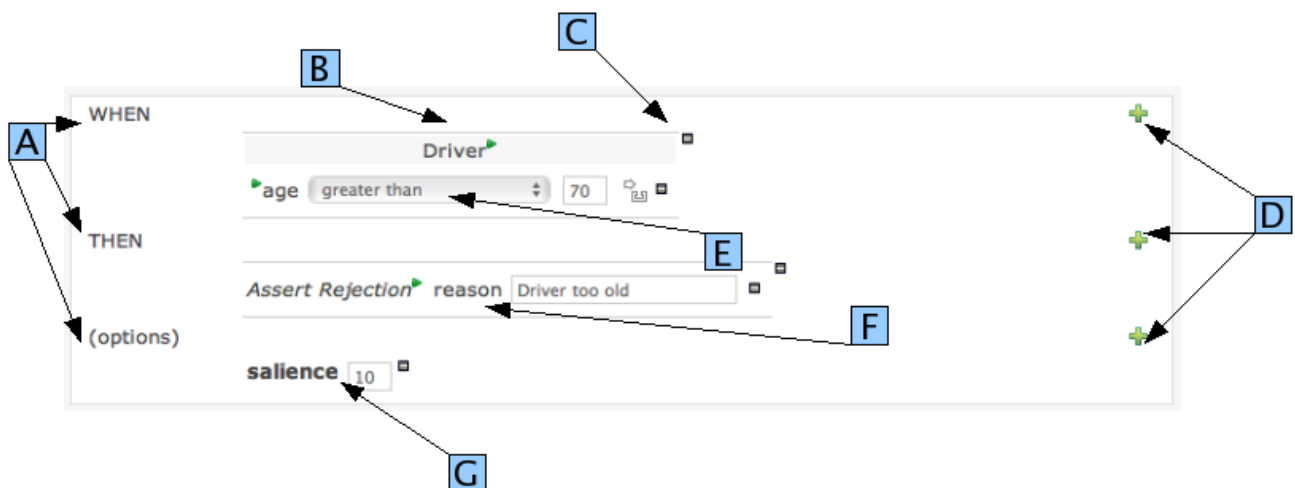


Figure 1.12. The guided BRL editor

The above diagram shows the editor in action. The following description apply to the letter 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 (eg 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 (eg 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.

1.4.2.4.1.1. User driven drop down lists

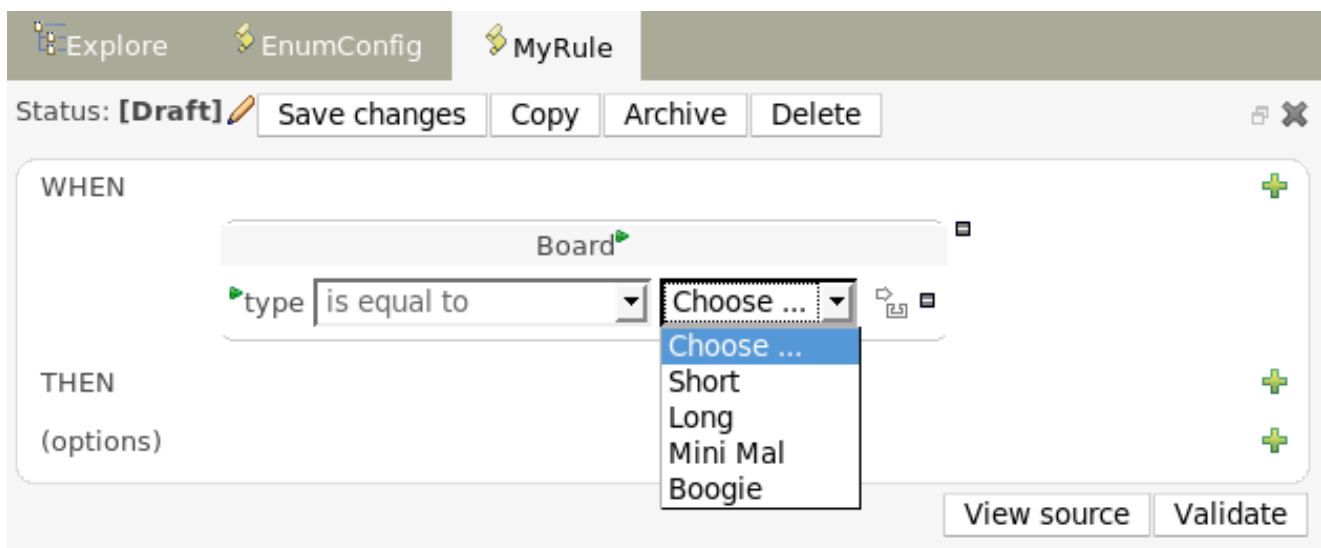


Figure 1.13. Data enumeration showing as a drop down list

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.

1.4.2.4.1.2. Augmenting with DSL sentences

If the package the rule is part of has a dsl configuration, when you add conditions or actions, then it will provide a list of "DSL Sentences" which you can choose from - when you choose one,

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

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

WHEN

A template captures values in a form style of input

THEN

Action sentence template

(options)

Figure 1.14. DSL in guided editor

1.4.2.4.1.3. A more complex example:

WHEN

Person

age is less than 2

age < (age * 2) && name != "boo"

age is less than 32 * 2

Board

There is no Any of: type is equal to 42

name is equal to myname

THEN

Assert Person age 42

(options)

Add a condition to the rule... x

Fact Choose fact type...

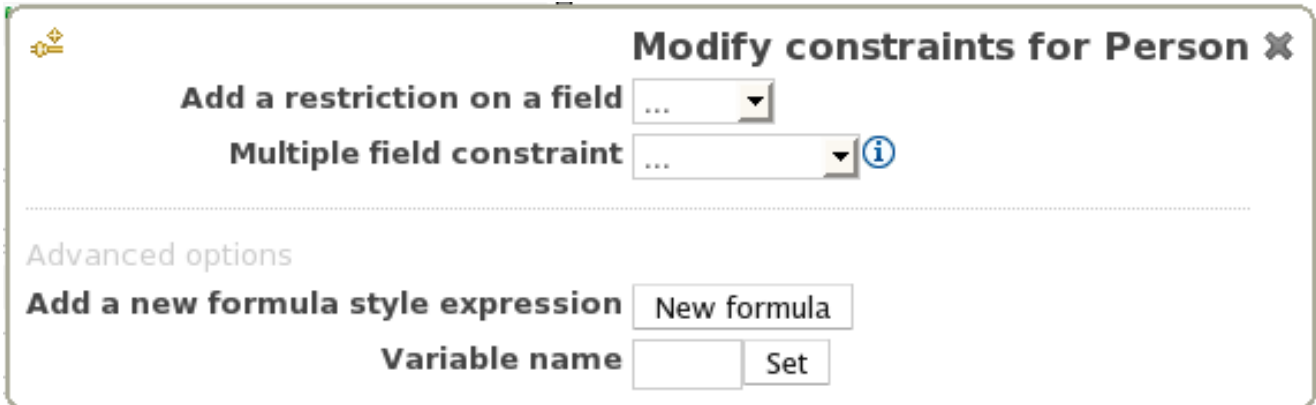
Condition type Choose condition type...

Figure 1.15. 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).



Modify constraints for Person ✕

Add a restriction on a field ...

Multiple field constraint ... ⓘ

Advanced options

Add a new formula style expression

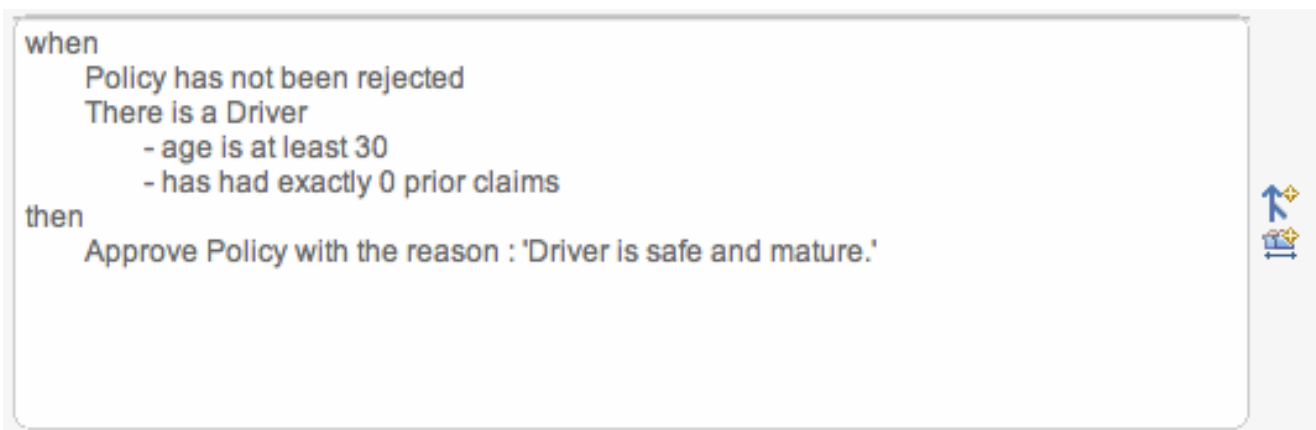
Variable name

Figure 1.16. 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).

1.4.2.4.2. DSL rules

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



```
when
  Policy has not been rejected
  There is a Driver
    - age is at least 30
    - has had exactly 0 prior claims
then
  Approve Policy with the reason : 'Driver is safe and mature.'
```

Figure 1.17. 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).

1.4.2.4.3. 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).



dt

Upload new version:

Download current version:

This is a decision table in a spreadsheet (XLS). Typically they contain many rules in one sheet.

Figure 1.18. 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).

1.4.2.4.4. 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 1.19. Decision table

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

Save changes Copy Archive

Decision table

Attribute columns

Condition columns

- Advertiser type
- age is at least
- Postcode greater than
- Postcode less than

Action columns

- Set the value score
- Set the reason

(options)

Group by column: Advertiser type Apply

Modify...

Description	Advertiser type	age is at least	Postcode gre...	Postcode les...	Set the value...	Set the rea...
-------------	-----------------	-----------------	-----------------	-----------------	------------------	----------------

Advertiser type: Agency (3 items)

Figure 1.20. 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.

1.4.2.4.5. Rule flows

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

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

1.4.2.4.6. 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 few "technical rules" - you can mix and match all the rule types together of course.

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

Figure 1.21. DRL technical rule

1.4.2.4.7. 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 1.22. Function

1.4.2.4.8. Data enumerations (drop down list configurations)

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

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

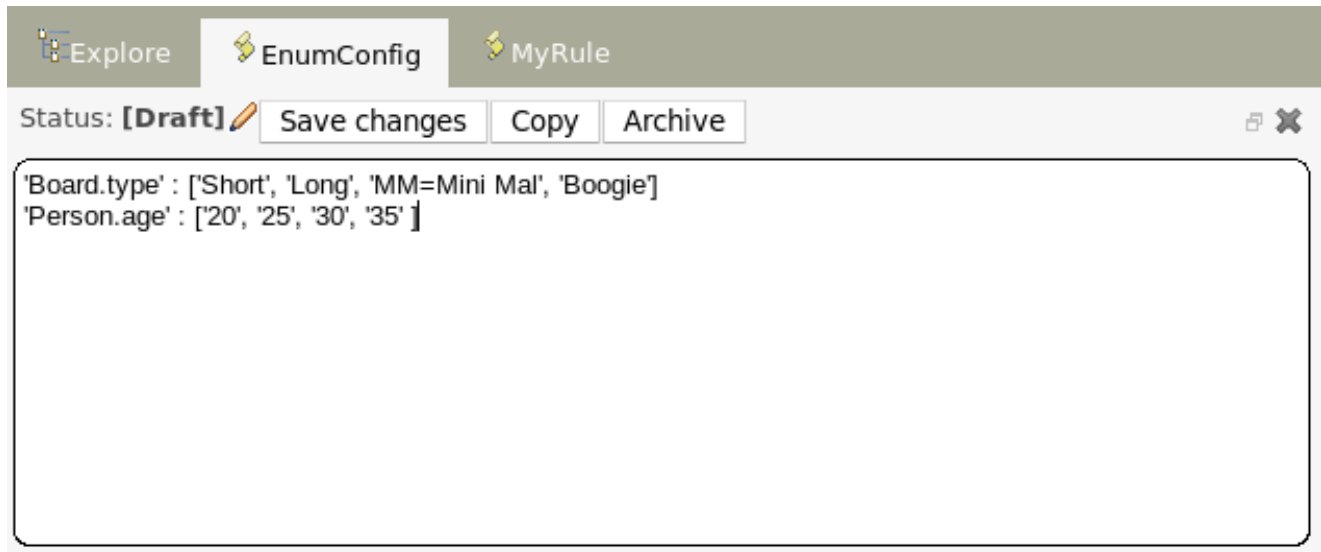


Figure 1.23. Data enumeration

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.

1.4.2.4.9. 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".

1.4.2.5. 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).

1.4.2.6. Status management

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

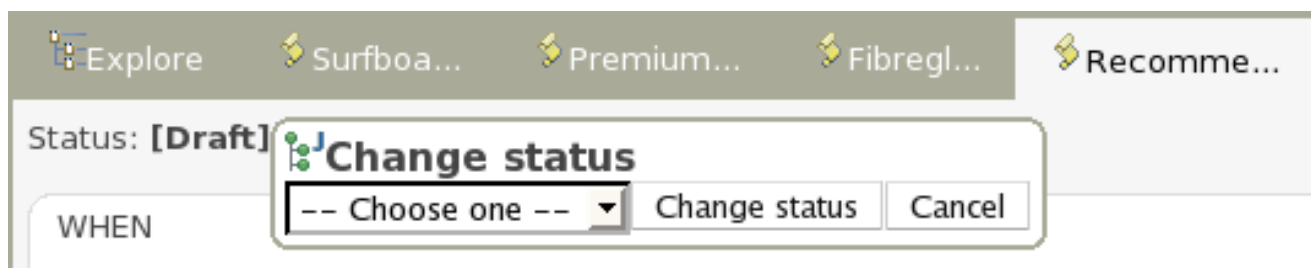


Figure 1.24. Asset status

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

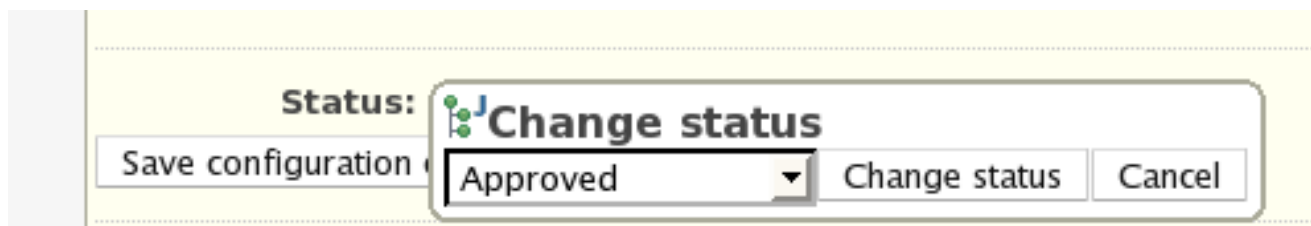


Figure 1.25. Asset status

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).

1.4.2.7. 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.

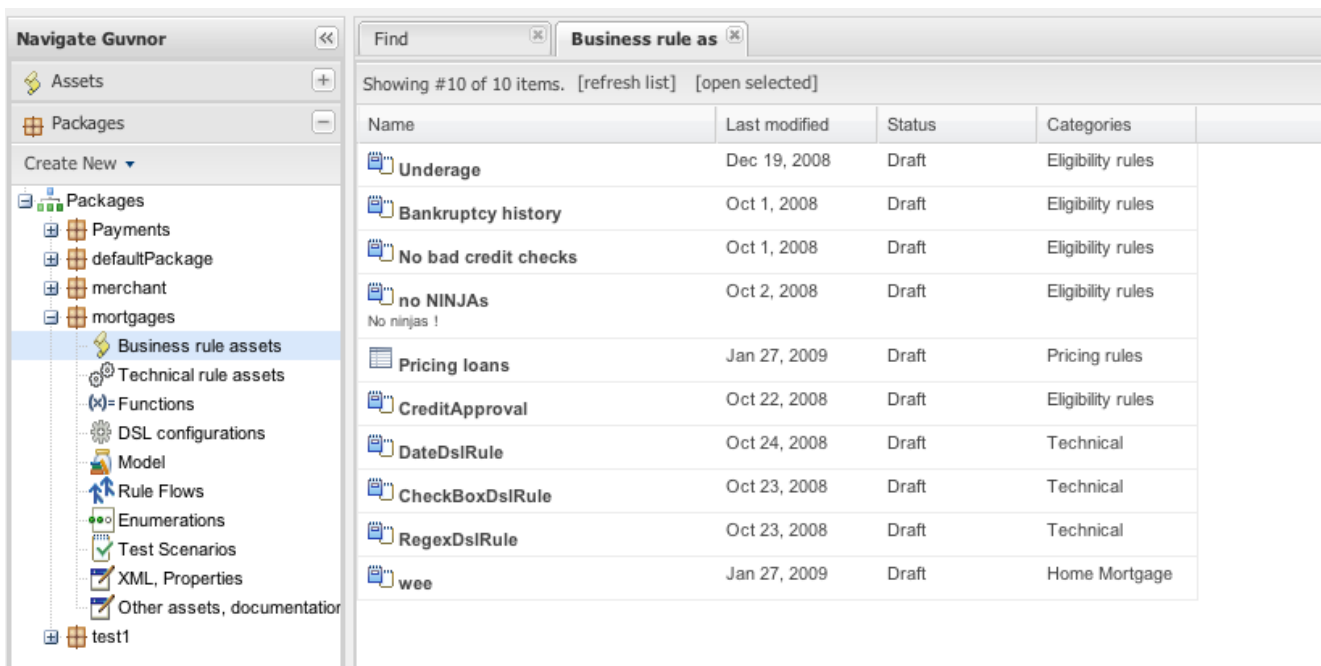


Figure 1.26. The package explorer

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 (eg 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.

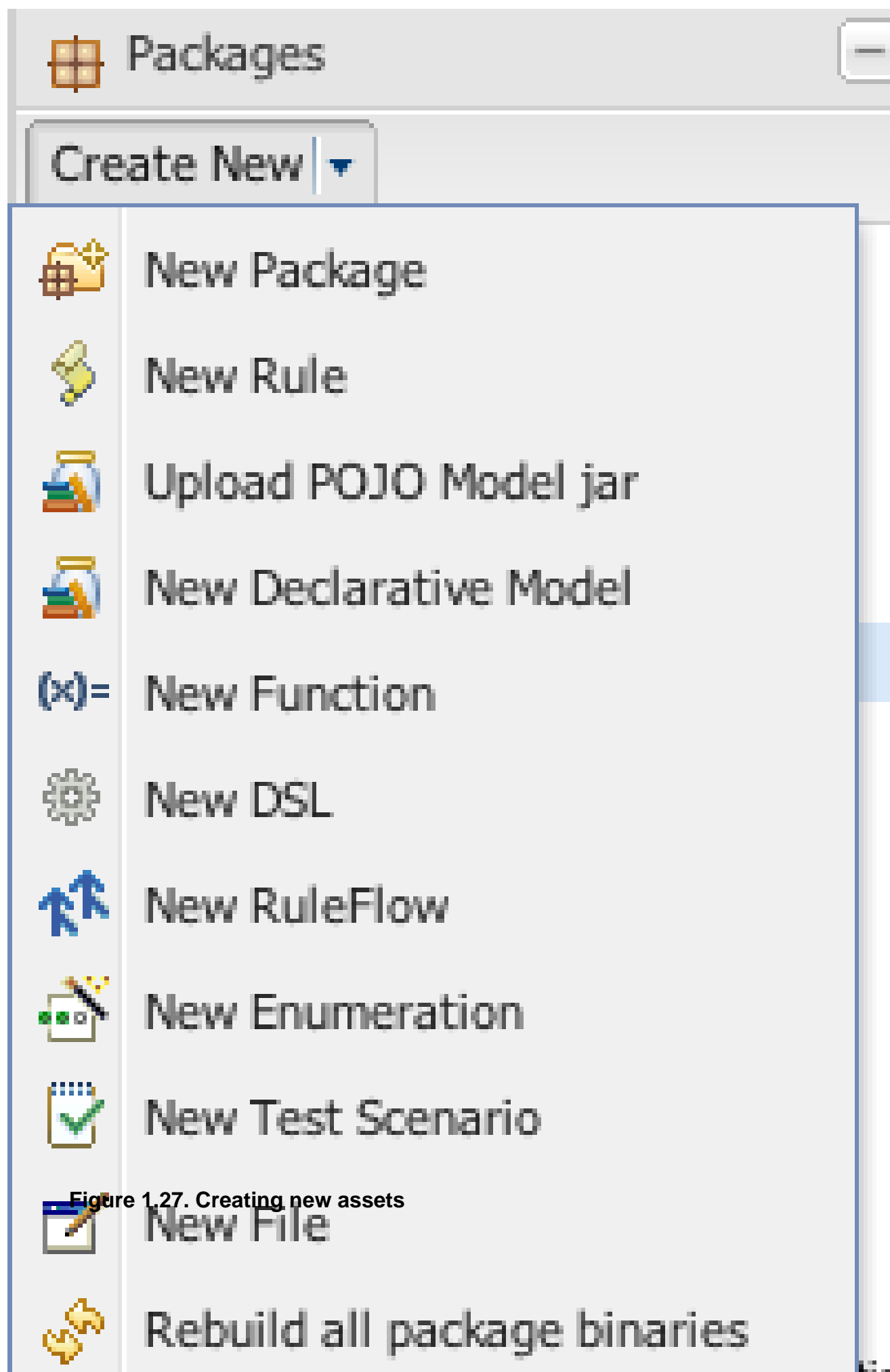


Figure 1.27. Creating new assets

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.

The screenshot displays the 'Package Configuration' window for a package named 'Payments'. At the top, there is a package icon (a gift box) and buttons for 'Copy', 'Rename', and 'Archive'. The main section is titled 'Configuration' and contains several fields and lists:

- Imported types:** A list box containing 'com.lb.gpps.drools.PaymentData', 'com.lb.gpps.drools.ProcessorBlackList', and 'com.lb.gpps.drools.ResultMerchant'. To the right of this list are two trash icons and a '+' icon.
- Globals:** A list box containing 'Advanced view'. To the right of this list are two trash icons and a '+' icon.
- Description:** A text input field.
- Category Rules:** A text input field with a '+' icon and an information icon (i).
- Save and validate configuration:** A button.

Below the 'Configuration' section is the 'Build and validate' section, which includes:

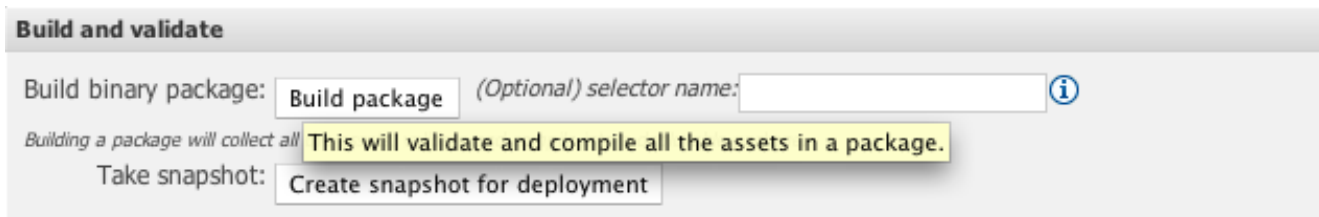
- Build binary package:** A button labeled 'Build package' and a text input field for '(Optional) selector name:' with an information icon (i).
- Take snapshot:** A button labeled 'Create snapshot for deployment'.

The bottom section is titled 'Information and important URLs' and contains:

- Last modified:** Jan 14, 2009 10:10:10 PM
- Last contributor:** god
- Date created:** Jan 14, 2009 10:09:42 PM
- Show package source:** A button labeled 'Show package source'.
- URL for package source:** <http://localhost:8888/org.drools.guvnor.Guvnor/package/Payments/LATEST.drl> with an information icon (i).
- URL for package binary:** <http://localhost:8888/org.drools.guvnor.Guvnor/package/Payments/LATEST> with an information icon (i).
- URL for running tests:** <http://localhost:8888/org.drools.guvnor.Guvnor/package/Payments/LATEST/SCENARIOS> with an information icon (i).
- Status:** A small orange icon.

Figure 1.28. 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 globals 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.



Build and validate

Build binary package: (Optional) selector name: ⓘ

Building a package will collect all **This will validate and compile all the assets in a package.**

Take snapshot:

Figure 1.29. 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).

1.4.2.7.1. 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.

1.4.2.8. 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 history			
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 1.30. 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.

1.4.2.9. 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.

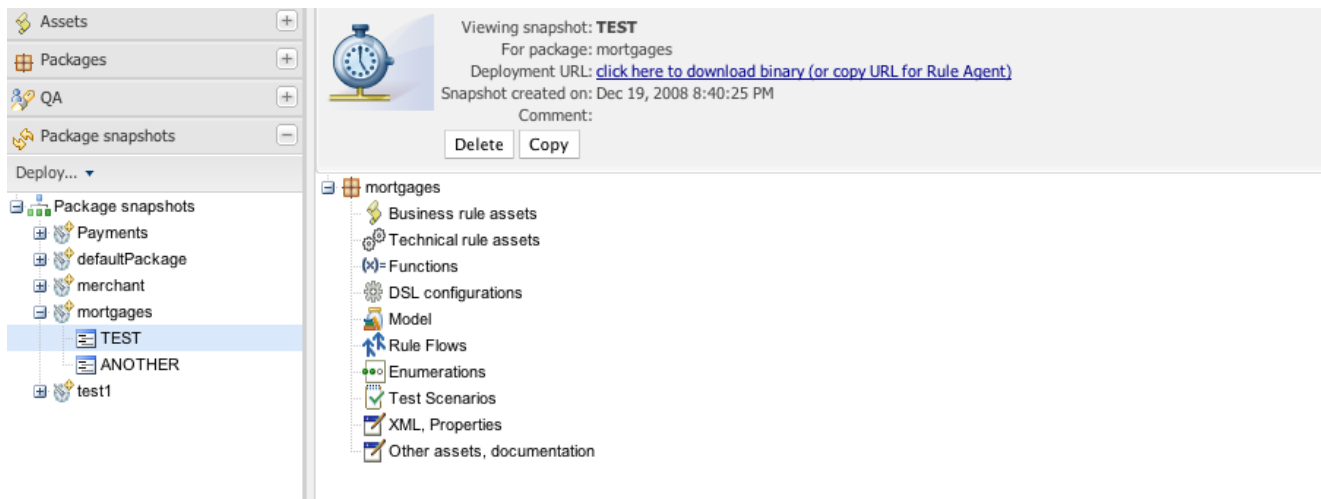


Figure 1.31. Deployment snapshots

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.

1.4.2.10. 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.

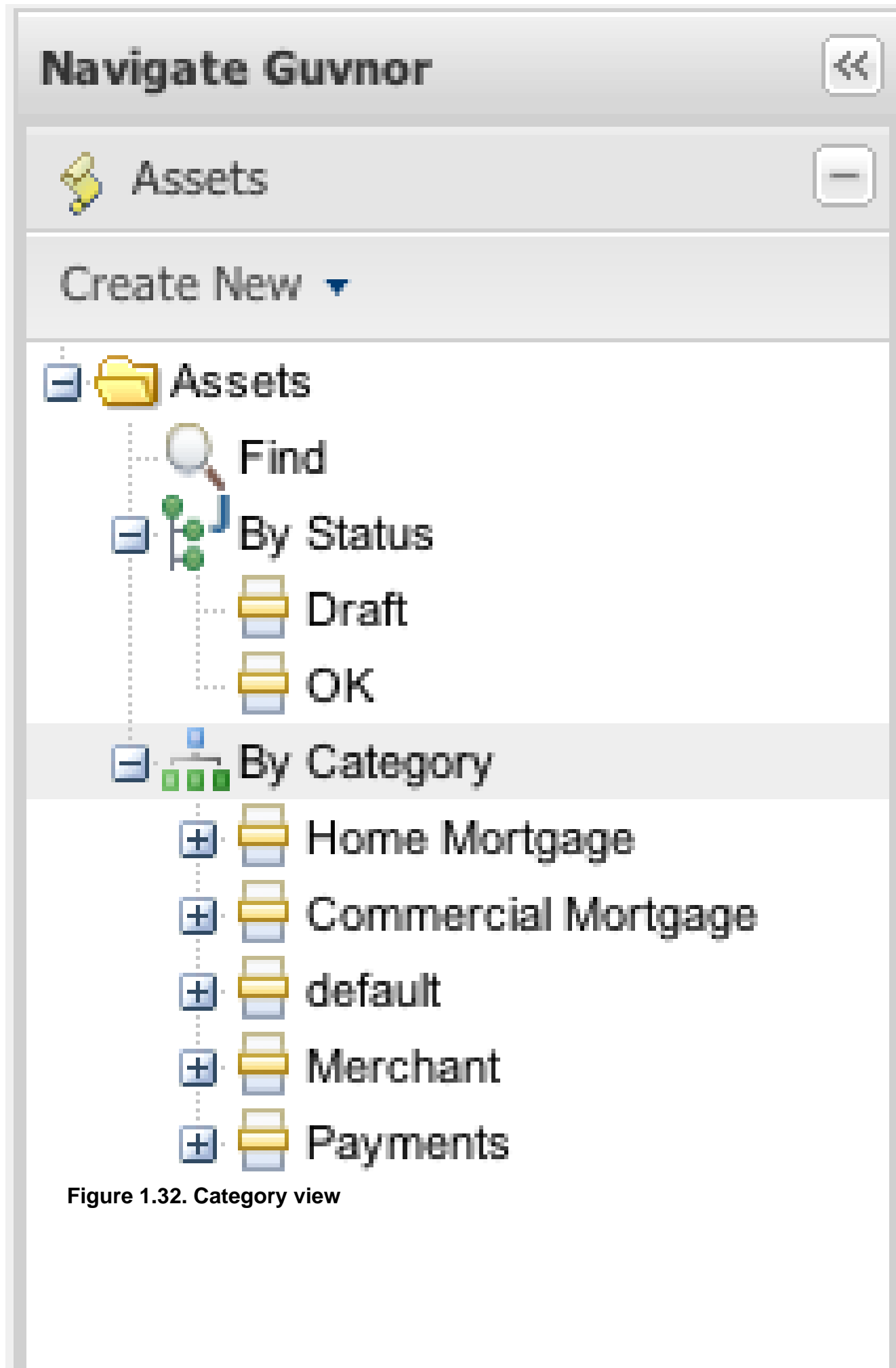


Figure 1.32. Category view

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).

The screenshot shows the 'Navigate Guvnor' interface. On the left is a package explorer tree with 'Business rule assets' selected. On the right is a table titled 'Business rule as' showing a list of 10 items.

Name	Last modified	Status	Categories
Underage	Dec 19, 2008	Draft	Eligibility rules
Bankruptcy history	Oct 1, 2008	Draft	Eligibility rules
No bad credit checks	Oct 1, 2008	Draft	Eligibility rules
no NINJAs No ninjas !	Oct 2, 2008	Draft	Eligibility rules
Pricing loans	Jan 27, 2009	Draft	Pricing rules
CreditApproval	Oct 22, 2008	Draft	Eligibility rules
DateDslRule	Oct 24, 2008	Draft	Technical
CheckBoxDslRule	Oct 23, 2008	Draft	Technical
RegexDslRule	Oct 23, 2008	Draft	Technical
wee	Jan 27, 2009	Draft	Home Mortgage

Figure 1.33. Package view

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

1.4.3. 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.

1.4.4. 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.

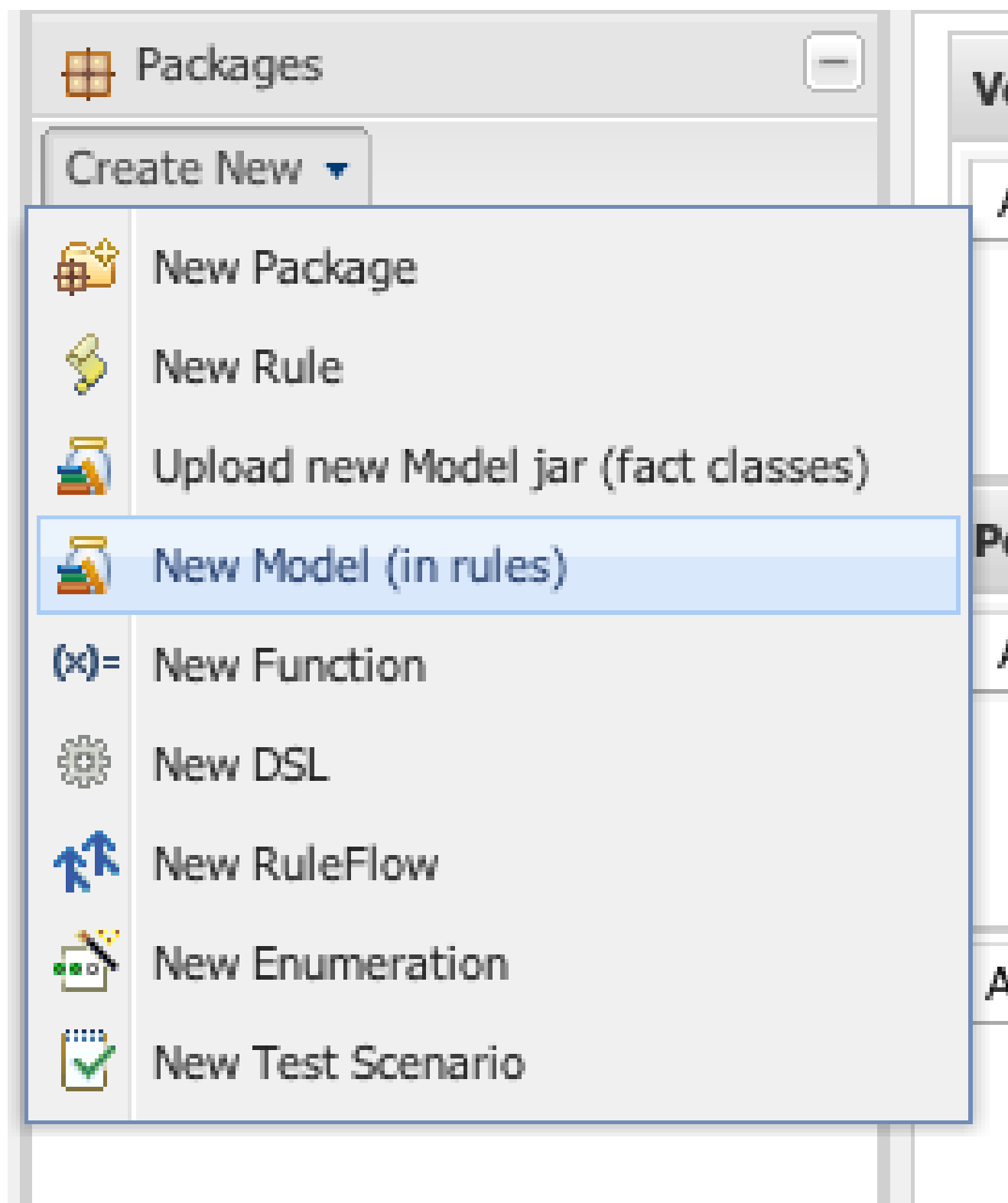


Figure 1.34. Choosing a model type

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:

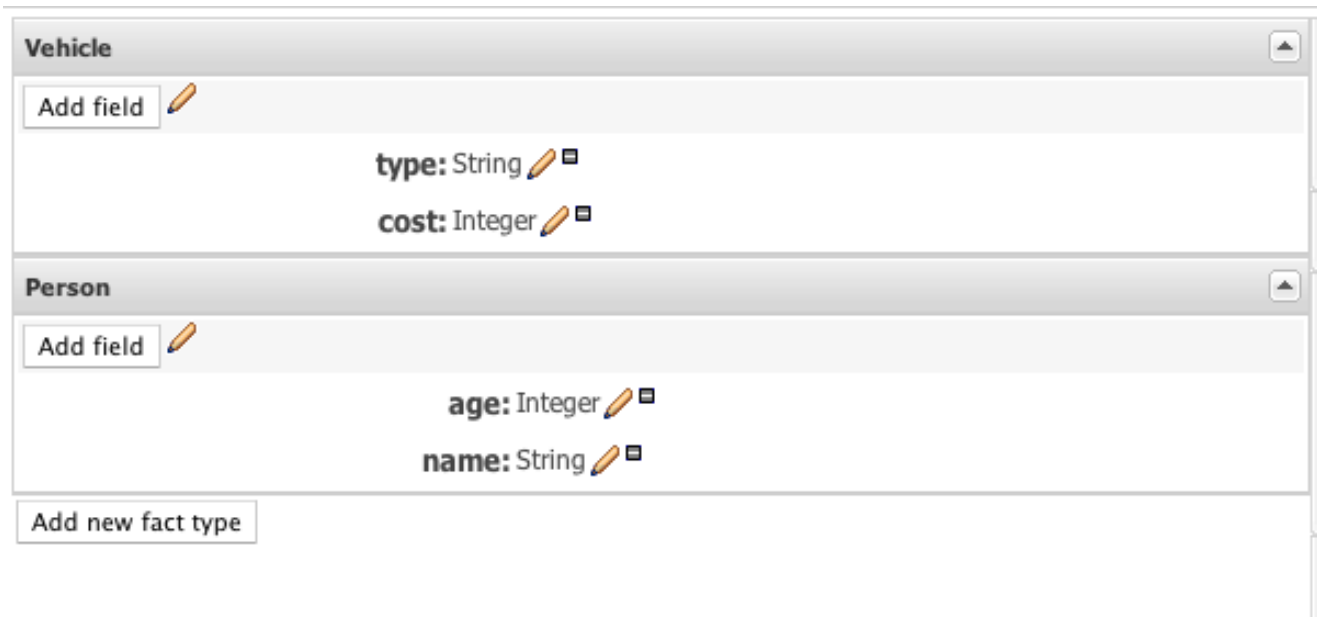


Figure 1.35. Choosing a model type

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):

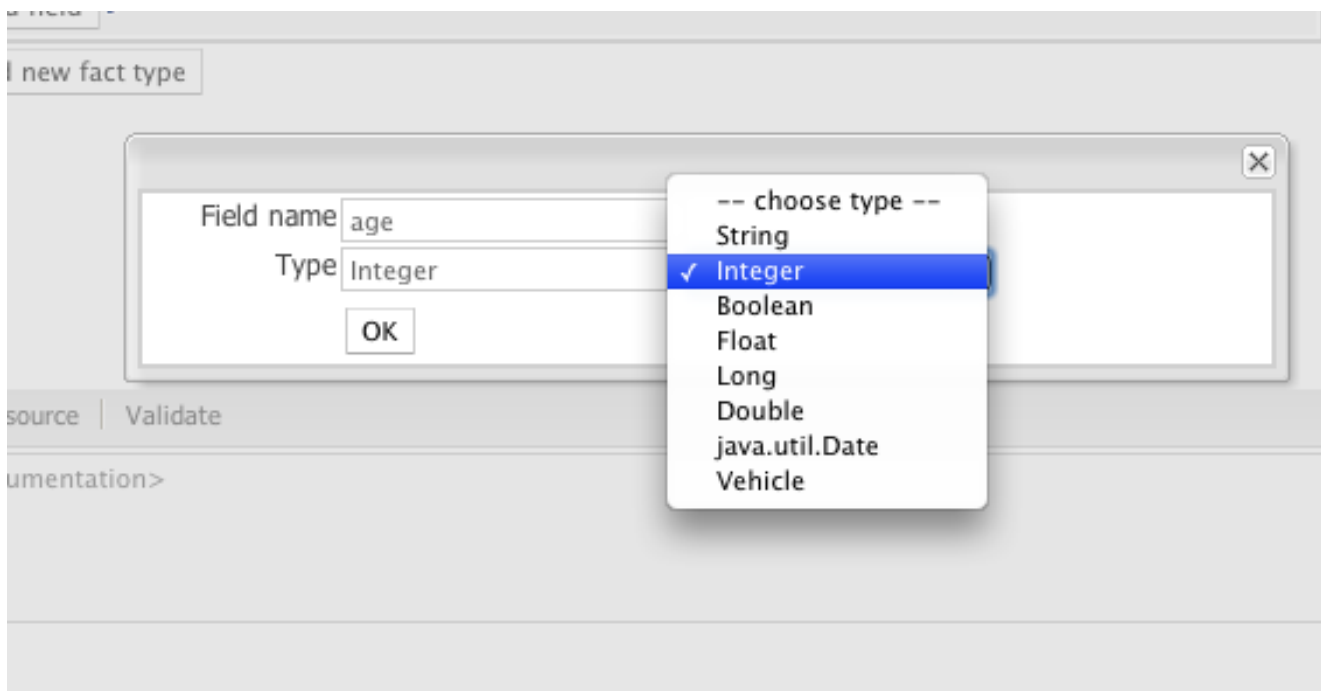


Figure 1.36. Choosing a model type

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 lifecycle separate from the application. It also removed the hassle of keeping jar files in sync between rules and the applications that use the rules.

1.4.5. The business user perspective

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

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

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

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

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

1.4.6. Advanced config options in a rule package

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

1.4.7. 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.

1.4.7.1. 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 Guvnor, 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 Guvnor at all - the drools-ant task will generate .pkg files the same as Guvnor.

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

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 and away you go !
```

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 Guvnor. You can have one or many. The name of the file is not important. Each package must be in its own file.

NOTE: 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 Guvnor which is exposing the packages (see below for more details).

- localCacheDir

This is used in conjunction with the url above, so that if the Guvnor 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 Guvnor, which provides URLs and downloads of packages.

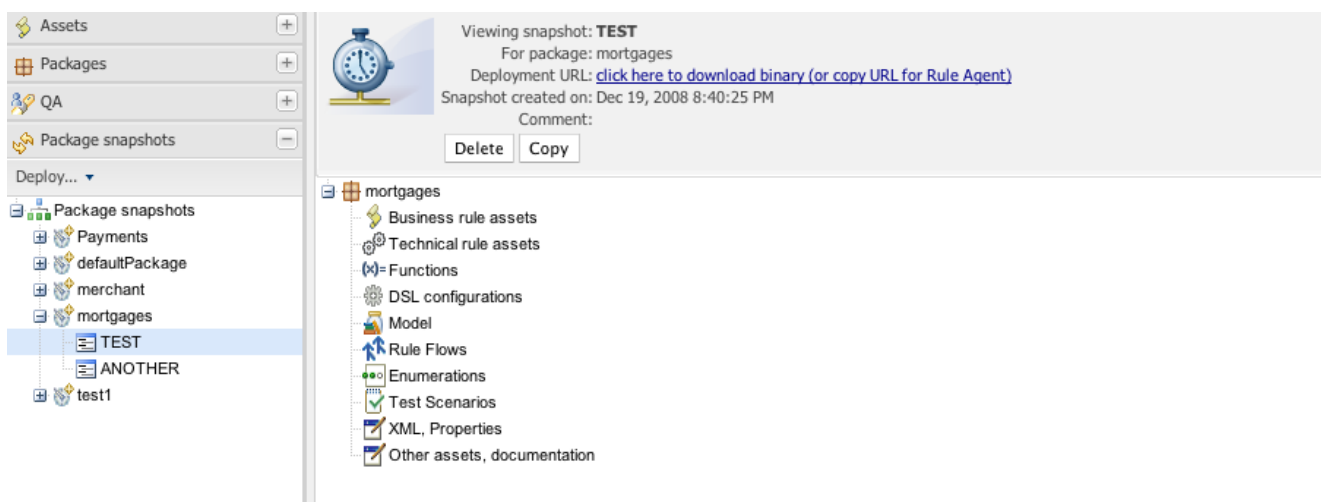


Figure 1.37. Snapshot deployment

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 Guvnor for updates - but that is generally the easiest way for many people).

1.4.7.2. Drools execution server

IMPORTANT NOTE: the exact message formats described here are likely to slightly change post drools 5 (as support for processes and stateful sessions is added). The following will be preserved for compatability, but may be deprecated in favour of more flexible formats that are in the pipelines.

The drools execution server (drools-server) module is a war which you can deploy to execute knowledgebases (rulebases) remotely for any sort of client application. This is not limited to JVM application clients, but any technology that can use HTTP, and either XML or JSON. Currently this execution server is for stateless sessions (which also makes it easy to scale out).

A "restful" style of interface is provided, with URLs defining what knowledgebase is being accessed. Client applications then execute knowledgebases remotely via HTTP(S). The rule execution server uses the rule agent described in the sections above.

1.4.7.2.1. Configuration and deployment

drools-server is a war file, which can be deployed in a application server (such as JBoss AS). As the service is stateless, it is possible to have have as many of these services deployed as you need to serve the client load. The war file should be expanded to a folder for deployment (which will be clear later) - so you can deploy this as a folder called drools-server.war.

Execution server is running

This server allows you to execute rules/knowledge bases remotely using a RESTful interface. One service per RuleAgent.

Stateless services

URL:

`http://{server address, port etc}/drools-server/knowledgebase/{configurationName}`
a HTTP POST to this URL will perform a stateless execution of the knowledgebase/rules.

The {configurationName} is the name of a configured rule agent properties file (not including the .properties extension) with both Guvnor, but also DRL files, etc). This properties file must be in the classpath for this war - in the WEB-INF/

Sample request content:

```
<knowledgebase-request>
  <globals>
    <named-fact>
      <id>myglobal</id>
      <fact class="org.drools.server.ExampleFact">
        <carType>Saab</carType>
        <carPrice>42</carPrice>
      </fact>
    </named-fact>
  </globals>
  <inOutFacts>
    <named-fact>
      <id>myfact</id>
      <fact class="org.drools.server.ExampleFact">
        <carType>BMW</carType>
        <carPrice>50</carPrice>
      </fact>
    </named-fact>
  </inOutFacts>
  <inFacts>
    <anon-fact>
```

Sample response content:

```
<knowledgebase-response>
```

Once the war has been deployed, you should be able to go to `http://localhost:8080/drools-server` and see a page like the above (obviously substitute a correct server address). The page shown should provide information on how to use it, and indicates that it is healthy.

To configure a "knowledgebase" - you create a properties file with a rule agent configuration in it (see the section above for details on the options in the rule agent configuration). The name of this properties file will be the name in the URL that you use to access this service. So let's take the example of `teamallocation.properties`: an agent configuration for a service that will allocate some piece of work to a team. `teamallocation.properties` should contain details on what packages are to be loaded, and from where.

Place `teamallocation.properties` in the `WEB-INF/classes` directory in the war directory (and restart the server if needed). Importantly, you will ALSO need any jar files that your rules need placed in the `WEB-INF/lib` directory of each execution server instance (eg your model, if you are using pojo

models, or supporting classes if needed - you may not need any). Typically if you are using the execution server, your client code will not use a object model jar to talk to the service so

Once this is done, the server is accessed via a url of the pattern: `http://your-server/drools-server/knowledgebase/{agent configuration name}`. So in the example above it would be:

```
http://your-server/drools-server/knowledgebase/teamallocation
```

You can see from this that you can have as many agents configured as you like, perhaps for many concurrent versions of knowledgebases.

1.4.7.2.2. Consuming the service

Consuming the service is quite simple, you need to choose to use either XML or JSON. By default XML is used, but by setting the Content-Type HTTP header to `application/json`, JSON will be used (JSON can be more performant in some circumstances).

Client libraries: one of the nice things about a REST service is that no special client libraries are required, regardless of the language used. In java however, Apache commons "HttpClient" is recommended as an easier API to use than the defaults. Only HTTP is required (of course https can be used for secure transport if on an untrusted network).

A sample request:

```
<knowledgebase-request>
  <globals>
    <named-fact>
      <id>myglobal</id>
      <fact class="org.drools.server.ExampleFact">
        <carType>Saab</carType>
        <carPrice>42</carPrice>
      </fact>
    </named-fact>
  </globals>
  <inOutFacts>
    <named-fact>
      <id>myfact</id>
      <fact class="org.drools.server.ExampleFact">
        <carType>BMW</carType>
        <carPrice>50</carPrice>
      </fact>
    </named-fact>
  </inOutFacts>
  <inFacts>
    <anon-fact>
      <fact class="org.drools.server.ExampleFact">
        <carType>Audi</carType>
        <carPrice>55</carPrice>
      </fact>
    </anon-fact>
  </inFacts>
</knowledgebase-request>
```

```

</anon-fact>
<anon-fact>
  <fact class="org.drools.server.ExampleFact">
    <carType>Mercedes</carType>
    <carPrice>65</carPrice>
  </fact>
</anon-fact>
</inFacts>
</knowledgebase-request>

```

Elements of the request: note that there are 3 parts: globals, inOutFacts and inFacts. Both globals and inOutFacts are returned in the response message. Globals and inOutFacts are named (each name must be unique) for this purpose (in the case of globals, the name of the global is the name used in the rules). The "id" tag is used for the name. Note that the "fact" tag refers to a fact as used by the knowledgebase - the fields inside that enclosing tag are defined by the fact class itself. If a fact class has a nested class, then the data for that nested data would show up as <nestedFactFieldName> - where nestedFactFieldName is the name of the field in the "parent" class. Inside that tag are the tags with the values of the fields for that nested data (the class name is not needed as that is derived from the parent fact).

A sample response:

```

<knowledgebase-response>
  <globals>
    <named-fact>
      <id>myglobal</id>
      <fact class="org.drools.server.ExampleFact">
        <carType>Saab</carType>
        <carPrice>42</carPrice>
      </fact>
    </named-fact>
  </globals>
  <inOutFacts>
    <named-fact>
      <id>myfact</id>
      <fact class="org.drools.server.ExampleFact">
        <carType>BMW</carType>
        <carPrice>50</carPrice>
      </fact>
    </named-fact>
  </inOutFacts>
</knowledgebase-response>

```

Elements of response: Similar to the request (note the enclosing tags are different) - of course only the global and inOutFacts have their state returned.

JSON: (Javascript Object Notation) follows the same basic object graph layout as the requests above. Some notes: in JSON, @class is used to indicate the type of the fact that the rules use.

Also, in javascript, associative arrays (maps) are indicated by "{" and "}", and arrays via "[" and "]". In some cases, if there would be a list (array) but only 1 element of data is present, then "[" will not be shown. eg {"a" : "b"} can have the same meaning as [{"a" : "b"}] in results.

A sample request:

```
{ "knowledgebase-request":
  { "globals": { "named-
    fact": { "id": "myglobal", "fact": { "@class": "org.drools.server.ExampleFact", "carType": "Saab", "ca
    "inOutFacts": { "named-
      fact": { "id": "myfact", "fact": { "@class": "org.drools.server.ExampleFact", "carType": "BMW", "carPr
      "inFacts": { "anon-
        fact": [ { "fact": { "@class": "org.drools.server.ExampleFact", "carType": "Audi", "carPrice": 55 } }, { "
```

A sample response:

```
{ "knowledgebase-response":
  { "globals": { "named-
    fact": { "id": "myglobal", "fact": { "@class": "org.drools.server.ExampleFact", "carType": "Saab", "ca
    "inOutFacts": { "named-
      fact": { "id": "myfact", "fact": { "@class": "org.drools.server.ExampleFact", "carType": "BMW", "carPr
```

Following is an example code snippet showing how a knowledgebase is accessed using JSON from Ruby:

```
require 'json'
http = Net::HTTP.new('localhost', 8080)
path = "/drools-server/knowledgebase/teamallocation"
post_data = { "knowledgebase-request" => {
  :globals => { "named-fact" => [{ :id => "a", :fact =>
    { "@class" => "teamallocation.Assignment" } ] },
  :inFacts => { "anon-fact" => [ { :fact => { "@class" =>
    "teamallocation.Claim", "value" => 150 } } ],
  :inOutFacts => { "named-fact" => [ { :id => "x", :fact =>
    { "@class" => "teamallocation.Team", "specialty" => "FATAL" } },
    { :id => "y", :fact =>
    { "@class" => "teamallocation.Team" } } ] }
  }
}

headers = {
  "Content-Type" => "application/json"
}
resp, data = http.post(path, post_data.to_json, headers)

answer = JSON.parse(data)
#digging out the results:
```

```
puts
answer["knowledgebase-response"]["globals"]["named-
fact"]["fact"]["teamName"]
#if there is more then one fact, they are a list
puts
answer["knowledgebase-response"]["inOutFacts"]["named-
fact"][0]["fact"]["specialty"]
```

1.4.7.3. 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).

1.4.7.4. 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 Guvnor are serialized Package objects. You can deserialize them and add them into any rulebase - essentially that is all you need to do.

From Guvnor, 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 Guvnor 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.

1.4.8. WebDAV and HTTP

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

1.4.8.1. WebDAV

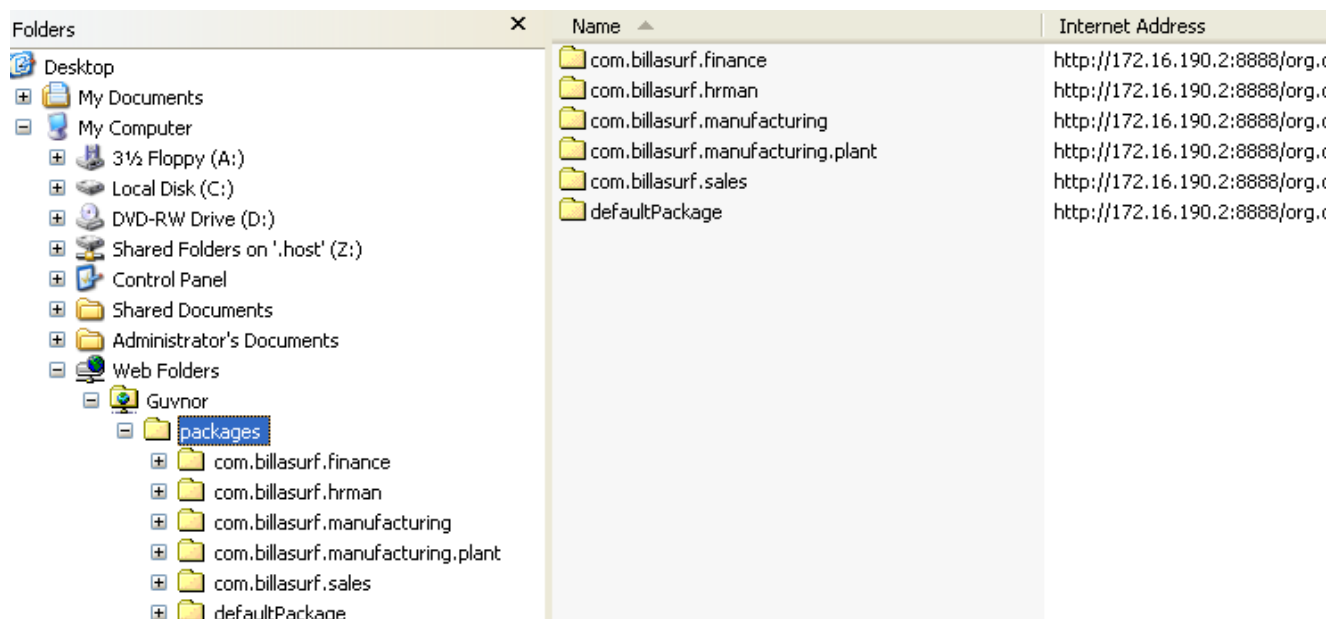


Figure 1.38. Windows webdav

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

1.4.8.2. 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. eg: /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). eg /package/testPDSGetPackage/LATEST/SomeFile.drl.

1.4.9. Eclipse Guvnor integration

The Eclipse Guvnor tools (EGT) provide the ability to push/pull artifacts from the Guvnor repository server and the developers workspace in eclipse. It is therefore possible for artifacts to be both managed via Guvnor as well as in traditional developer friendly SCM systems (such as subversion). The Guvnor repository is not intended as a Source Code Management (SCM) solution, and the EGT are not intended to be Eclipse "team provider" extensions or replacements. Rather, the Guvnor repository is a location where certain artifacts (such as rules and SOA policy

definitions) are controlled (“governed”) by policies defined by the deployment environment. The purpose of the EGT is then to enable access to resources held by the Guvnor repository, so they can be used in development. Thus, limited capabilities for reading, writing, adding, and removing Guvnor repository resources are provided in the EGT.

1.4.9.1. 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` and `org.eclipse.webdav` and requires Eclipse 3.3.x. The current Eclipse Drools plug-ins are also useful for viewing Guvnor repository resources such as rule definitions, but not required for operation of the EGT.

1.4.9.2. Functionality Overview

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

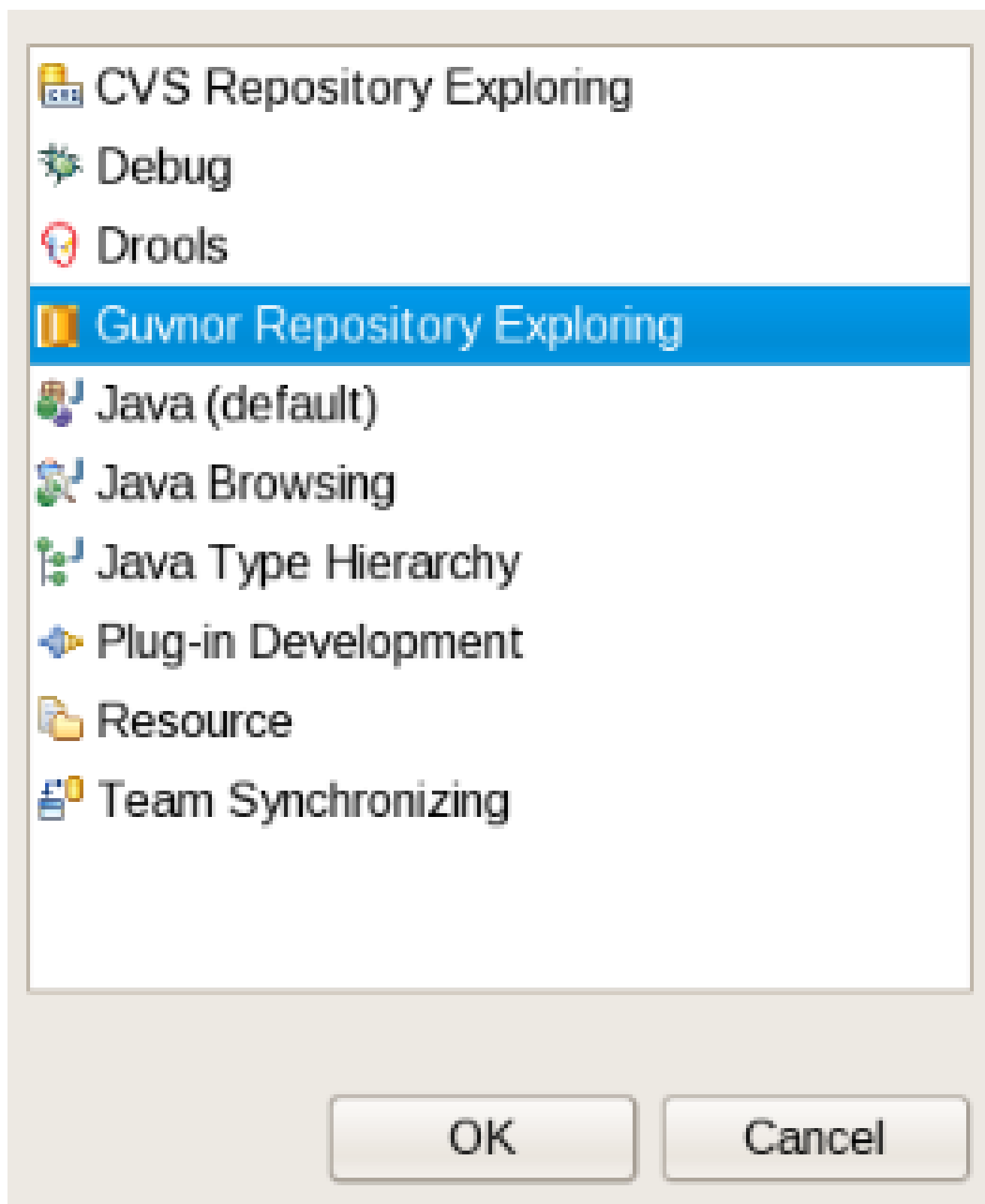


Figure 1.39. Views and perspectives

and then choose “Guvnor Repository Exploring.” This opens the Guvnor perspective:

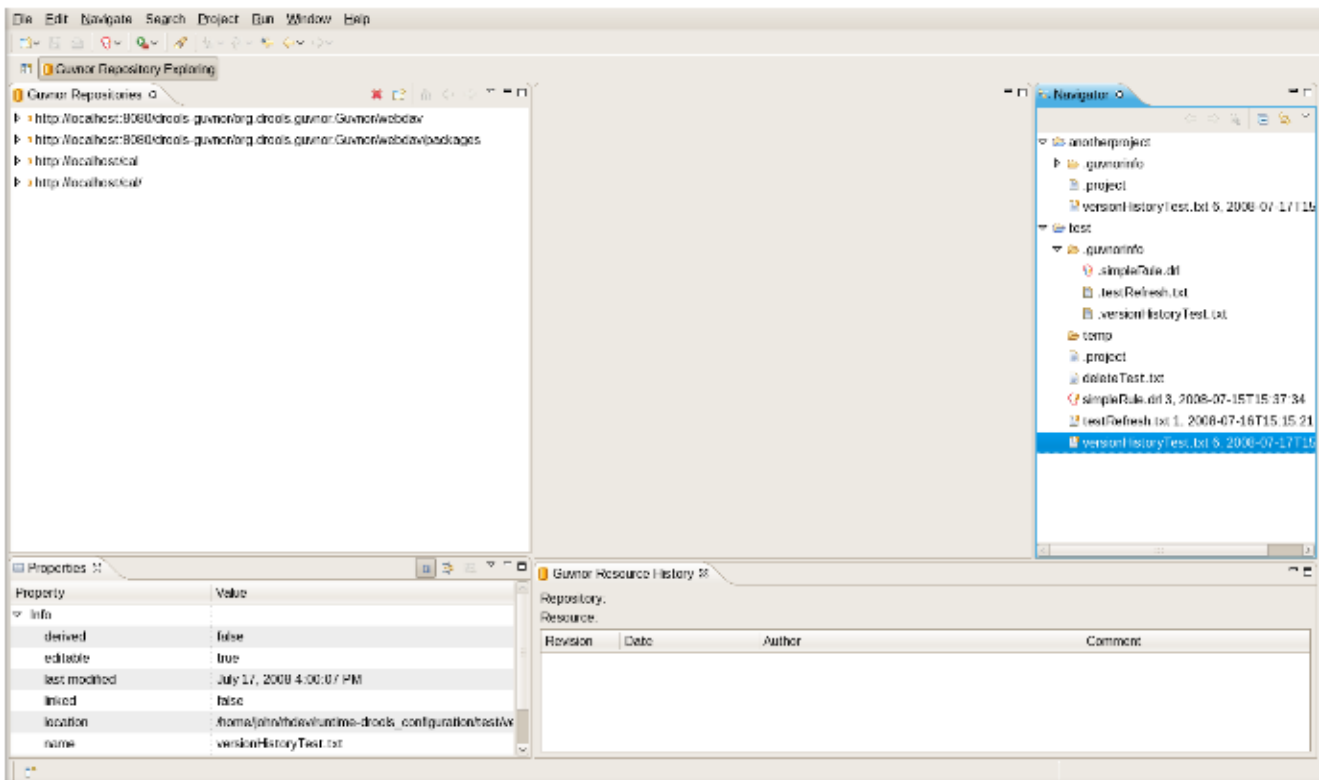


Figure 1.40. Views and perspectives

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

1.4.9.3. 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:

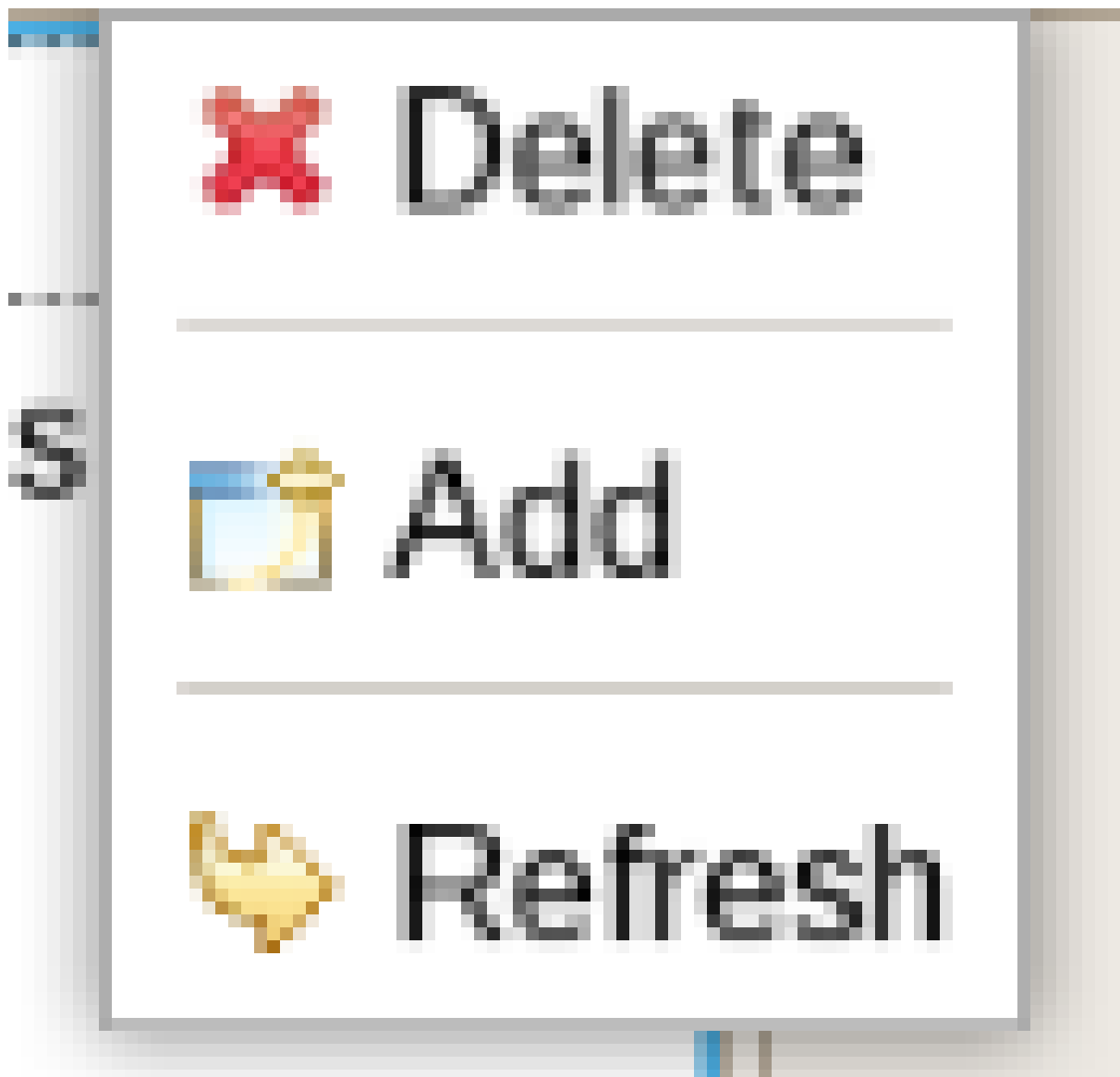


Figure 1.41. Connection wizard

or the menu button:

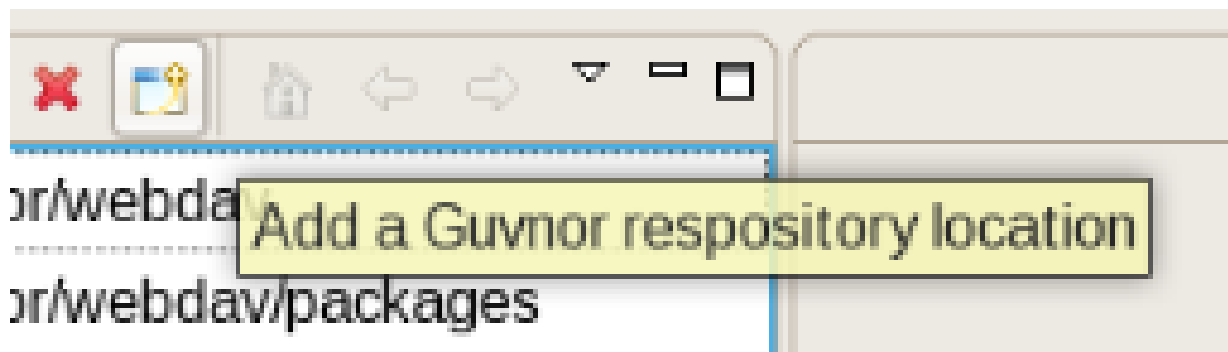


Figure 1.42. Connection wizard

Choosing either of these will start the Guvnor connection wizard:

New Guvnor location

Create a new Guvnor repository connection



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

Figure 1.43. 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:

Guvnor Repository Log in

Authentication required for repository: `http://localhost/cal`



User Name:

Password:

☐ Save user name and password

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




Figure 1.44. Login

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

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

1.4.9.4. Guvnor Repository Explorer

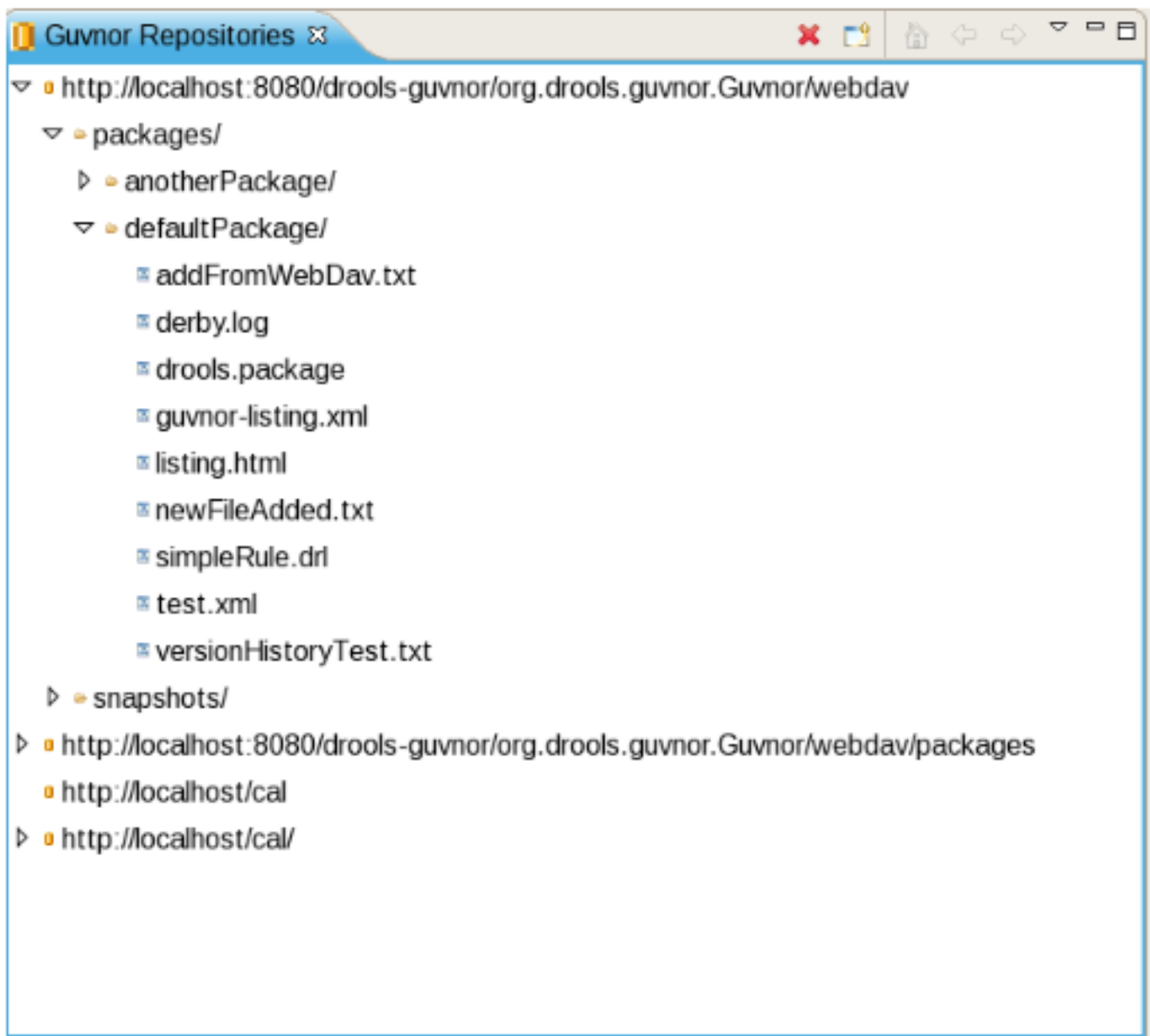


Figure 1.45. Explorer

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:

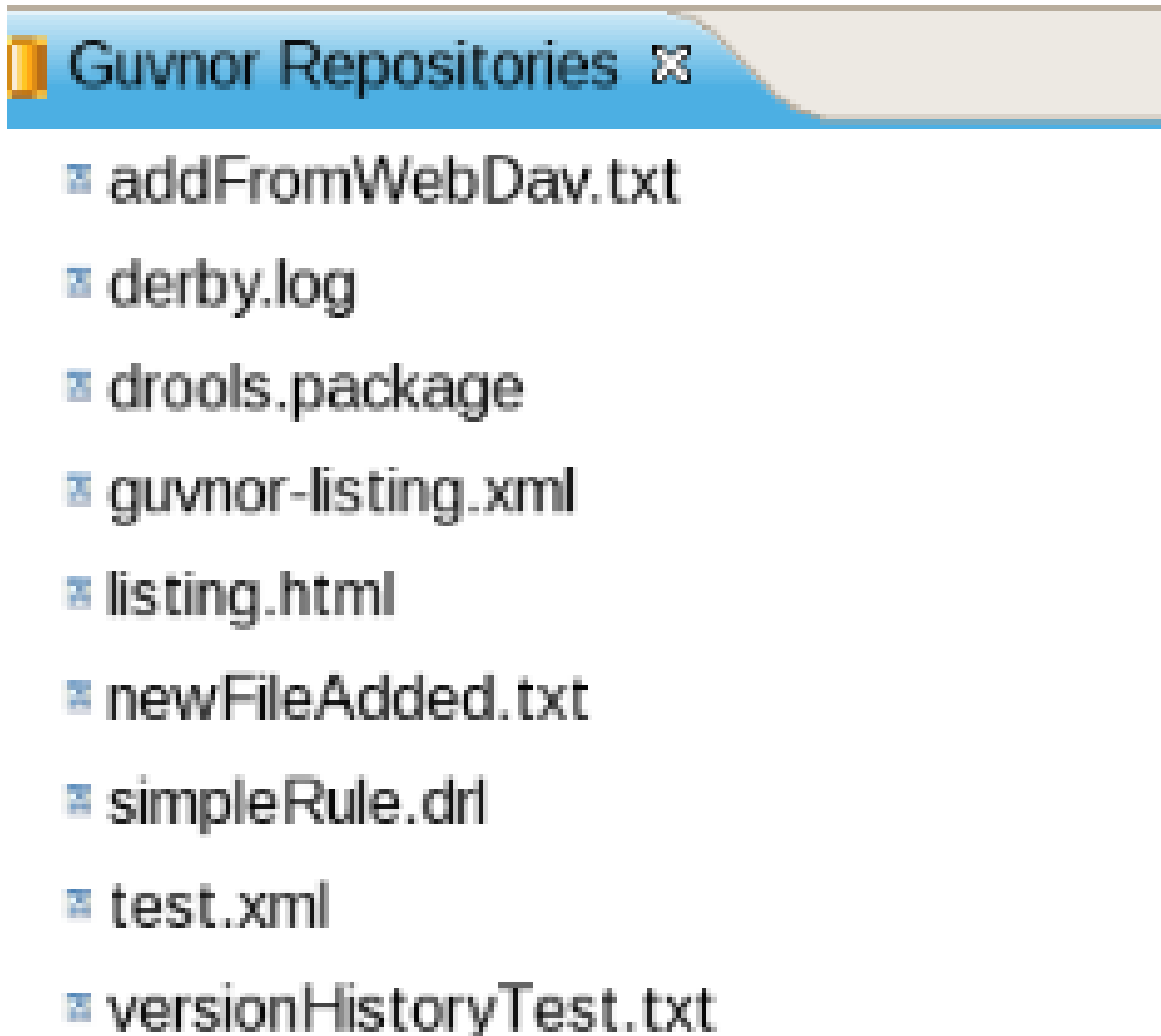
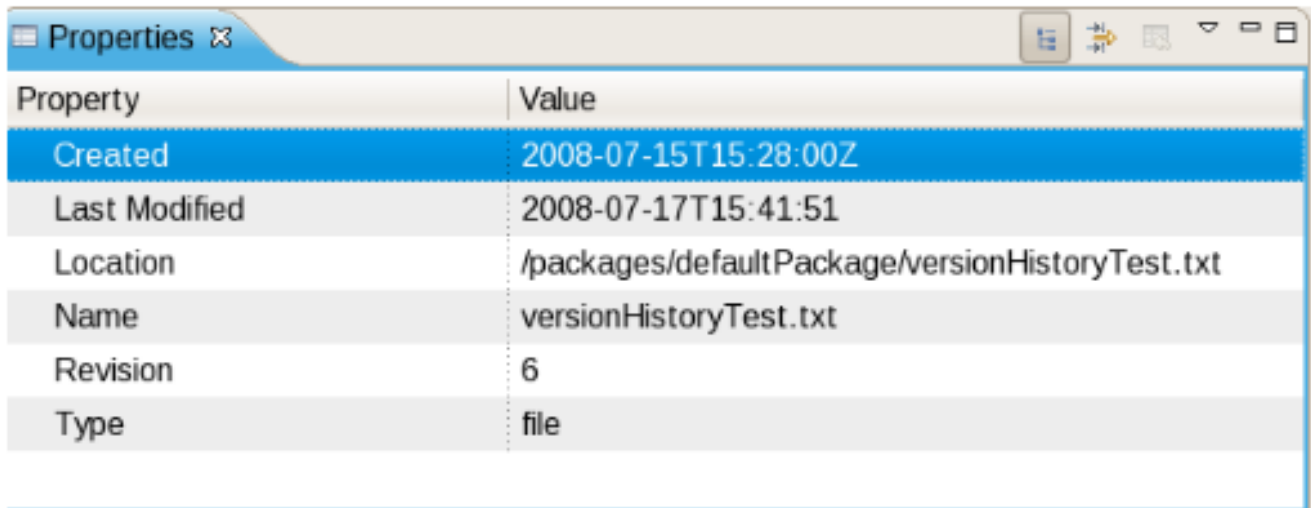


Figure 1.46. Explorer

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

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

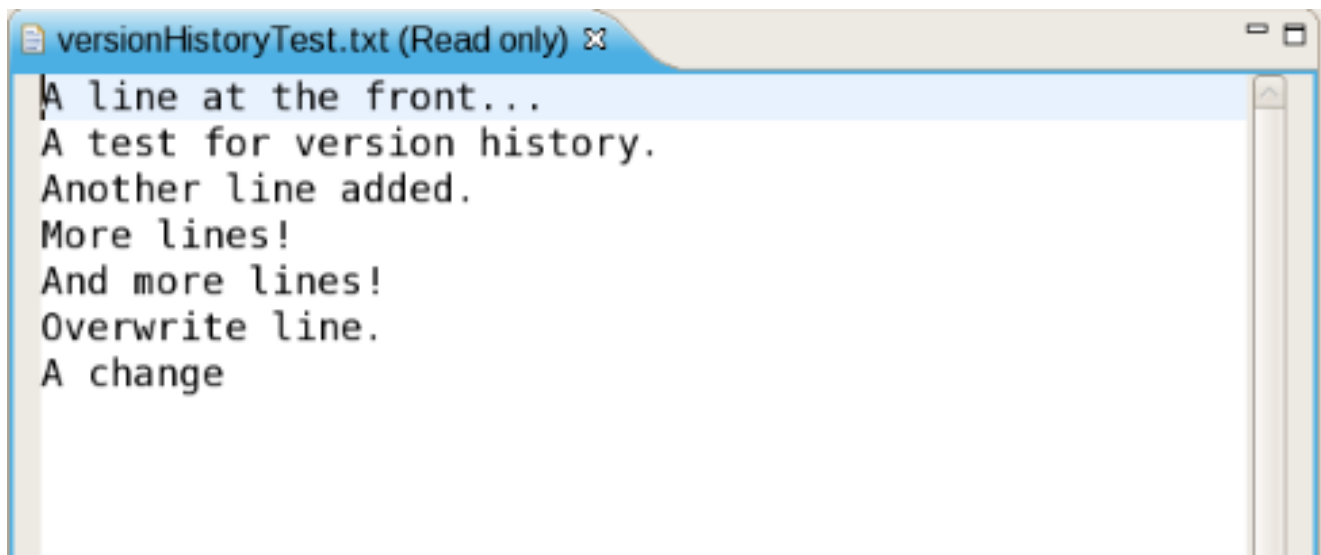


The screenshot shows the Eclipse Properties view for a file. The table lists the following properties and values:

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

Figure 1.47. 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:



The screenshot shows the Eclipse editor with a read-only file named versionHistoryTest.txt. The file contains the following text:

```
A line at the front...
A test for version history.
Another line added.
More lines!
And more lines!
Overwrite line.
A change
```

Figure 1.48. 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.)

1.4.9.5. Local Copies of Guvnor Files

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

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

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

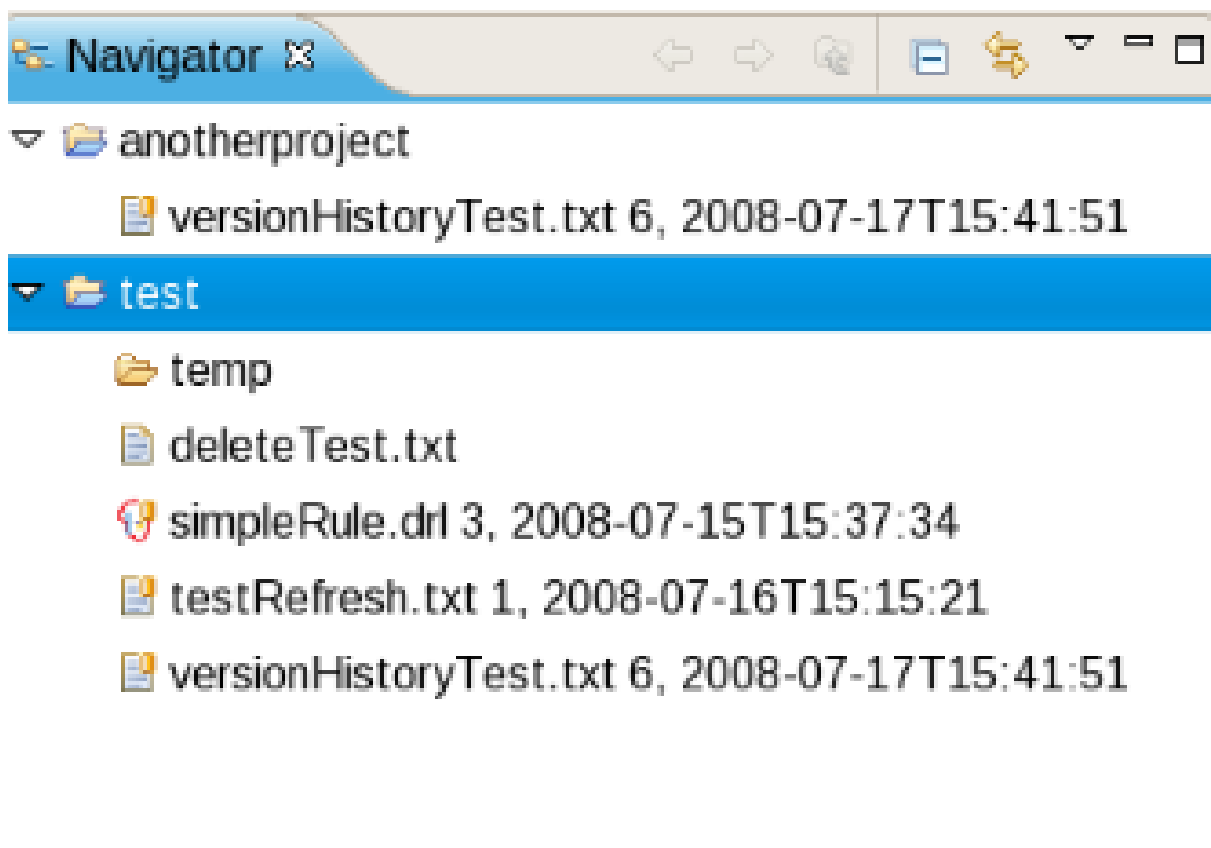


Figure 1.49. Navigator

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 1.50. 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.

1.4.9.6. 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 1.51. 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

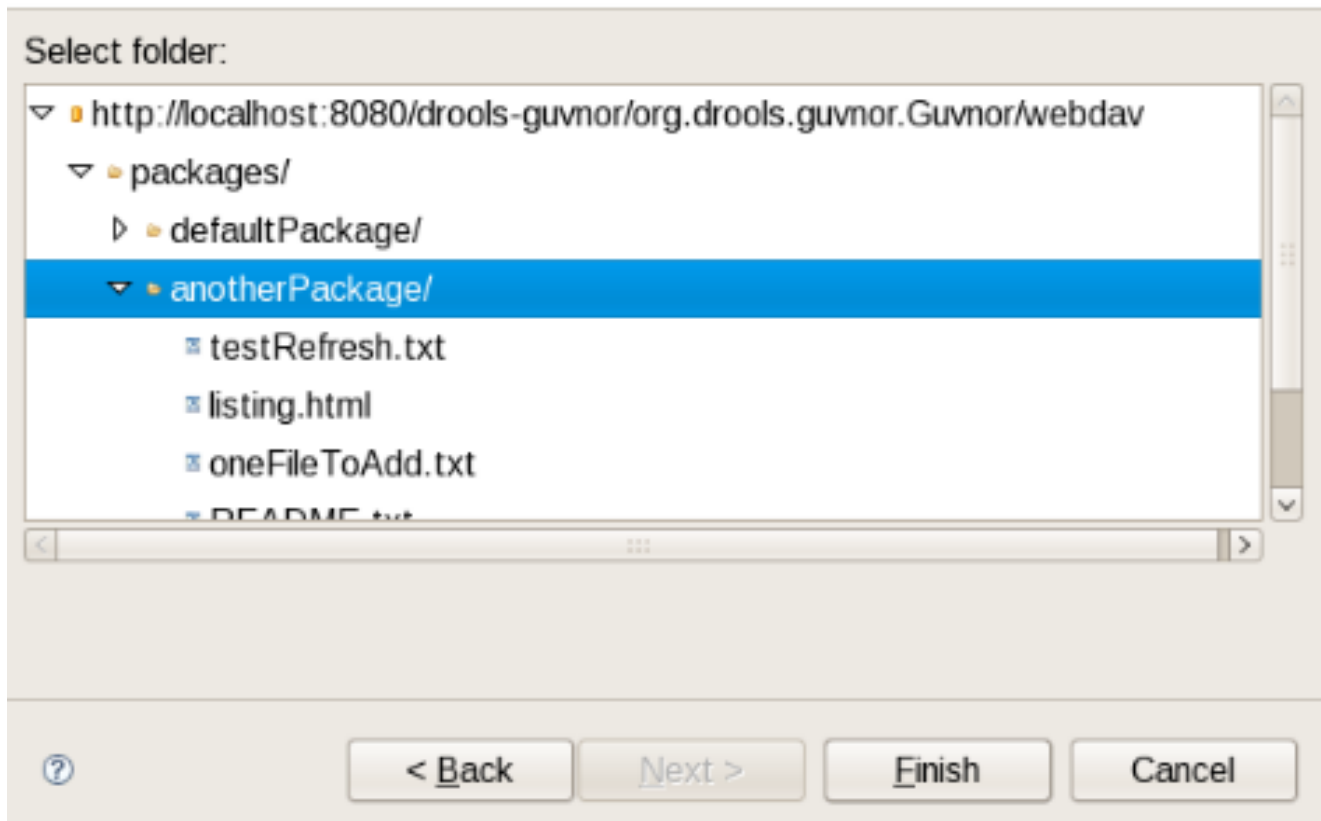


Figure 1.52. Add action

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

Compare with Version Action:

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



Resource Versions

Choose a version for versionHistoryTest.txt

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

?

OK Cancel

Figure 1.53. Compare

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

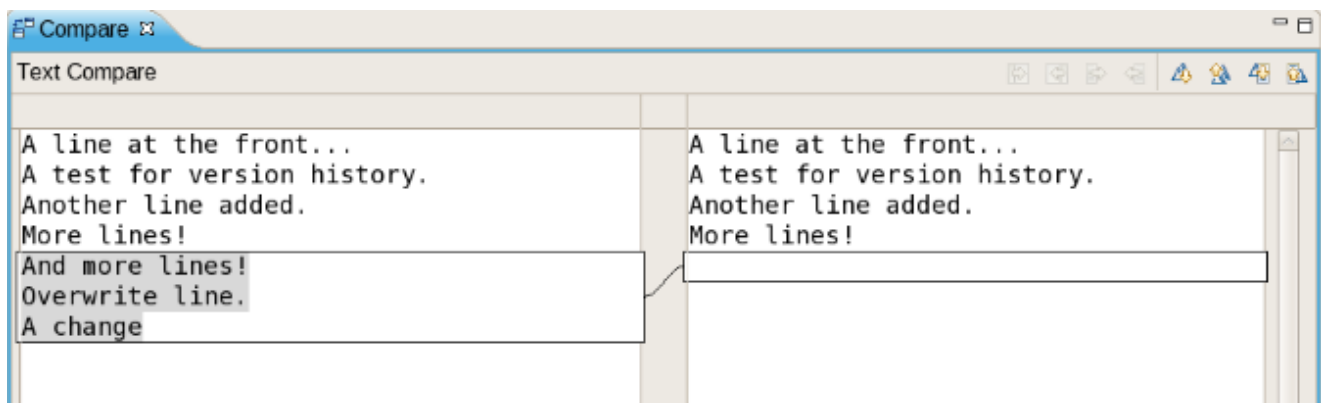


Figure 1.54. Compare

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

Switch to Version Action:

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

Resource Versions

Choose a version for versionHistoryTest.txt



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

Figure 1.55. 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:

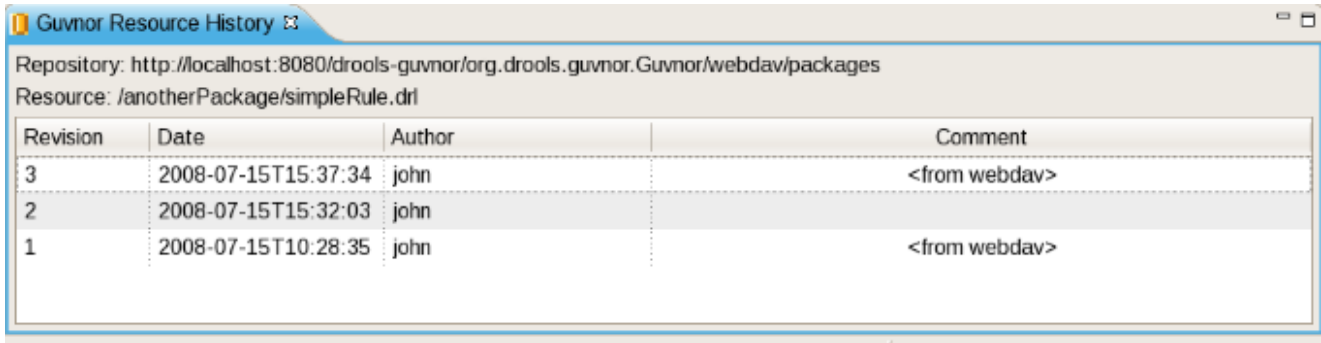
The Guvnor Resource History view should details about revision history for selected files, both local and those in Guvnor repositories. The initial state of this view is:

 A screenshot of the 'Guvnor Resource History' view. It has a title bar with the text 'Guvnor Resource History' and a close button. Below the title bar, there are labels 'Repository:' and 'Resource:'. The main area contains a table with the same structure as Figure 1.55, but it is currently empty.

Revision	Date	Author	Comment

Figure 1.56. 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:



The screenshot shows a window titled "Guvnor Resource History". Inside, it displays the repository URL "http://localhost:8080/drools-guvnor/org.drools.guvnor.Guvnor/webdav/packages" and the resource path "/anotherPackage/simpleRule.drl". Below this is a table with four columns: Revision, Date, Author, and Comment. The table contains three rows of data, each representing a revision of the file.

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 1.57. 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.)

1.4.9.7. Importing Guvnor Repository Resources

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

Select Guvnor repository location

Select an existing Guvnor repository location or create a new one



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

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



< Back

Next >

Finish

Cancel

Figure 1.58. Import

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

Select resources

Select resources to copy from the Guvnor repository

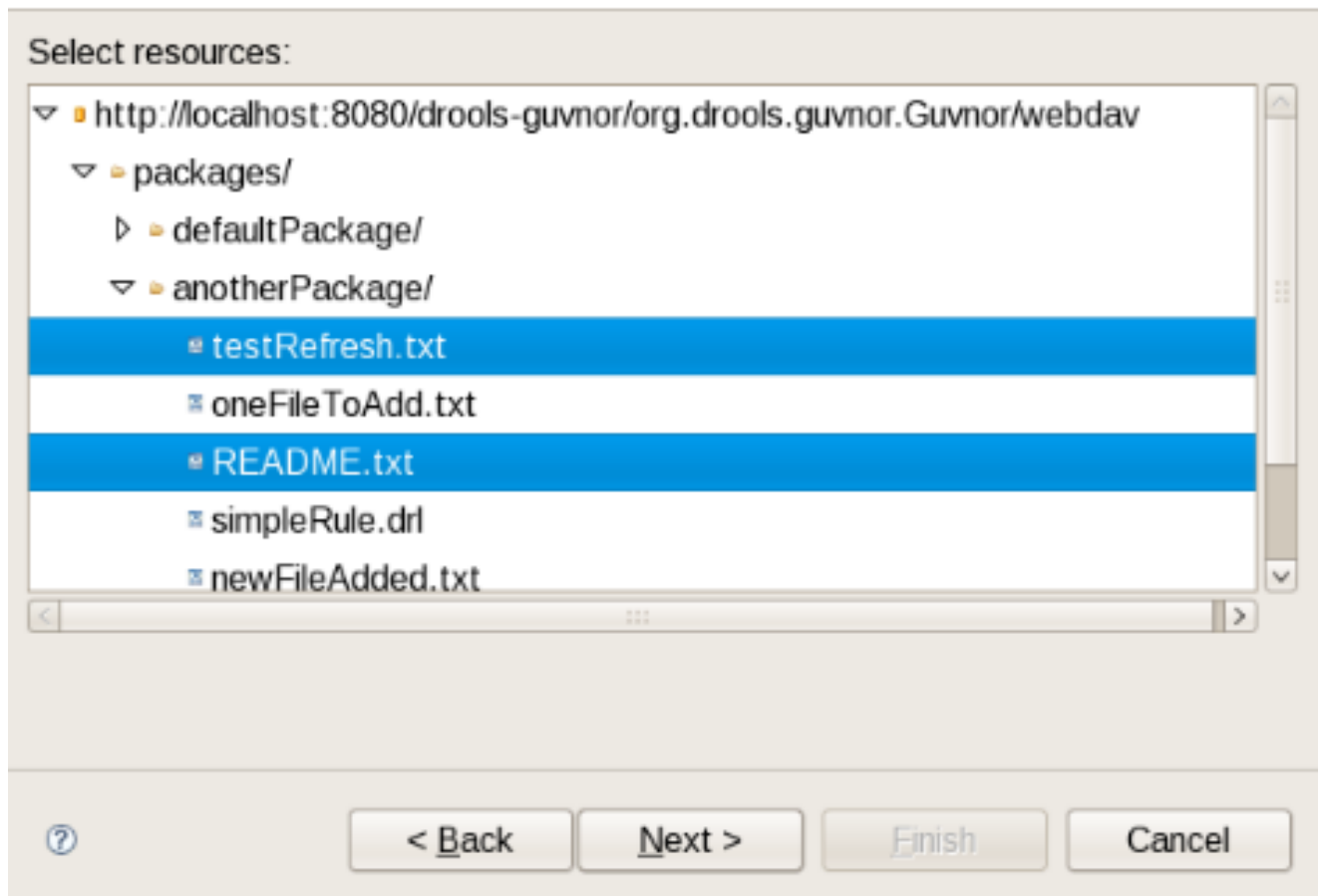


Figure 1.59. Import

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

Select copy location

Select the destination location

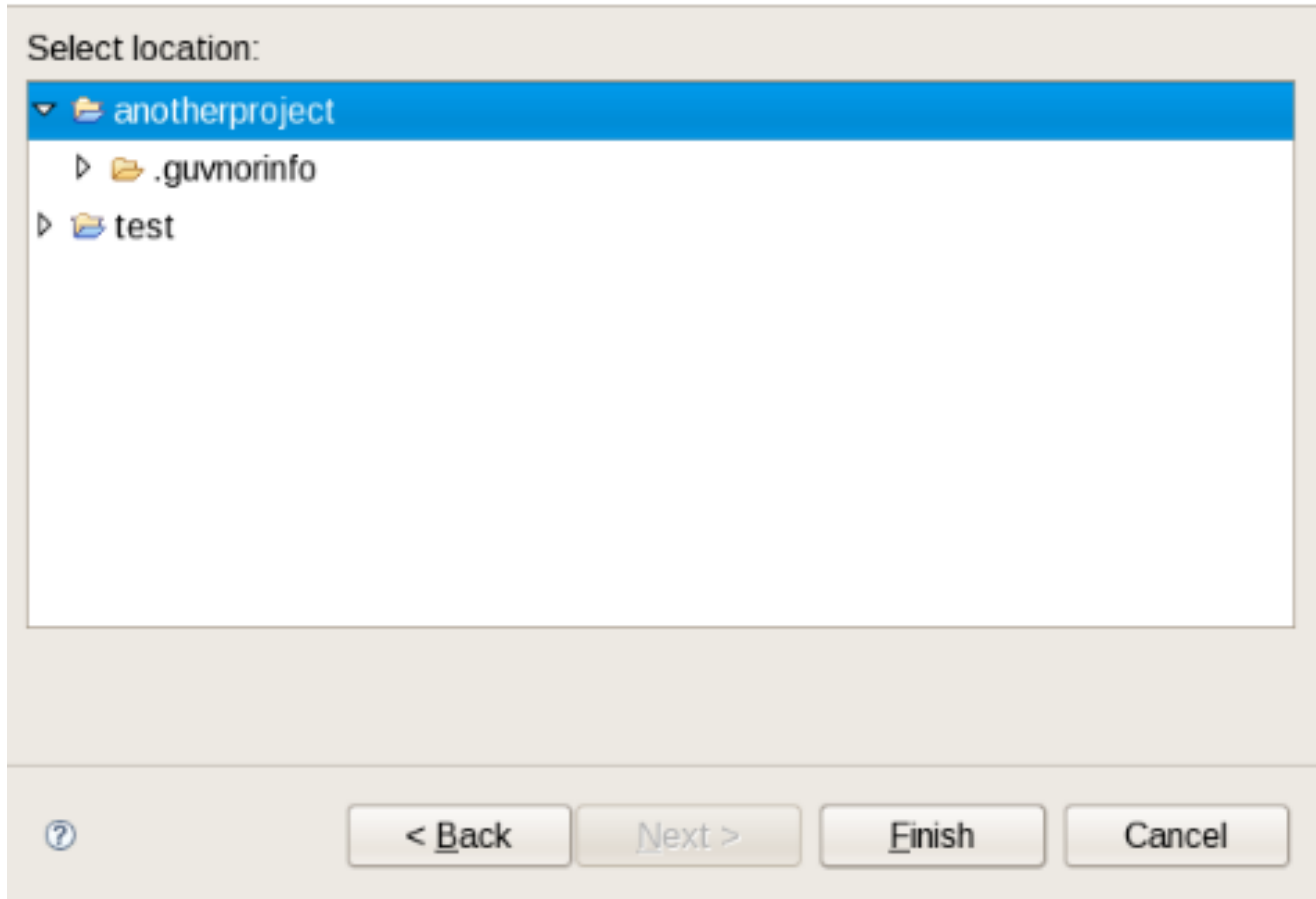


Figure 1.60. Import

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:

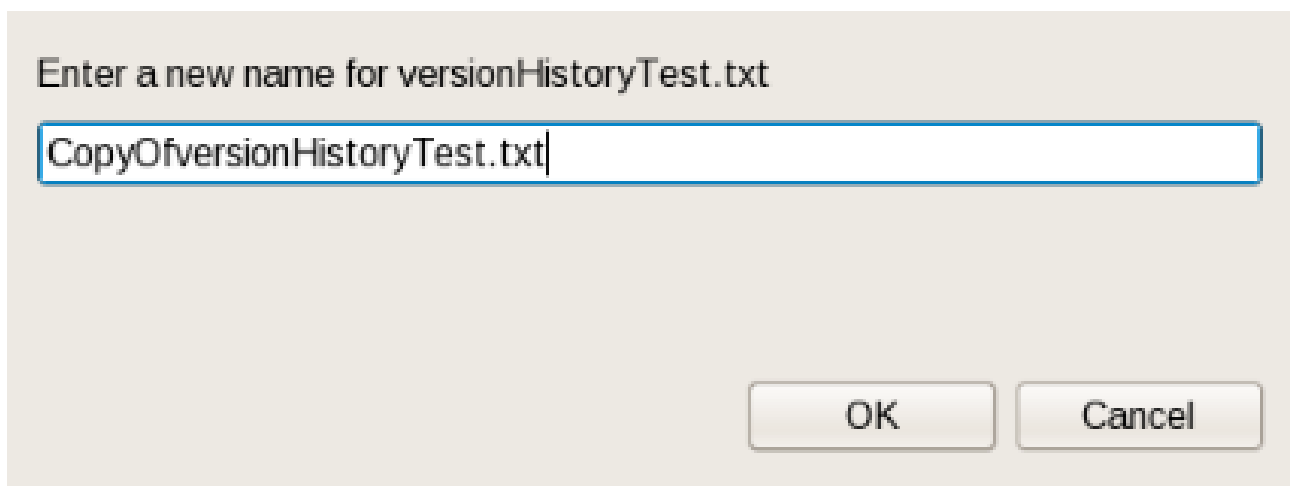


Figure 1.61. Copy

1.4.9.8. Guvnor plugin Preferences

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

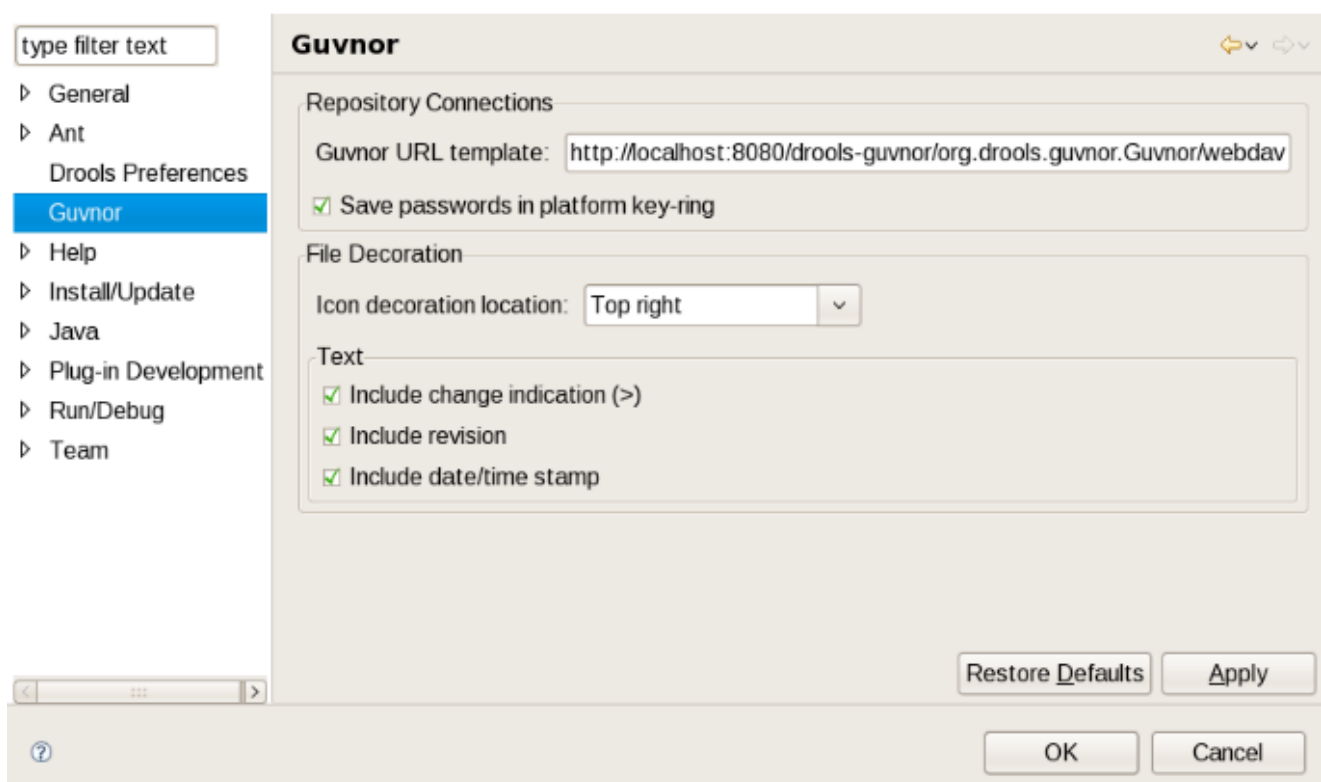


Figure 1.62. Preferences

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.

Index

