Drools Introduction and General User Guide


Drools
Business Logic
integration
Platform /

Chapter 1. Welcome

I've always stated that end business users struggle understanding the differences between rules and processes, and more recently rules and event processing. For them they have this problem in their mind and they just want to model it using some software. The traditional way of using two vendor offerings forces the business user to work with a process oriented or rules oriented approach which just gets in the way, often with great confusion over which tool they should be using to model which bit.

PegaSystems and Microsoft have done a great job of showing that the two can be combined and a behavioural modelling approach can be used. This allows the business user to work more naturally where the full range of approaches is available to them, without the tools getting in the way. From being process oriented to rule oriented or shades of grey in the middle - whatever suites the problem being modelled at that time.

Drools 5.0 takes this one step further by not only adding BPMN2 based workflow with Drools Flow but also adding event processing with Drools Fusion, creating a more holistic approach to software development. Where the term holistic is used for emphasizing the importance of the whole and the interdependence of its parts.

Drools 5.0 is now split into 5 modules, each with their own manual - Guvnor (BRMS/BPMS), Expert (Rules), Fusion (CEP), Flow (Process/Workflow) and Planner. Guvnor is our web based governance system, traditionally referred to in the rules world as a BRMS. We decided to move away from the BRMS term to a play on governance as it's not rules specific. Expert is the traditional rules engine. Fusion is the event processing side, it's a play on data/sensor fusion terminology. Flow is our workflow module, Kris Verlaenen leads this and has done some amazing work; he's currently moving flow to be incorporated into jBPM 5. The fith module called Planner, authored by Geoffrey De Smet, solves allocation and scheduling type problem and while still in the early stage of development is showing a lot of promise. We hope to add Semantics for 2011, based around description logc, and that is being work on as part of the next generaion Drools designs.

I've been working in the rules field now for around 7 years and I finally feel like I'm getting to grips with things and ideas are starting to gel and the real innovation is starting to happen. To me It feels like we actually know what we are doing now, compared to the past where there was a lot of wild guessing and exploration. I've been working hard on the next generation Drools Expert design document with Edson Tirelli and Davide Sottara. I invite you to read the document and get involved, *http://community.jboss.org/wiki/DroolsLanguageEnhancements*. The document takes things to the next level pushing Drools forward as a hybrid engine, not just a capable production rule system, but also melding in logic programming (prolog) with functional programming and description logic along with a host of other ideas for a more expressive and modern feeling language.

I hope you can feel the passion that my team and I have while working on Drools, and that some of it rubs off on you during your adventures.

Mark Proctor
Drools Creator and Lead

Chapter 2. Drools Release Notes

2.1. What is New and Noteworthy in Drools 5.1.0

2.1.1. Drools API

As in Drools 5.0 it is still possible to configure a `KnowledgeBase` using configuration, via a xml change set, instead of programmatically. However the change-set namespace is now versioned. This means that for Drools 5.1, the 1.0.0 xsd should be referenced.

Example 2.1. Here is a simple version 1.0.0 change set

```
<change-set xmlns='http://drools.org/drools-5.0/change-set'
xmlns:xs='http://www.w3.org/2001/XMLSchema-instance'
xs:schemaLocation='http://drools.org/drools-5.0/change-set change-set-5.0.xsd http://
anonsvn.jboss.org/repos/labs/labs/jbossrules/trunk/drools-api/src/main/resources/change-
set-1.0.0.xsd' >
  <add>
    <resource source='classpath:org/domain/someRules.drl' type='DRL' />
    <resource source='classpath:org/domain/aFlow.drf' type='DRF' />
  </add>
</change-set>
```

2.1.2. Core

2.1.2.1. JMX Monitoring

JMX monitoring was added to support KnowledgeBase monitoring. This is specially importand for long running processes like the ones usually required for event processing. Initial integration with JOPR was also added. JMX can be inabled with using the properties setting the knowledge base:

drools.mbeans = <enabled|disabled>

or this option at runtime

kbaseConf.setOption(MBeansOption.ENABLED)

2.1.2.2. Spring

Drools now has extensive Spring support, the XSD can be found in the the drools-spring jar. The namespace is "http://drools.org/schema/drools-spring"

Example 2.2. KnowledgeBuilder example

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
      xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
      xmlns:drools="http://drools.org/schema/drools-spring"
      xmlns:camel="http://camel.apache.org/schema/spring"
                  xsi:schemaLocation="http://www.springframework.org/schema/beans    http://
www.springframework.org/schema/beans/spring-beans-2.0.xsd
                                                                  http://drools.org/schema/drools-
spring    http://anonsvn.jboss.org/repos/labs/labs/jbossrules/trunk/drools-container/drools-spring/
src/main/resources/org/drools/container/spring/drools-spring-1.0.0.xsd
                  http://camel.apache.org/schema/spring http://camel.apache.org/schema/spring/
camel-spring.xsd">
  <drools:resource id="resource1" type="DRL" source="classpath:org/drools/container/spring/
testSpring.drl"/>

 <drools:kbase id="kbase1">
  <drools:resources>
   <drools:resource type="DRL" source="classpath:org/drools/container/spring/testSpring.drl"/>
    <drools:resource ref="resource1"/>
                              <drools:resource      source="classpath:org/drools/container/spring/
IntegrationExampleTest.xls" type="DTABLE">
      <drools:decisiontable-conf input-type="XLS" worksheet-name="Tables_2" />
    </drools:resource>
  </drools:resources>

  <drools:configuration>
    <drools:mbeans enabled="true" />
    <drools:event-processing-mode mode="STREAM" />
  </drools:configuration>
 </drools:kbase>
</beans>
```

KnowledgeBase takes the following configurations: "advanced-process-rule-integration, multithread, mbeans, event-processing-mode, accumulate-functions, evaluators and assert-behavior".

From the the kbase reference ksessions can be created

Example 2.3. Knowlege Sessions

```
<drools:ksession id="ksession1" type="stateless" name="stateless1" kbase="kbase1" />
```

```
<drools:ksession id="ksession2" type="stateful" kbase="kbase1" />
```

Like KnowledgeBases Knowlege sessions can take a number of configurations, including "work-item-handlers, "keep-references", "clock-type", "jpa-persistence".

Example 2.4. Knowlege Sessions Configurations

```
<drools:ksession id="ksession1" type="stateful"  kbase="kbase1" >
  <drools:configuration>
    <drools:work-item-handlers>
      <drools:work-item-handler name="handlername" ref="handlerid" />
    </drools:work-item-handlers>
    <drools:keep-reference enabled="true" />
    <drools:clock-type type="REALTIME"  />
  </drools:configuration>
</drools:ksession>
```

StatefulKnowledgeSessions can be configured for JPA persistence

Example 2.5. JPA configuration for StatefulKnowledgeSessions

```
<bean id="ds" class="org.springframework.jdbc.datasource.DriverManagerDataSource">
  <property name="driverClassName" value="org.h2.Driver" />
  <property name="url" value="jdbc:h2:tcp://localhost/DroolsFlow" />
  <property name="username" value="sa" />
  <property name="password" value="" />
</bean>

<bean id="myEmf" class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean">
  <property name="dataSource" ref="ds" />
  <property name="persistenceUnitName" value="org.drools.persistence.jpa.local" />
</bean>

<bean id="txManager" class="org.springframework.orm.jpa.JpaTransactionManager">
  <property name="entityManagerFactory" ref="myEmf" />
</bean>

<drools:ksession id="jpaSingleSessionCommandService" type="stateful" kbase="kbase1">
  <drools:configuration>
    <drools:jpa-persistence>
      <drools:transaction-manager ref="txManager" />
```

```
    <drools:entity-manager-factory ref="myEmf" />
    <drools:variable-persisters>
                                                    <drools:persister for-
class="java.lang.Object" implementation="org.drools.persistence.processinstance.persisters.JPAVariablePersister"/
>
                                                    <drools:persister for-
class="java.lang.String" implementation="org.drools.container.spring.beans.persistence.StringVariablePersister"/
>
                                                    <drools:persister for-
class="java.io.Serializable" implementation="org.drools.persistence.processinstance.persisters.SerializableVariablePersister"/
>
    </drools:variable-persisters>
   </drools:jpa-persistence>
  </drools:configuration>
</drools:ksession>
```

Knowledge Sessions can support startup batch scripts, previous versions used the "script" element name, this will be updated to "batch". The following commands are supported: "insert-object", "set-global", "fire-all-rules", "fire-until-halt", "start-process", "signal-event". Anonymous beans or named "ref" attributes may be used.

Example 2.6. Startup Batch Commands

```
<drools:ksession id="jpaSingleSessionCommandService" type="stateful" kbase="kbase1">
  <drools:script>
    <drools:insert-object ref="person1" />
    <drools:start-process process-id="proc name">
      <drools:parameter identifier="varName" ref="varRef" />
    </drools:start-process>
    <drools:fire-all-rules />
  </drools:script>
</drools:ksession>
```

ExecutionNodes are supported in Spring , these provide a Context of registered ksessions; this can be used with Camel to provide ksession routing.

Example 2.7. Execution Nodes

```
<execution-node id="node1" />

<drools:ksession id="ksession1" type="stateless" name="stateless1" kbase="kbase1" node="node1"/
>
```

```
<drools:ksession id="ksession2" type="stateful" kbase="kbase1" node="node1"/>
```

2.1.2.3. Camel

Spring can be combined with Camel to provide declarative rule services. a Camel Policy is added from Drools which provides magic for injecting the ClassLoader used by the ksession for any data formatters, it also augments the Jaxb and XStream data formatters. In the case If Jaxb it adds additional Drools related path info and with XStream it registers Drools related converters and aliases.

You can create as many endpoints as you require, using different addresses. The CommandMessagBodyReader is needed to allow the payload to be handled by Camel.

Example 2.8. Rest Endpoint Configuration

```
<cxf:rsServer id="rsServer"
          address="/kservice/rest"
          serviceClass="org.drools.jax.rs.CommandExecutorImpl">
  <cxf:providers>
    <bean class="org.drools.jax.rs.CommandMessageBodyReader"/>
  </cxf:providers>
</cxf:rsServer>
```

Camel routes can then be attached to CXF endpoints, allowing you control over the payload for things like data formatting and executing against Drools ksessions. The DroolsPolicy adds some smarts to the route. If JAXB or XStream are used, it would inject custom paths and converters, it can also set the classloader too on the server side, based on the target ksession. On the client side it automatically unwrapes the Response object.

This example unmarshalls the payload using an augmented XStream DataFormat and executes it against the ksession1 instance. The "node" there refers to the ExecutionContext, which is a context of registered ksessions.

Example 2.9. Camel Route

```
<bean id="droolsPolicy" class="org.drools.camel.component.DroolsPolicy" />

<camelContext id="camel" xmlns="http://camel.apache.org/schema/spring">
  <route>
    <from uri="cxfrs://bean://rsServer"/>
    <policy ref="droolsPolicy">
      <unmarshal ref="xstream" />
```

```
        <to uri="drools://node/ksession1" />
        <marshal ref="xstream" />
      </policy>
    </route>
</camelContext>
```

The Drools endpoint "drools:node/ksession1" consists of the execution node name followed by a separator and optional knowledge session name. If the knowledge session is not specified the route will look at the "lookup" attribute on the incoming payload instace or in the head attribute "DroolsLookup" to find it.

2.1.2.4. Drools Server

Spring, Camel and CXF can be combined for declarative services, drools-server is a .war that combines these with some sample xml that works out of the box to get you started, acting like a sort of template. If you are using the war in JBoss container you'll need to add this component, http://camel.apache.org/camel-jboss.html. The war includes a test.jsp showing an echo like example to get you started. This example just executes a simple "echo" type application. It sends a message to the rule server that pre-appends the word "echo" to the front and sends it back. By default the message is "Hello World", different messages can be passed using the url parameter msg - test.jsp?msg="My Custom Message".

2.1.2.5. Knowledge Agent Incremental Change Support

The new version of the Knowledge Agent supports newInstance = false in its configuration (incremental change-set build).

When setting this property to false, the KnowledgeAgent uses the exisitng KnowledgeBase references apply the incremental changes. Now KnowledgeAgent's can process monitored resource modifications in an incremental way. Modified definitions are compiled and compared against the original version. According to definition's type, the behaves in different ways:

- Rules: For rules, the Agent searches for modifications in its attributes, LHS and RHS.
- Queries: queries are always replaced on kbase wether they are modified or not.
- Other definitions: All other definitions are always replaced in kbase (like if they were modified). We expect to add better support for definition's modification detection in further versions.

The current implementation only supports the deletion of rules, querries and functions definitions. Type declarations cannot be deleted.

2.1.2.6. Session Inspection and Reporting framework

A new API based framework for runtime session inspection and reporting was introduced, allowing for better data gathering during debugging or profiling of the application. This inspection framework

will become the basis of the tooling features to help providing more detailed information about the contents of each session. This api is experimental and not in drools-api for now, but feel free to play and help us improve it.

To inspect a session, one can use the following API calls:

Example 2.10. Creating a SessionInspector

```
StatefulKnowledgeSession ksession = ...

// ... insert facts, fire rules, etc

SessionInspector inspector = new SessionInspector( ksession );
StatefulKnowledgeSessionInfo info = inspector.getSessionInfo();
```

The StatefulKnowledgeSessionInfo instance will contain a lot of relevant data gathered during the analysis of the session. A simple example report template is provided and can be generated with the following API call:

Example 2.11. Generating a Report

```
String report = SessionReporter.generateReport( "simple", info, null );
```

2.1.3. Expert

2.1.3.1. Differential Update

Rete traditional does an update as a retract + assert, for a given fact this causes all partial matches to be destroyed, however during the assert some of which will be recreated again; because they were true before the update and still true after the update. This causes a lot of uneccessary object destruction and creation which puts more load on the Garbage Collector. Now an update is a single pass and inspects the partial matches in place avoiding the unecessary destruction of partial matches. It also removes the need to under go a normalisation process for events and truth maintenance; the normalisation process was where we would look at the activations retracted and activations inserted to figure out what was truly added and what was truly inserted to determine the "diff".

2.1.3.2. Channels

Exit Points have been replaced by the more aptly named channels, we felt this was more appropriate as they may be used by more than juse the rule engine and not an exact oppposte if Entry Points. Where entry points are explicitey related to entering a partition in the Rete network.

2.1.3.3. Live Querries

Drools has always had query support, but the result was returned as an iterable set; this makes it hard to monitor changes over time.

We have now complimented this with Live Querries, which has a listener attached instead of returning an iterable result set. These live querries stay open creating a view and publish change events for the contents of this view. So now you can execute your query, with parameters and listen to changes in the resulting view.

Example 2.12. Implementing ViewChangedEventListener

```
final List updated = new ArrayList();
final List removed = new ArrayList();
final List added = new ArrayList();

ViewChangedEventListener listener = new ViewChangedEventListener() {
 public void rowUpdated(Row row) {
  updated.add( row.get( "$price" ) );
 }

 public void rowRemoved(Row row) {
  removed.add( row.get( "$price" ) );
 }

 public void rowAdded(Row row) {
  added.add( row.get( "$price" ) );
 }
};

// Open the LiveQuery
LiveQuery query = ksession.openLiveQuery( "cheeses",
                                  new Object[] { "cheddar", "stilton" },
                                  listener );
...
...
query.dispose() // make sure you call dispose when you want the query to close
```

A Drools blog article contains an example of Glazed Lists integration for live queries,

http://blog.athico.com/2010/07/glazed-lists-examples-for-drools-live.html

2.1.3.4. Timers and Calendars

Rule's now suport both interval and cron based timers, which replace the now deprecated duration attribute.

Example 2.13. Sample timer attribute uses

```
timer ( int: <initial delay> <repeat interval>? )
timer ( int: 30s )
timer ( int: 30s 5m )

timer ( cron: <cron expression> )
timer ( cron:* 0/15 * * * ? )
```

Interval "int:" timers follow the JDK semantics for initial delay optionally followed by a repeat interval. Cron "cron:" timers follow standard cron expressions:

Example 2.14. A Cron Example

```
rule "Send SMS every 15 minutes"
   timer (cron:* 0/15 * * * ?)
when
   $a : Alarm( on == true )
then
   channels[ "sms" ].insert( new Sms( $a.mobileNumber, "The alarm is still on" );
end
```

Calendars can now controll when rules can fire. The Calendar api is modelled on *Quartz http://www.quartz-scheduler.org/* [http://www.quartz-scheduler.org/] :

Example 2.15. Adapting a Quartz Calendar

```
Calendar weekDayCal = QuartzHelper.quartzCalendarAdapter(org.quartz.Calendar quartzCal)
```

Calendars are registered with the StatefulKnowledgeSession:

Example 2.16. Registering a Calendar

```
ksession.getCalendars().set( "week day", weekDayCal );
```

They can be used in conjunction with normal rules and rules including timers. The rule calendar attribute can have one or more comma calendar names.

Example 2.17. Using Calendars and Timers together

```
rule "weekdays are high priority"
  calendars "weekday"
  timer (int:0 1h)
when
   Alarm()
then
   send( "priority high - we have an alarm# );
end

rule "weekend are low priority"
  calendars "weekend"
  timer (int:0 4h)
when
   Alarm()
then
   send( "priority low - we have an alarm# );
end
```

2.1.3.5. Decision Tables (Excel)

2.1.3.5.1. Simple templating for variable length comma separated lists within cells

It is now possible to have a comma separated list of values in a cell and render those with a forall template

Example 2.18. DTable forall syntax

```
forall(<separator>?){<codesnippt>}
```

Example 2.19. DTable forall examples

```
forall(,) {propertyName == $}
forall(&&) {propertyName == $}
forall(||) {propertyName == $}
forall(||) {propertyNameA == $} && forall(||){propertyNameB == $}
```

etc

2.1.4. Flow

2.1.4.1. BPMN2

As we already announced earlier, the Drools team has decided to support the use of the upcoming BPMN 2.0 specification for specifying business processes using XML. This milestone includes a significant extension of the BPMN2 parser to support more of the BPMN2 features using Drools Flow. More specifically:

- more extensive event support: much more combinations of event types (start, intermediate and end) and event triggers (including for example error, escalation, timer, conditional and signal events), have been included, as well as (interrupting and non-interrupting) boundary events
- sub-process parameters
- diverging inclusive gateway
- etc.

BPMN2 processes have also been integrated in the entire Drools tool chain, to support the entire life cycle of the business process. This includes

- The ability to use BPMN2 processes in combination with our Eclipse tooling
- Guvnor as process repository
- web-based management using the BPM console
- auditing and debugging
- domain-specific processes
- etc.

As a result, Drools Flow is not only the first open-source process engine that supports such a significant set of BPMN2 constructs natively, our knowledge-oriented approach also allows you to easily combine your BPMN2 processes with business rules and complex event processing, all using the same APIs and tools.

2.1.4.2. Web-based Management console

Drools Flow processes can now also be managed through a web console. This includes features like managing your process instances (starting/stopping/inspecting), inspecting your (human) task list and executing those tasks, and generating reports.

This console is actually the (excellent!) work of Heiko Braun, who has created a generic BPM console that can be used to support multiple process languages. We have therefore implemented the necessary components to allow this console to communicate with the Drools Flow engine.

2.1.4.3. Pluggable Variable Persistence

Drools Flow can persist the runtime state of the running processes to a database (so they don't all need to be in memory and can be restored in case of failure). Our default persistence mechanism stores all the runtime information related to one process instance as a binary object (with associated metadata). The data associated with this process instance (aka process instance variables) were also stored as part of that binary object. This however could generate problem (1) when the data was not Serializable, (2) when the objects were too large to persist as part of the process instance state or (3) when they were already persisted elsewhere. We have therefor implemented pluggable variable persisters where the user can define how variable values are stored. This for example allows you to store variable values separately, and does support JPA entities to be stored separately and referenced (avoiding duplication of state).

2.1.4.4. Improved Process Instance Migration

Over time, processes may evolve. Whenever a process is updated, it is important to determine what should happen to the already running process instances. We have improved our support for migrating running process instances to a newer version of the process definition. Check out the Drools Flow documentation for more information.

2.1.4.5. Installation Script

The Drools build now exports an installer that simplifies installing the Eclipse plugin, Guvnor and the gwt-console. It creates and copies the necessary jars and wars and deploys them to the JBoss AS. It also includes a simple evaluation process example you can use to test your setup. For more info, download the drools installer and take a look at the readme within.

2.1.5. Guvnor

Appearance has been cleaned, for example less pop ups. Reminders for save after changes in assets and information about actions that were taken, also better error reporting if something goes wrong.

2.1.5.1. Discussions

The comments are below the "documentation" section (and of course optional) (and there is an Atom feed to them).

Figure 2.1. Realtime Discussions

A "backchannel" type connection that is kept open from the browser to allow messages to push back - this means (when enabled) that messages show up in real time (and other handy things like if something is added to a list - the list is updated).

2.1.5.2. Inbox

The inbox feature provides the ability to track what you have opened, or edited - this shows up under an "Inbox" item in the main navigator. *http://blog.athico.com/2009/09/inbox-feature-to-track-recent-changes.html*

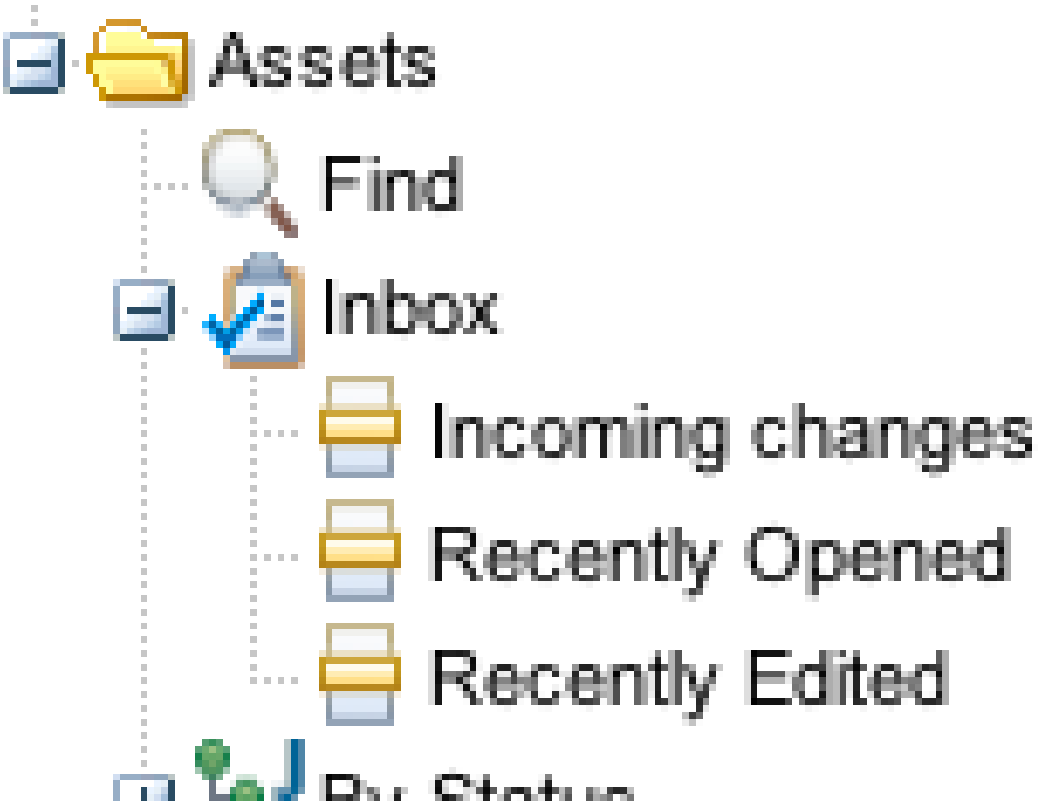


Figure 2.2. Inbox Categories

- **Recently Opened**
  - Clicking on the recently opened item will open a listing of all items you have "recently" opened (it tracks a few hundred items that you were last to look at).
- **Recently Edited**
  - Any items that you save changes to, or comment on will show up here, once again.
- **Incoming changes**
  - This tracks changes made by *other people* to items that are in *your* "Recently Edited" list. When you open these items they then are removed from this list (but remain in your Recently Edited list).

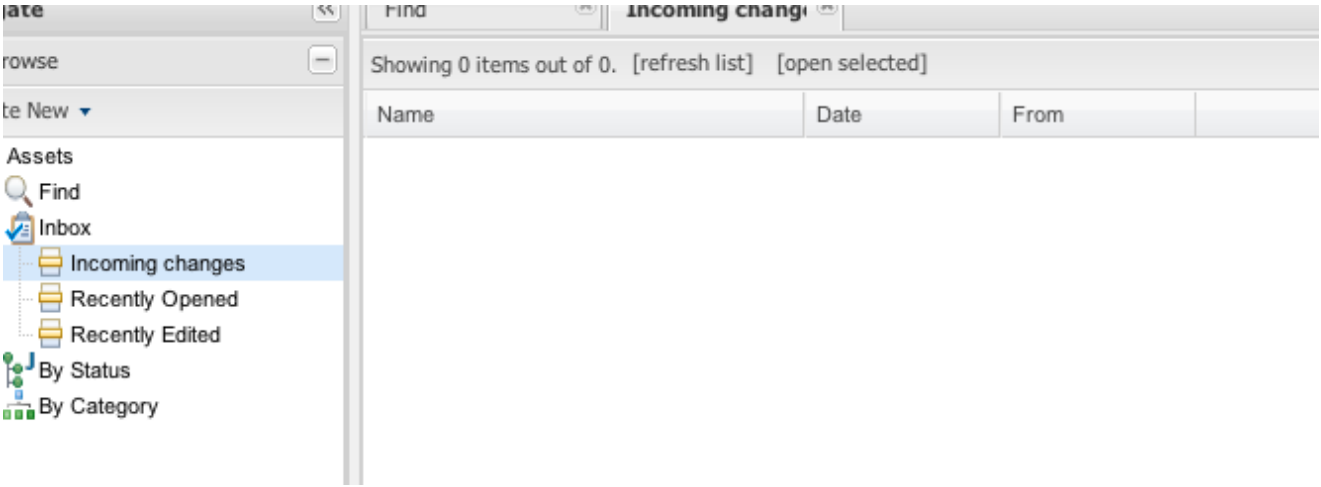


Figure 2.3. Inbox Item Lists

2.1.5.3. Bulk Importer

The Guvnor-Importer is a maven build tool that recurses your rules directory structure and constructs an xml import file that can be manually imported into the Drools-Guvnor web interface via the import/export administration feature.

2.1.5.4. DroolsDoc

PDF document containing information about the package and each DRL asset. DroolsDoc for knowledge package can be downloaded from package view under "Information and important URLs"

2.1.5.5. Update to GWT 2.0

GWT was updated, making Guvnor faster.

2.1.5.6. Build in Selector

The built in selector allows user to choose what assets to build according to:

- Status (eg, Dev, QA etc)
- Category
- Metadata

2.1.5.7. Single Asset Verification

It is possible to verify just the asset you are working on (ruleflow, rule, decision table). Verification finds issues like conflicting restrictions in a rule or redundant rows in decision tables.

2.1.5.8. Global Area

Assets stored in Global area can be shared to all packages.

2.1.5.9. Diff Check between Snapshots

Lists the changes between two snapshots.

2.1.5.10. View Multiple Assets in One Tab

Makes it possible to open more than one asset into one view. All the assets can be saved and edited as a group.

2.1.5.11. From/Collect/Accumulate support

Guided Editor has basic support for From, Accumulate and Collect Patterns. You can add any of these structures as regular Patterns. New expression builder component was created to add support for nested method calls of a variable. Using the “plus” button at the top of rule’s WHEN section or using the new “Add after” button present in every Pattern will open the popup to add new conditional elements to your rule. In the list of possible elements you will find three new entries: ”From”, “From Accumulate” and “From Collect”.

When you add a new “From” element, you will see something like the image below in the guided editor. The left pattern of the “From” conditional element is a regular Pattern. You can add there any type of conditional element you want.The right section of the “From” pattern is an expression builder.

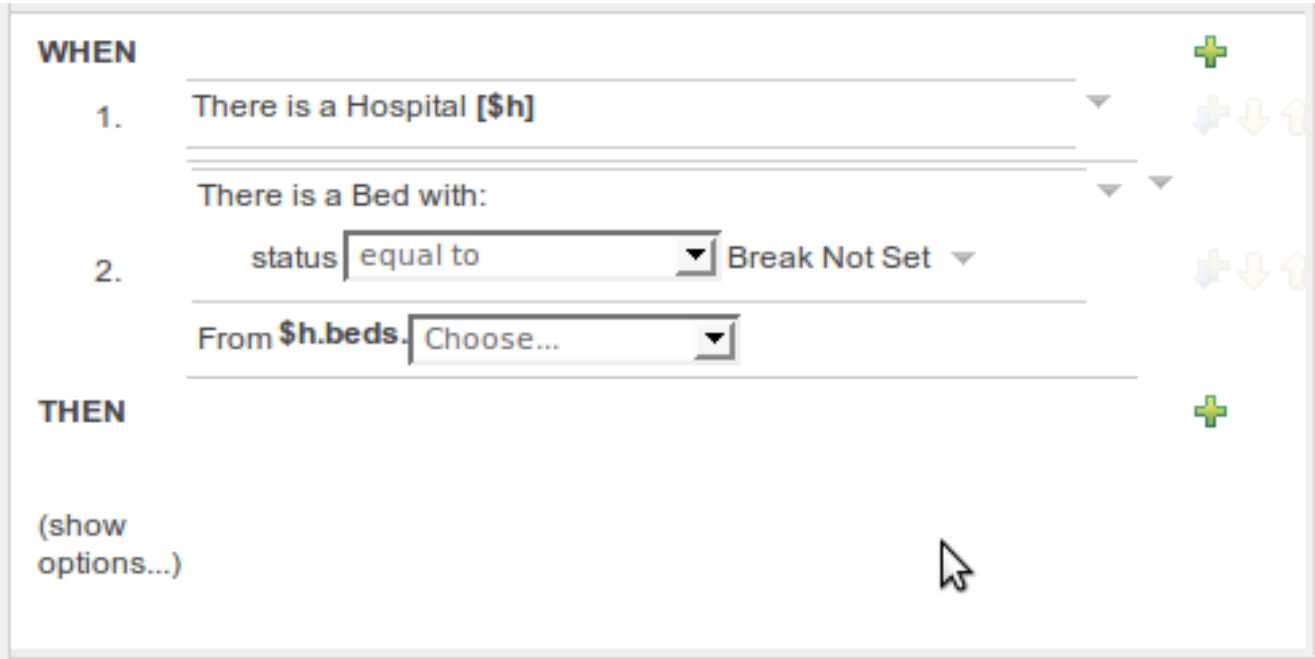


Figure 2.4. From CE Builder

When using 'from collect' In the left pattern you can choose from “java.util.Collection”, “java.util.List” or “java.util.Set” Fact Types. This Fact Types will be automatically included in the package’s Fact Types list.

The right pattern of the collect conditional element could be one of this patterns:

- Fact Type Pattern
- Free Form Expression
- From Pattern
- From Collect Pattern
- From Accumulate Pattern

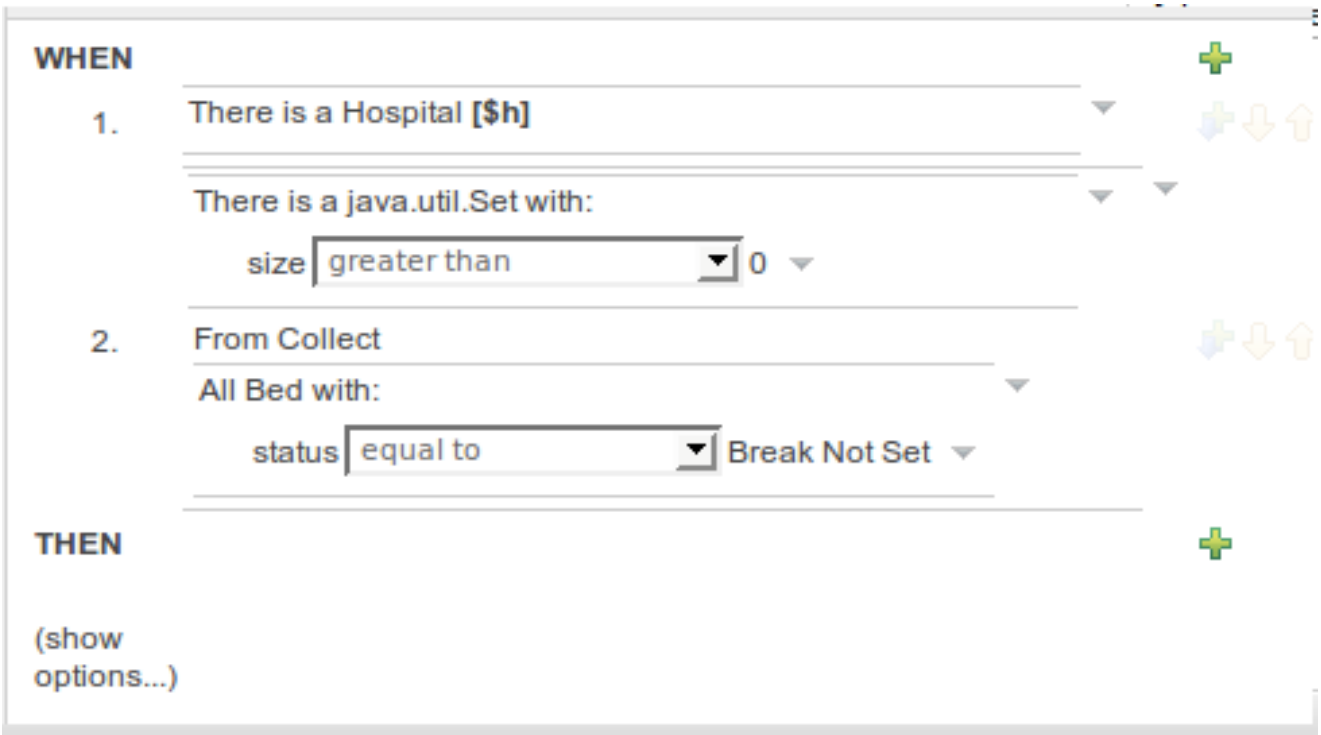


Figure 2.5. From Collect CE Builder

When using 'from accumulate' The left pattern could be any Fact Type Pattern. The right section of this conditional element is splited in two:

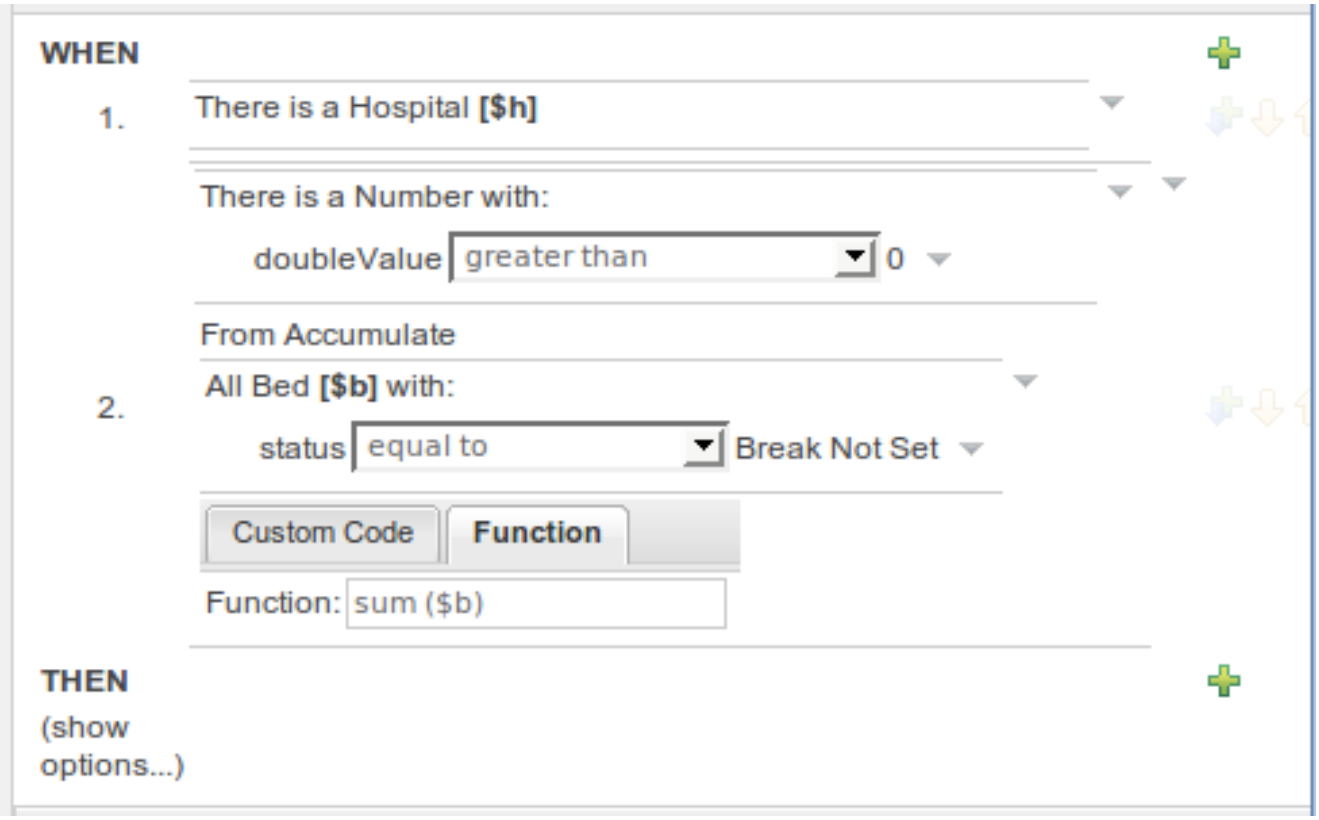


Figure 2.6. From Accumulate CE Builder

The left pattern could be any Fact Type Pattern. The right section of this conditional element is splited in two:

- Source Pattern: (Bed $n, in the screenshot) could be any Fact Type, From, Collect or Accumulate pattern.
- Accumulate function: Here you will find a tabbed panel where you can enter an accumulate function (sum() in the screenshot) or you can create an online custom function using the “Custom Code” tab.

2.1.5.12. Rule Templates

Rule Templates allow the Guided editor to be used to build complex rules that can then be authored easily through a spreadsheet's tabular data metaphor. Instead of a field's value, simply mark it as a named "Template Key" and that key is available as a column in the grid. Each row will be applied to the rule template to generate a rule.

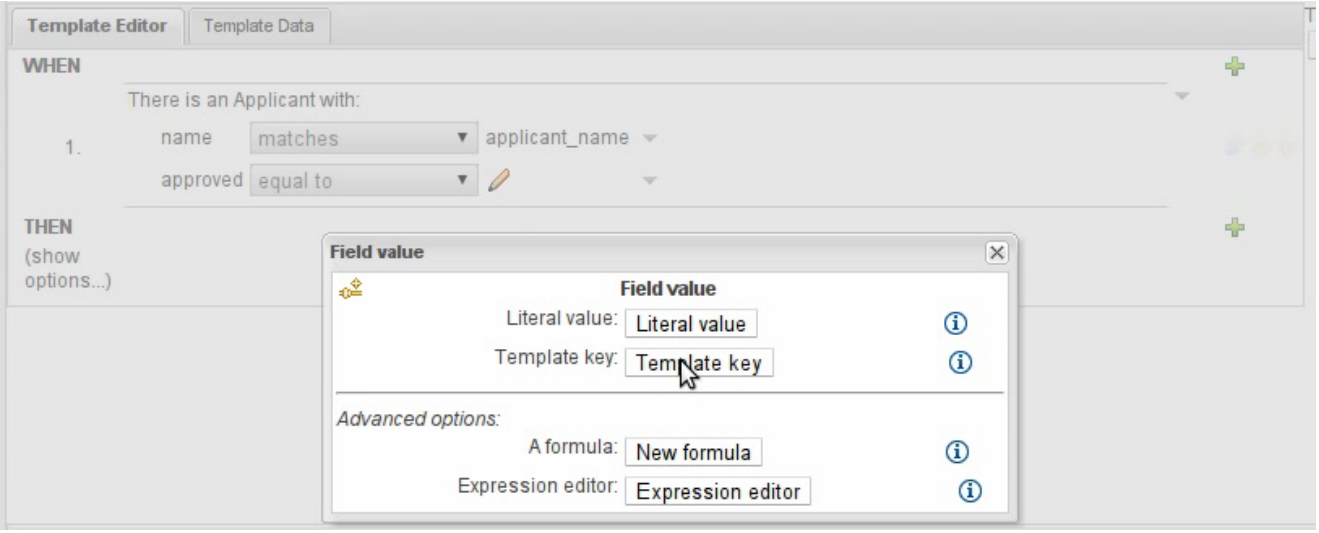


Figure 2.7. Adding a Template Key in the 'WHEN' Section

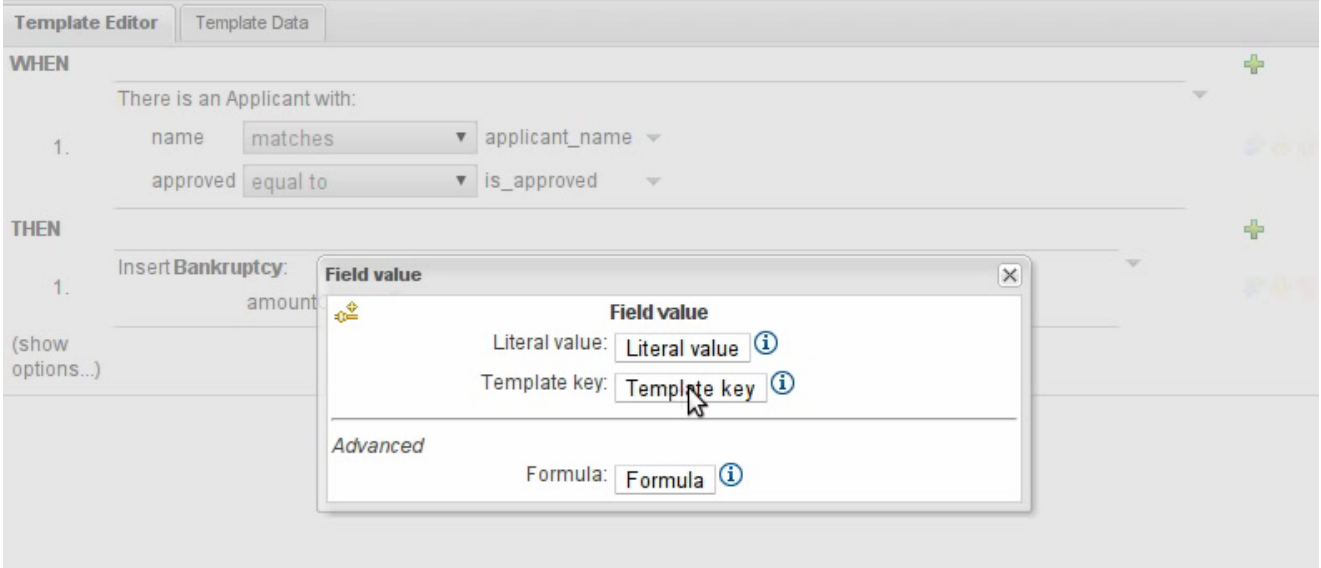


Figure 2.8. Adding a Template Key in the "THEN" Section

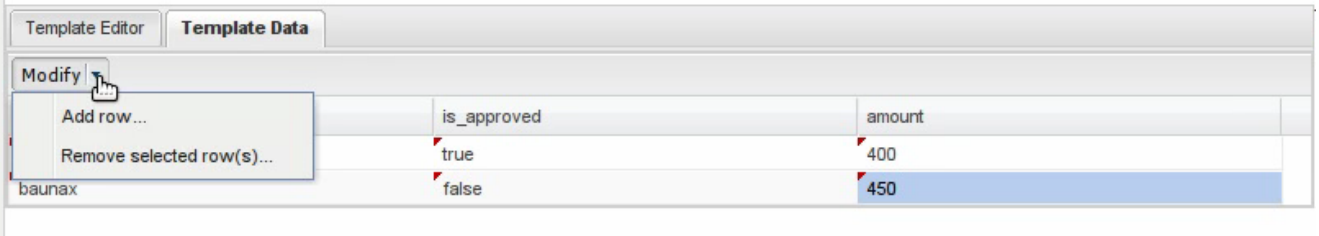


Figure 2.9. Populating Rows against those Template Keys

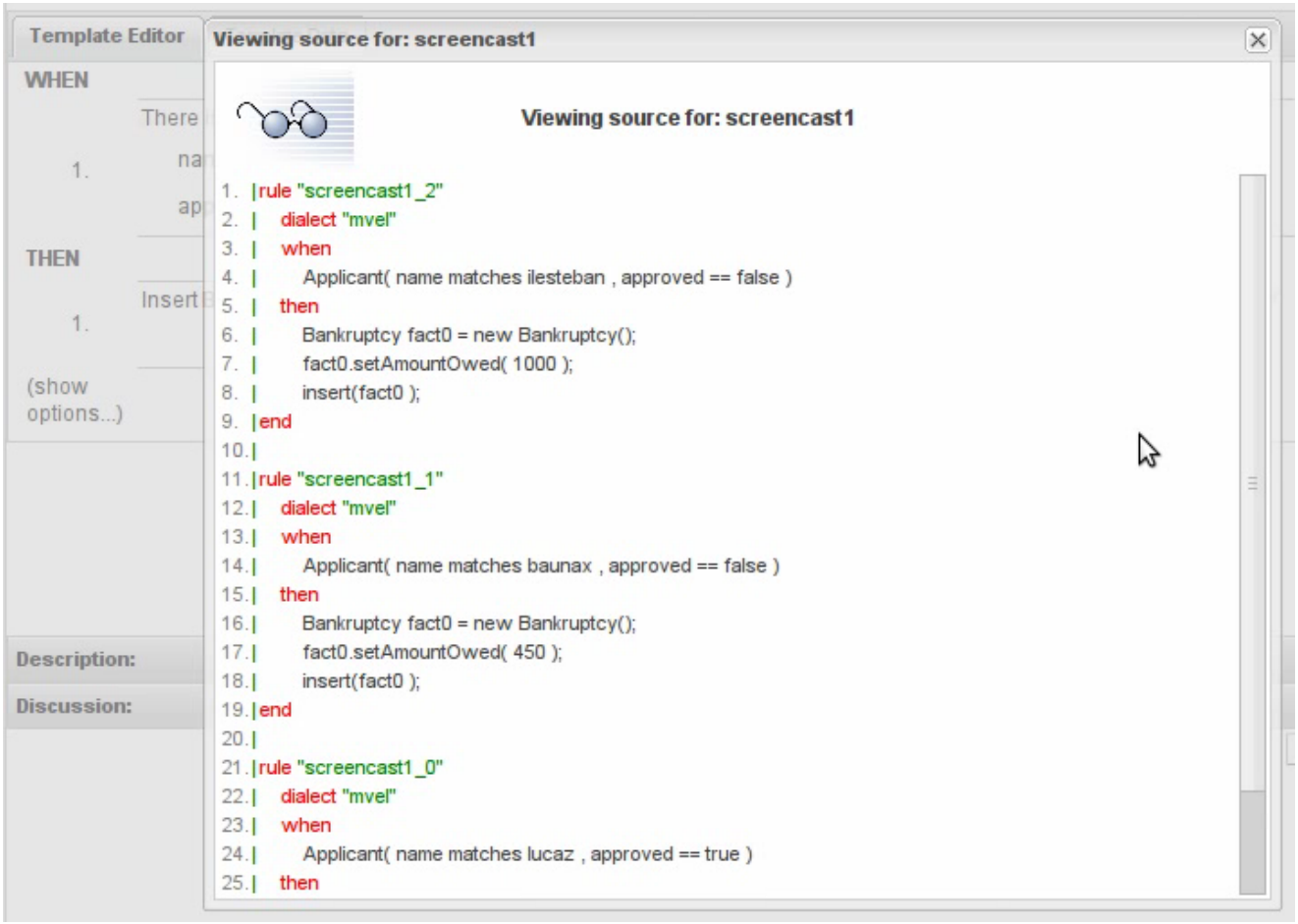


Figure 2.10. Each Row generates a Rule

2.1.5.13. Working Sets

When modelling rules the user gets exposed to all the fact types which can be a bit over whelming. Working Sets allow related fact types can be grouped together, provided a more managable view of selecting fact types, when authoring rules

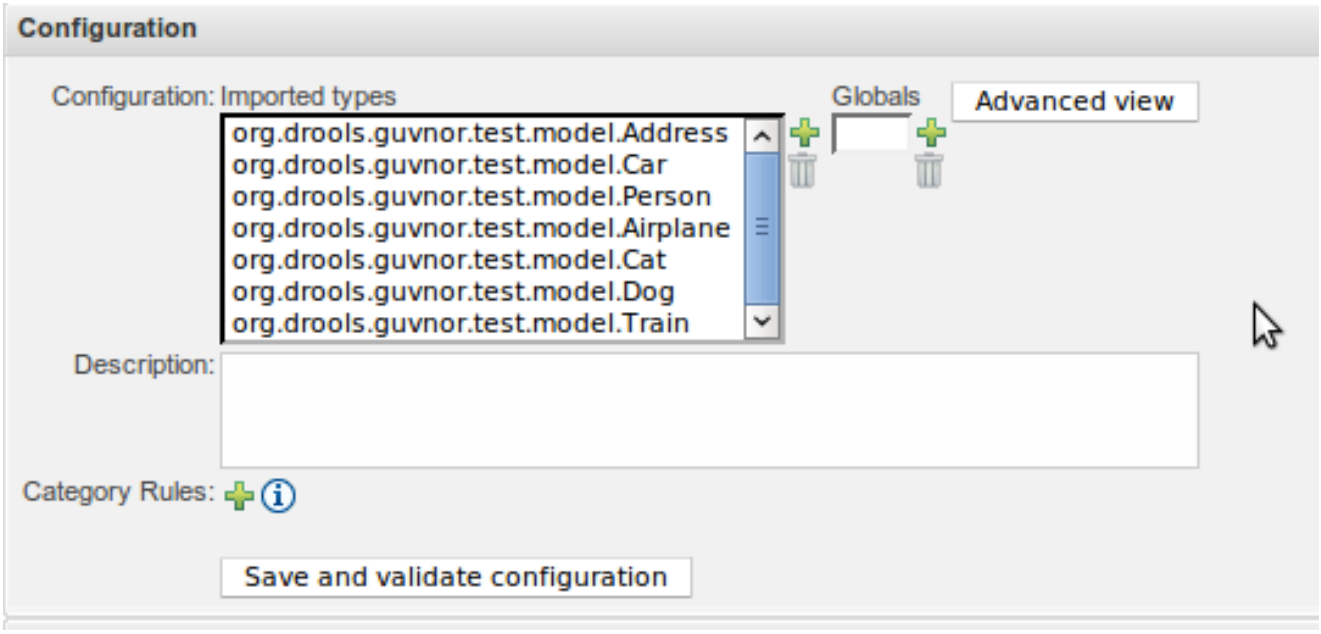


Figure 2.11. Show the imported types

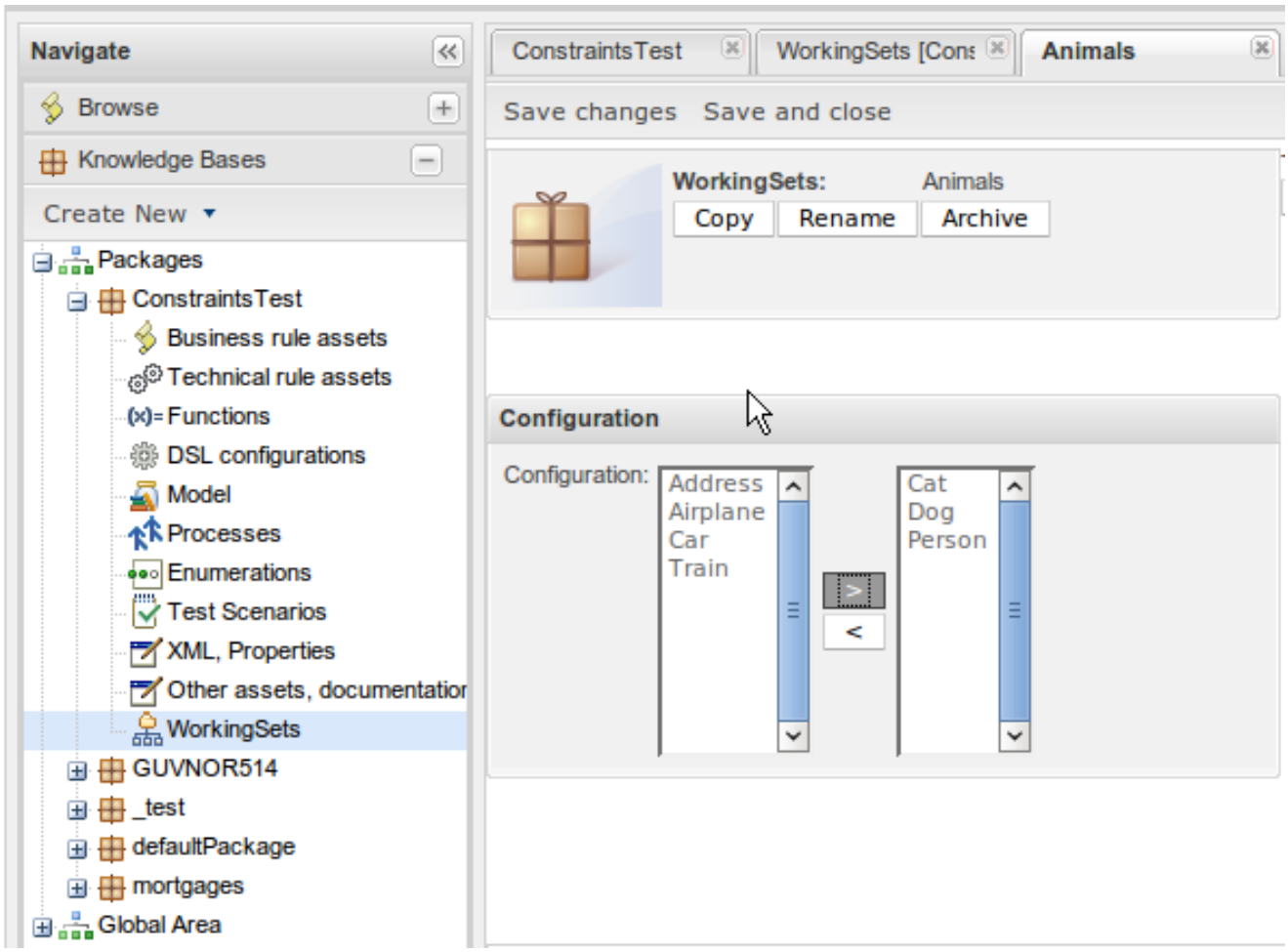


Figure 2.12. Create Animals Working Set

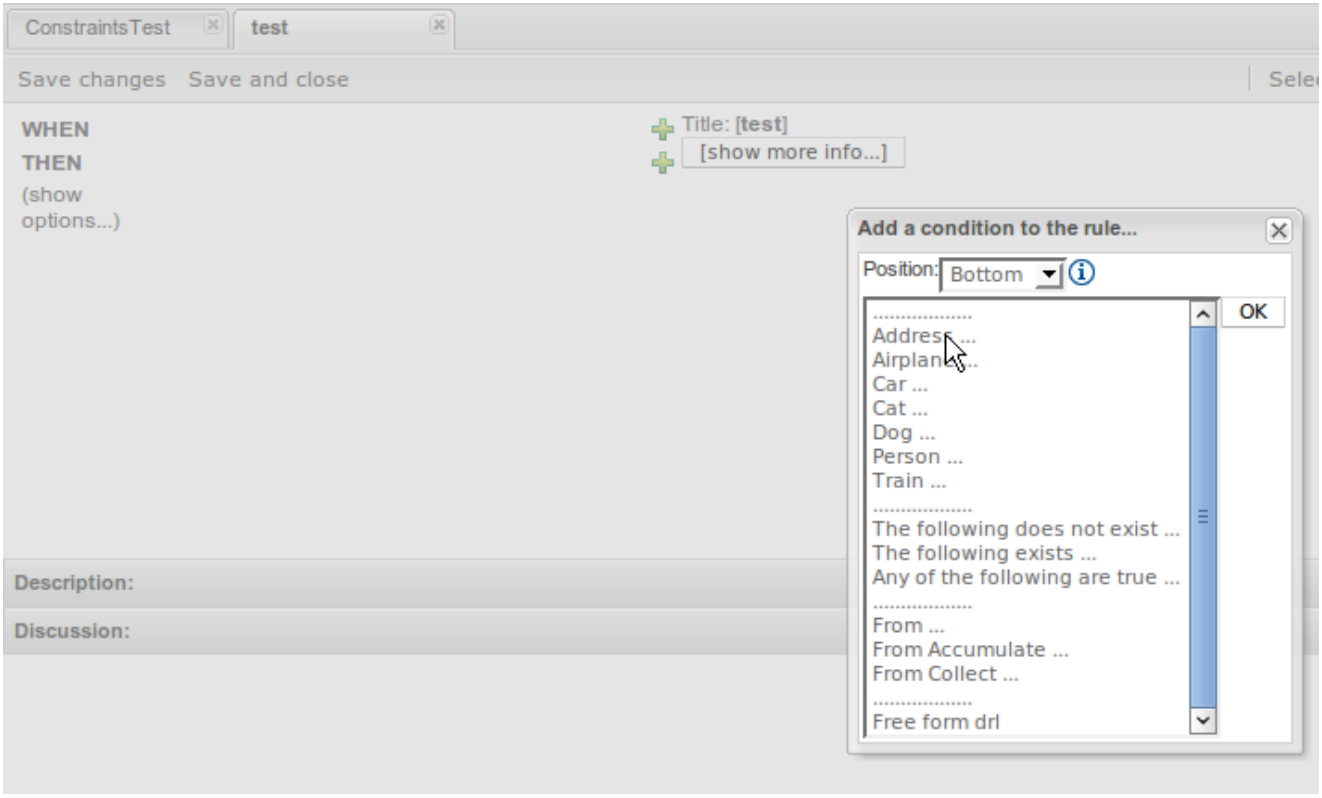


Figure 2.13. Without the Working Set all types are visible

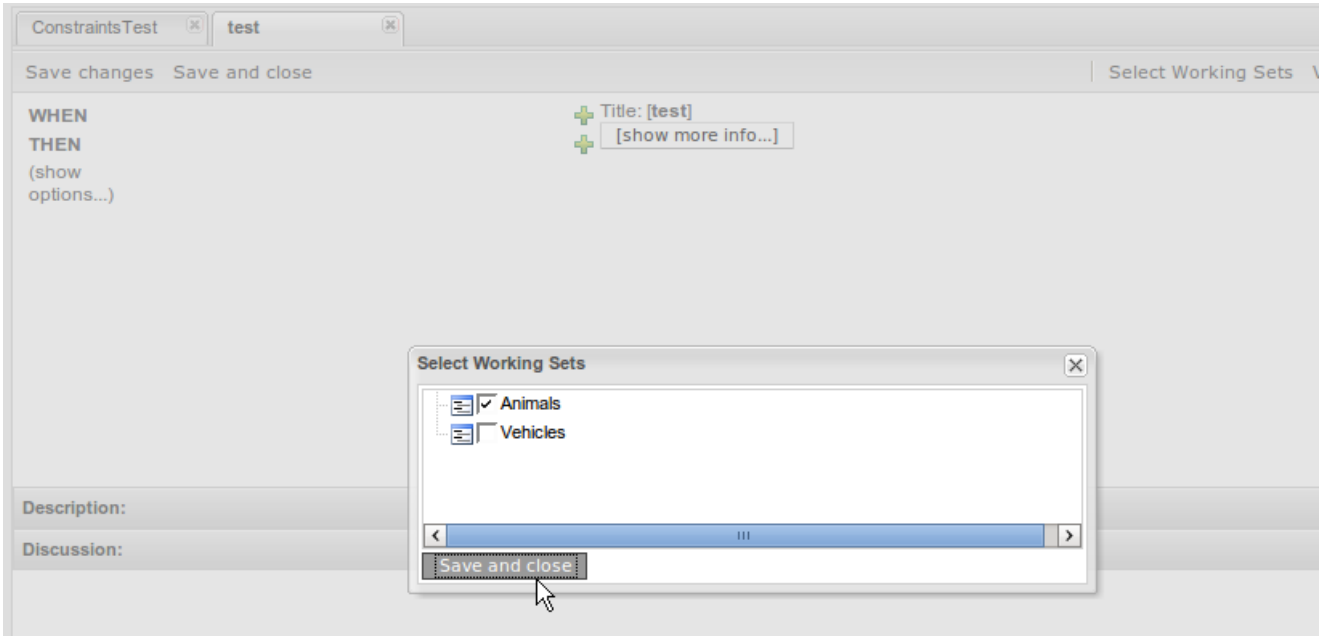


Figure 2.14. Select the Animals Working Set

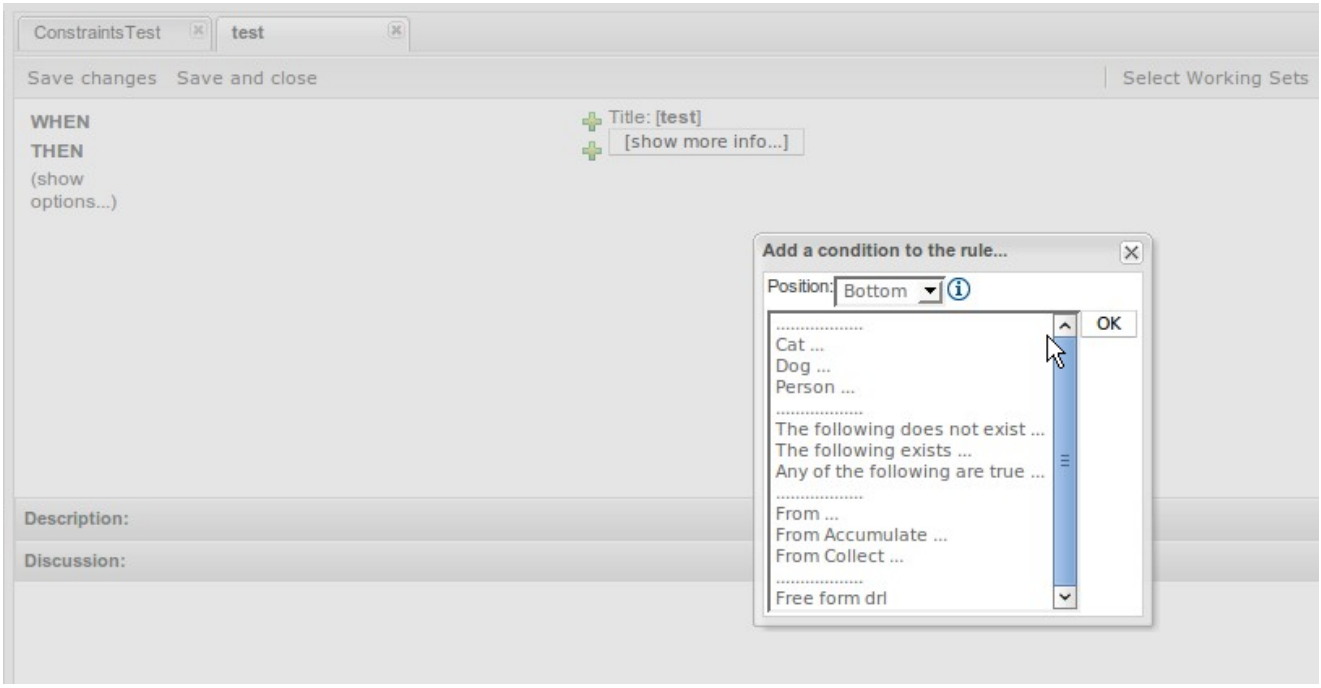


Figure 2.15. The available types are reduced

2.1.5.14. Fact Constraints

Working Sets can be combined with fact consraints to provide additional design time validation. For instance if you are authoring a rule on someone's age, we can know the valid ranges at design time and use this to constrain the author. The fact constraints are part of the workingset and when authoring a rule you must select the work set constraints that you wish to be applied as part of the validation process.

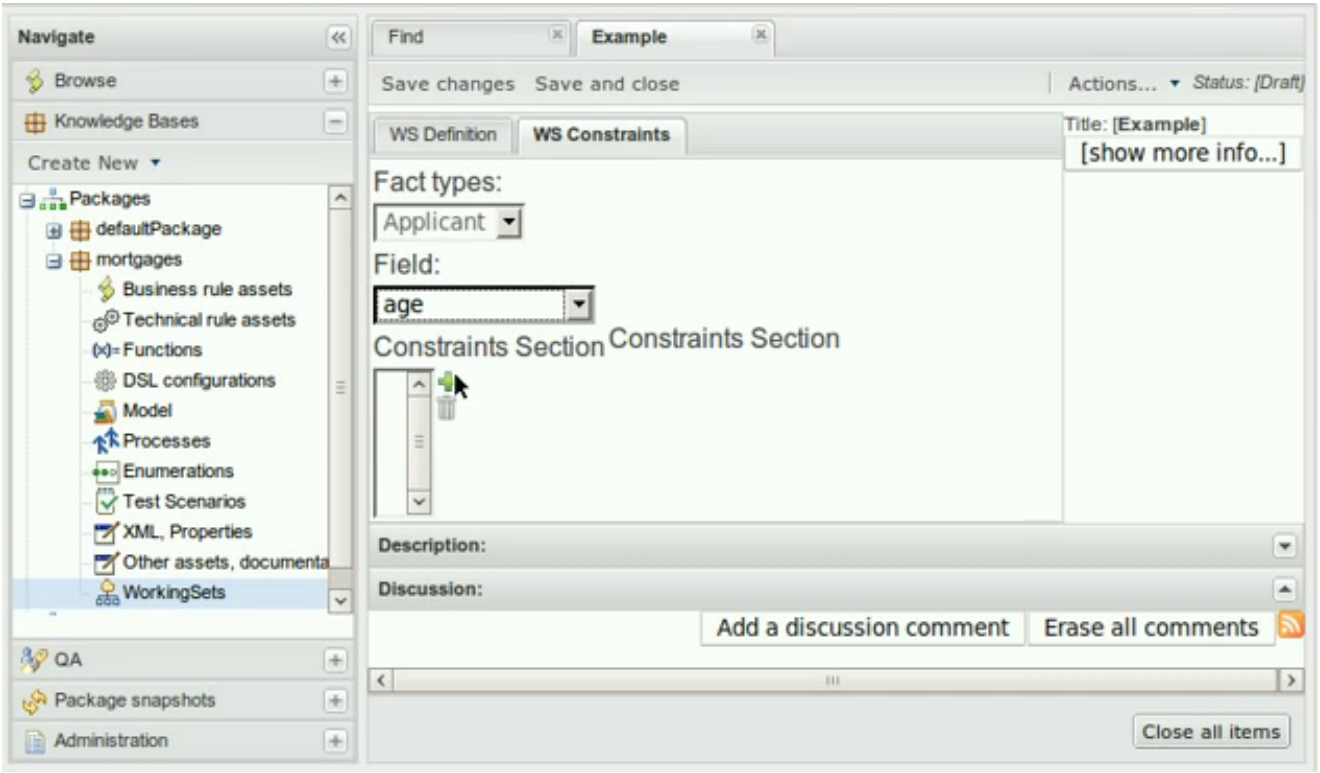


Figure 2.16. Selecting the field

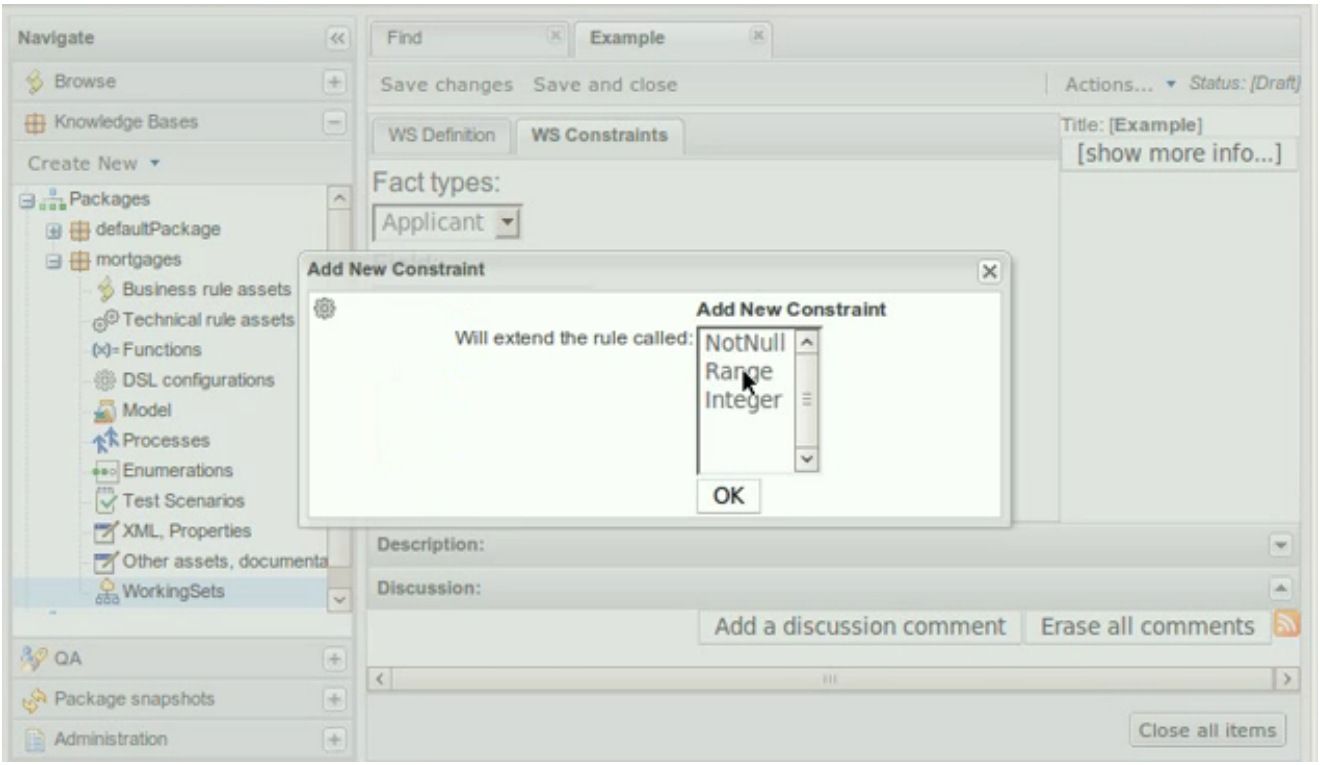


Figure 2.17. Selectng the type of field constraint

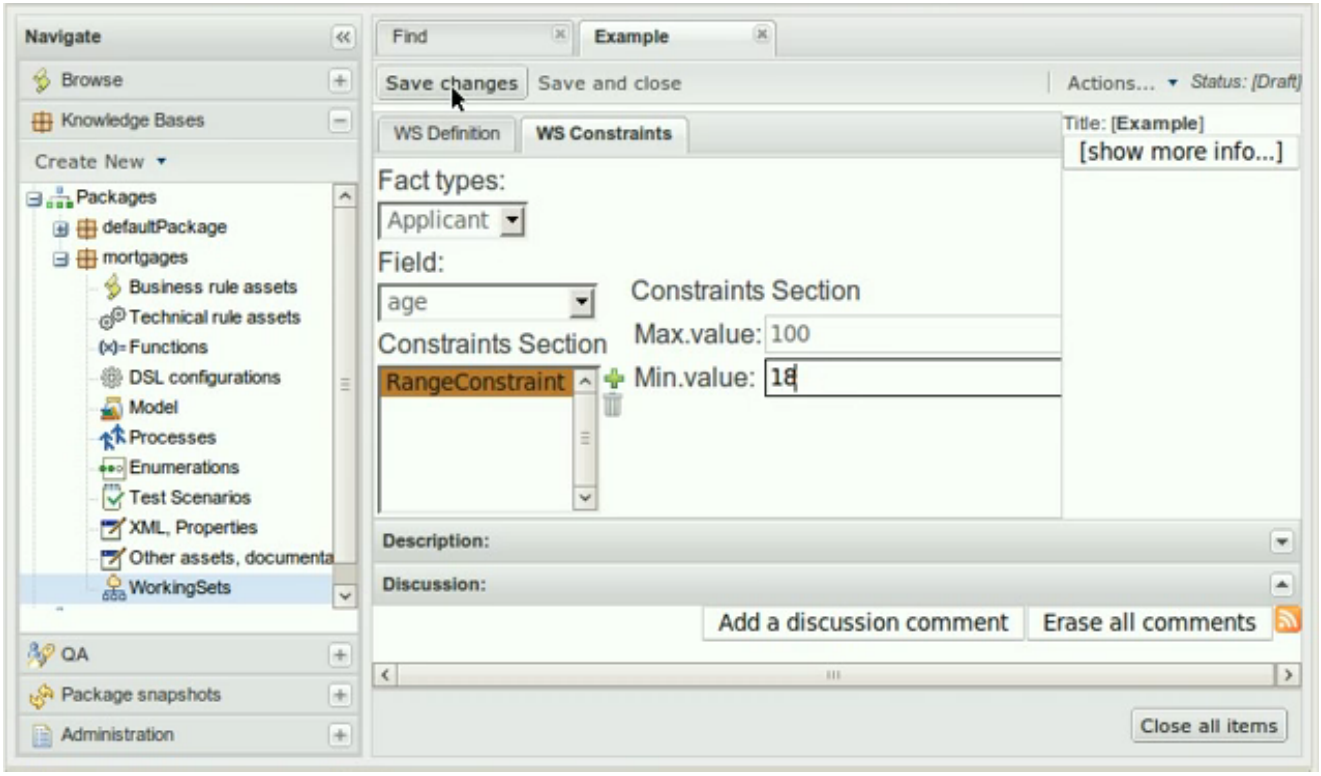


Figure 2.18. A example range constraint

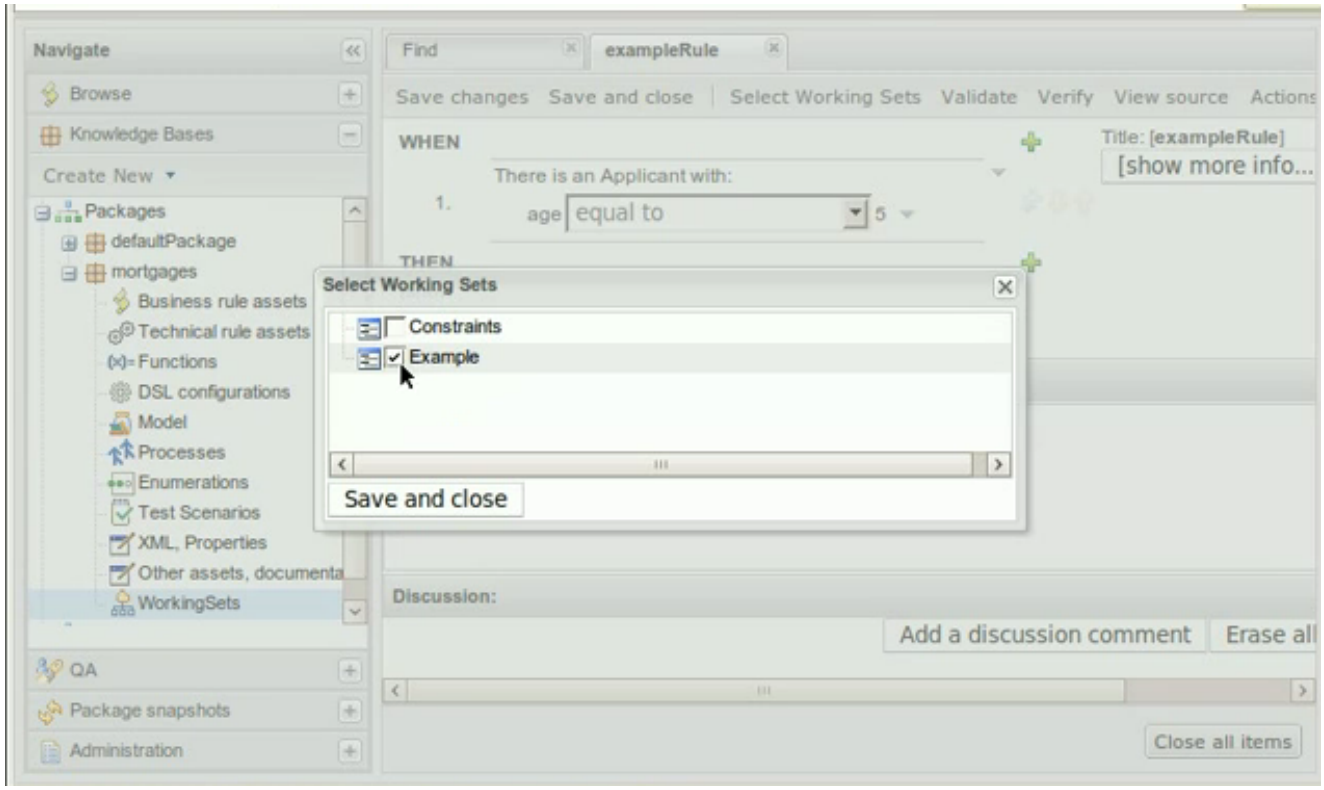


Figure 2.19. Select the working set to validate the age field

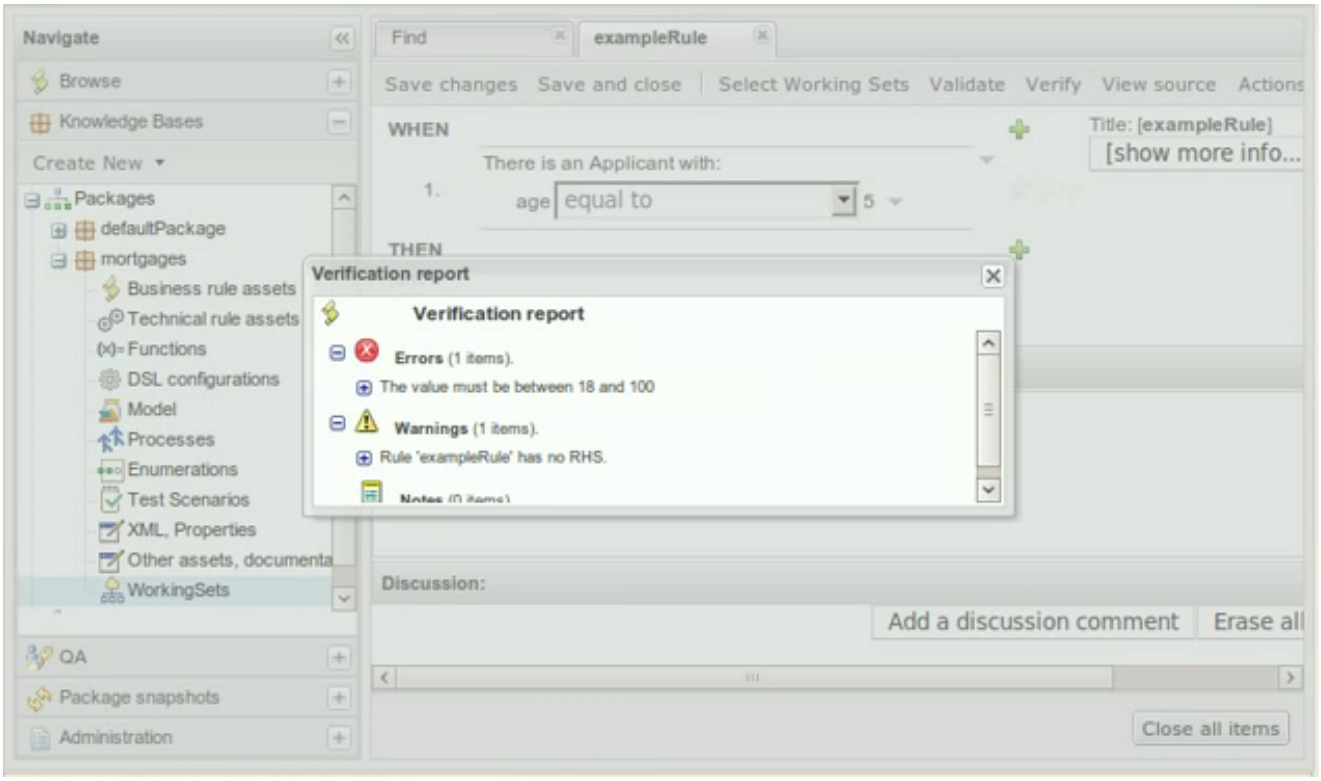


Figure 2.20. A age of 5 is invalid

2.1.5.15. Guided Rule Editor

Editing rules is made more explicit. Editor is less "boxy" and rules are written more as a normal text. "contains" keyword was added and binding variables to restrictions is now easier.

2.1.5.15.1. Pattern Order

Guided Editor supports Pattern reordering in LHS and RHS sections, as well as positional inserting, new Patterns can be inserted in any order (and not always at the end).

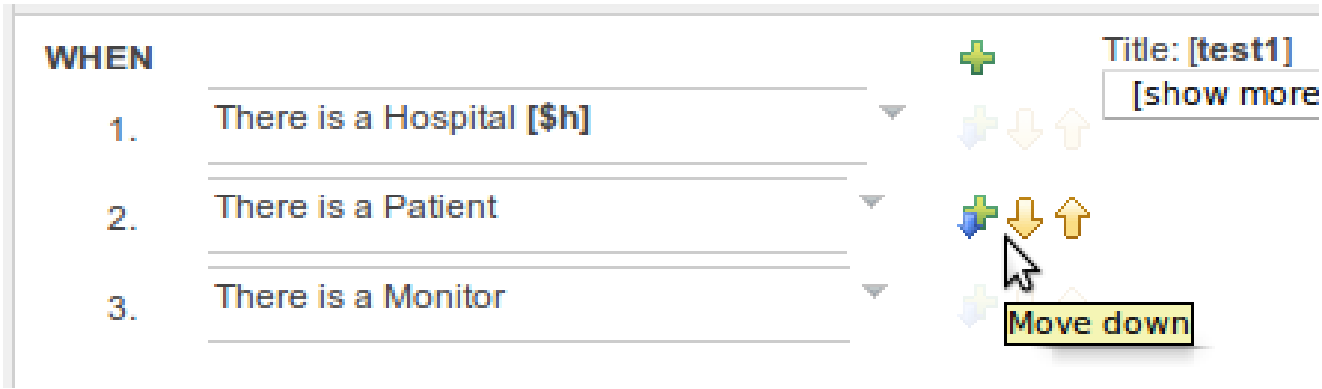


Figure 2.21. Move elements up or down

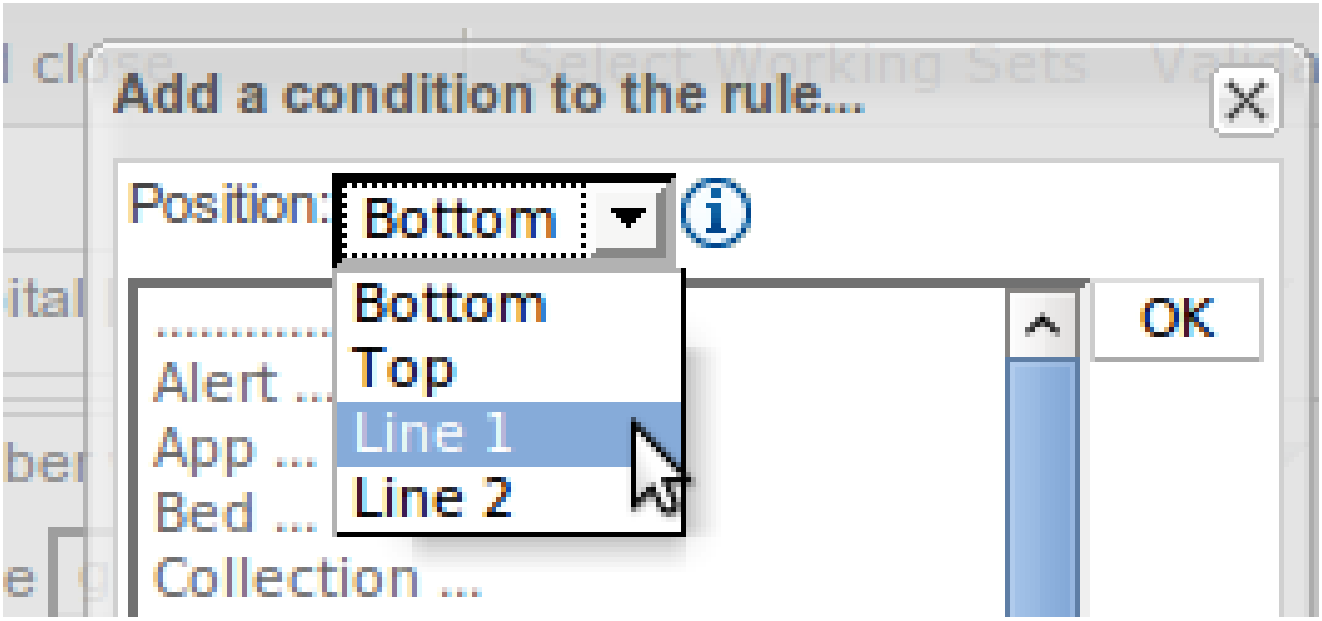


Figure 2.22. Insert Element at any position

2.1.5.16. Decision Table (Guvnor)

Keyword "in" was added.

Columns can be moved and location of a new row can be selected freely.

2.1.6. Eclipse

2.1.6.1. Group Rules in outline view

You can now use sorting and grouping for Agenda Groups.

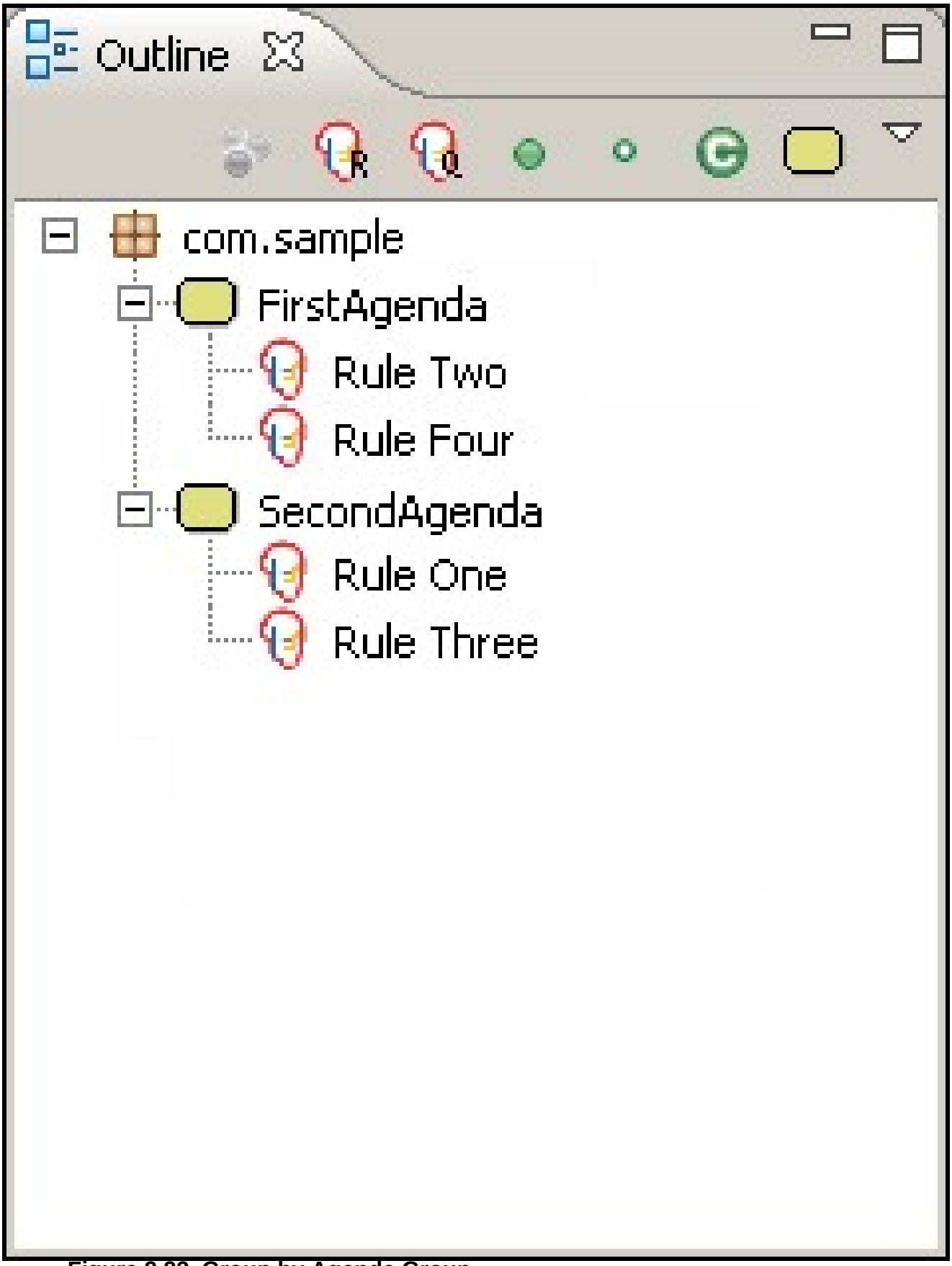


Figure 2.23. Group by Agenda Group

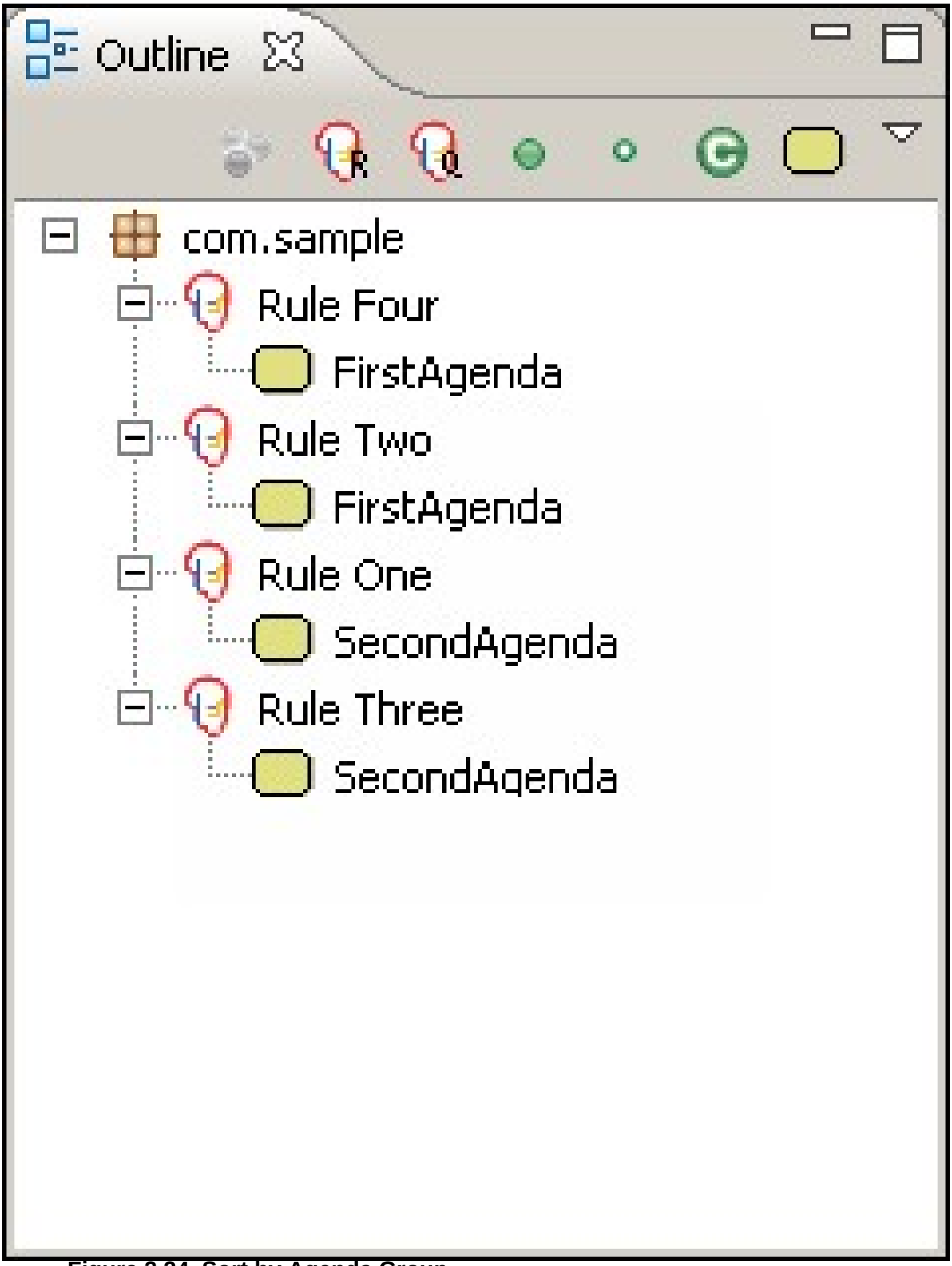


Figure 2.24. Sort by Agenda Group

2.1.6.2. Drag/Drop File support in audit View

It is now possible to drag and drop log files into the audit view.

2.1.7. Known Issues

2.1.7.1. Multithread mode

There is a known issue with the experimental multi-thread execution mode as described in the following JIRA.

https://jira.jboss.org/browse/JBRULES-2125 [???]

2.2. What is New and Noteworthy in Drools 5.0.0

2.2.1. Drools API

Drools now has complete api/implementation separation that is no longer rules oriented. This is an important strategy as we move to support other forms of logic, such as workflow and event processing. The main change is that we are now knowledge oriented, instead of rule oriented. The module drools-api provide the interfaces and factories and we have made pains to provide much better javadocs, with lots of code snippets, than we did before. Drools-api also helps clearly show what is intended as a user api and what is just an engine api, drools-core and drools-compiler did not make this clear enough. The most common interfaces you will use are:

- org.drools.builder.KnowledgeBuilder
- org.drools.KnowledgeBase
- org.drools.agent.KnowledgeAgent
- org.drools.runtime.StatefulKnowledgeSession
- org.drools.runtime.StatelessKnowledgeSession

Factory classes, with static methods, provide instances of the above interfaces. A pluggable provider approach is used to allow provider implementations to be wired up to the factories at runtime. The Factories you will most commonly used are:

- org.drools.builder.KnowledgeBuilderFactory
- org.drools.io.ResourceFactory
- org.drools.KnowledgeBaseFactory

- org.drools.agent.KnowledgeAgentFactory

Example 2.20. A Typical example to load a rule resource

```
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kbuilder.add( ResourceFactory.newUrlResource( url ),
          ResourceType.DRL );
if ( kbuilder.hasErrors() ) {
  System.err.println( builder.getErrors().toString() );
}

KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
kbase.addKnowledgePackages( builder.getKnowledgePackages() );

StatefulKnowledgeSession ksession = knowledgeBase.newStatefulKnowledgeSession();
ksession.insert( new Fibonacci( 10 ) );
ksession.fireAllRules();

ksession.dispose();
```

A Typical example to load a process resource. Notice the `ResourceType` is changed, in accordance with the `Resource` type:

Example 2.21. A Typical example to load a process resource. Notice the `ResourceType` is changed, in accordance with the `Resource` type

```
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kbuilder.add( ResourceFactory.newUrlResource( url ),
          ResourceType.DRF );
if ( kbuilder.hasErrors() ) {
  System.err.println( builder.getErrors().toString() );
}

KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
kbase.addKnowledgePackages( builder.getKnowledgePackages() );

StatefulKnowledgeSession ksession = knowledgeBase.newStatefulKnowledgeSession();
ksession.startProcess( "Buy Order Process" );

ksession.dispose();
```

'kbuilder', 'kbase', 'ksession' are the variable identifiers often used, the k prefix is for 'knowledge'.

Example 2.22. We have uniformed how decision trees are loaded, and they are now consistent with no need to pre generate the DRL with the spreadsheet compiler

```
DecisionTableConfiguration dtconf =
 KnowledgeBuilderFactory.newDecisionTableConfiguration();
dtconf.setInputType( DecisionTableInputType.XLS );
dtconf.setWorksheetName( "Tables_2" );
kbuilder.add( ResourceFactory.newUrlResource( "file://IntegrationExampleTest.xls" ),
        ResourceType.DTABLE,
        dtconf );
```

It is also possible to configure a `KnowledgeBase` using configuration, via a xml change set, instead of programmatically.

Example 2.23. Here is a simple change set

```
<change-set xmlns='http://drools.org/drools-5.0/change-set'
xmlns:xs='http://www.w3.org/2001/XMLSchema-instance'
xs:schemaLocation='http://drools.org/drools-5.0/change-set change-set-5.0.xsd' >
  <add>
    <resource source='classpath:org/domain/someRules.drl' type='DRL' />
    <resource source='classpath:org/domain/aFlow.drf' type='DRF' />
  </add>
</change-set>
```

Example 2.24. And it is added just like any other ResourceType

```
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
kbuilder.add( ResourceFactory.newUrlResource( url ),
        ResourceType.ChangeSet );
```

The other big change for the `KnowledgeAgent`, compared to the `RuleAgent`, is that polling scanner is now a service. further to this there is an abstraction between the agent notification and the resource monitoring, to allow other mechanisms to be used other than polling.

Example 2.25. These services currently are not started by default, to start them do the following

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

There are two new interfaces added, `ResourceChangeNotifier` and `ResourceChangeMonitor`. `KnowlegeAgents` subscribe for resource change notifications using the `ResourceChangeNotifier` implementation. The `ResourceChangeNotifier` is informed of resource changes by the added `ResourceChangeMonitors`. We currently only provide one out of the box monitor, `ResourceChangeScannerService`, which polls resources for changes. However the api is there for users to add their own monitors, and thus use a push based monitor such as JMS.

```
ResourceFactory.getResourceChangeNotifierService().addResourceChangeMonitor( myJmsMonitor);
```

2.2.2. Drools Guvnor

- New look web tooling

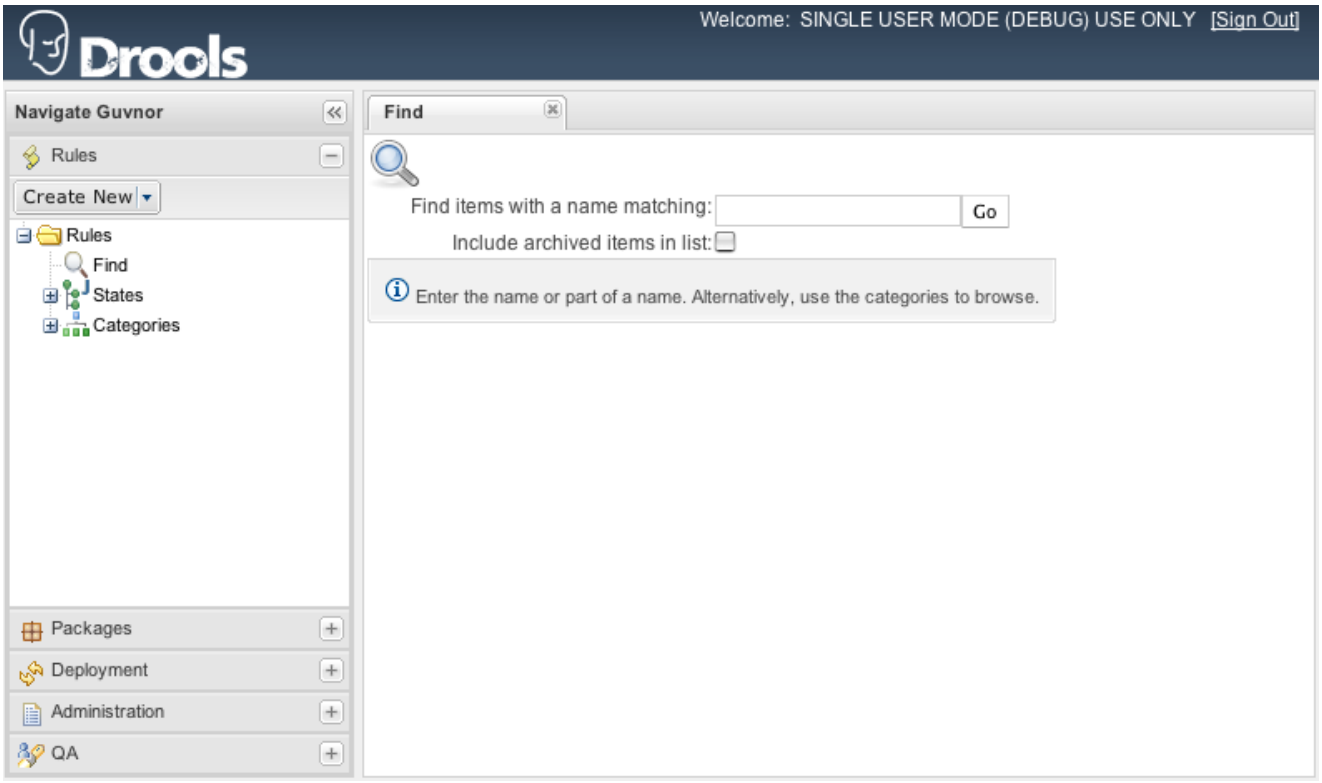


Figure 2.25. New Look

- Web based decision table editor

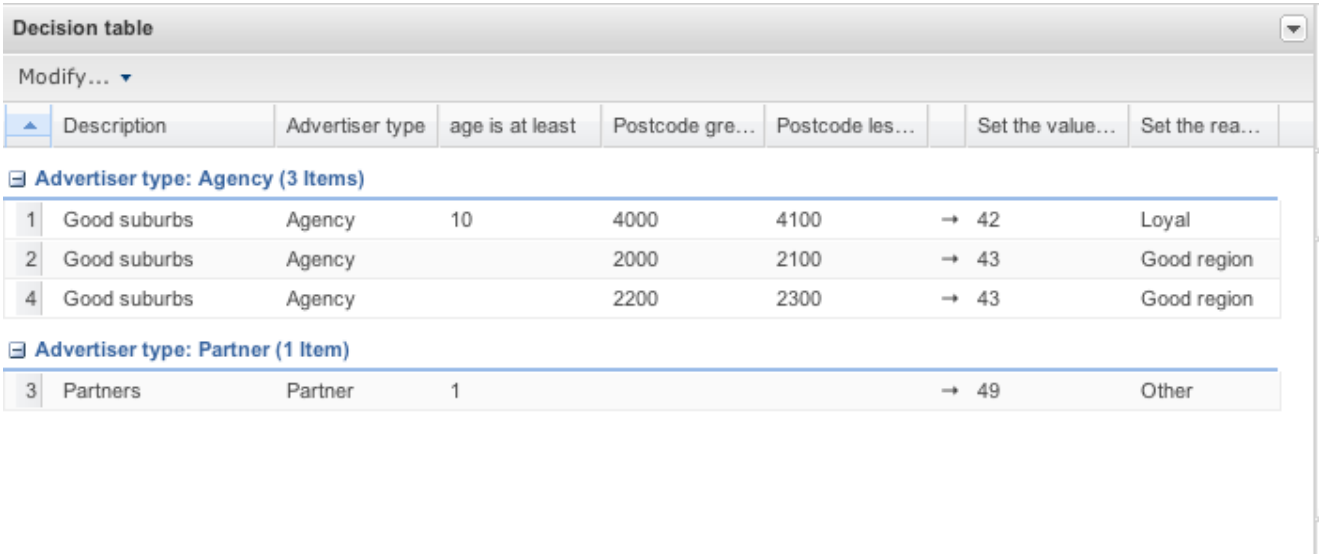


Figure 2.26. Web based decision table editor

- Integrated scenario testing

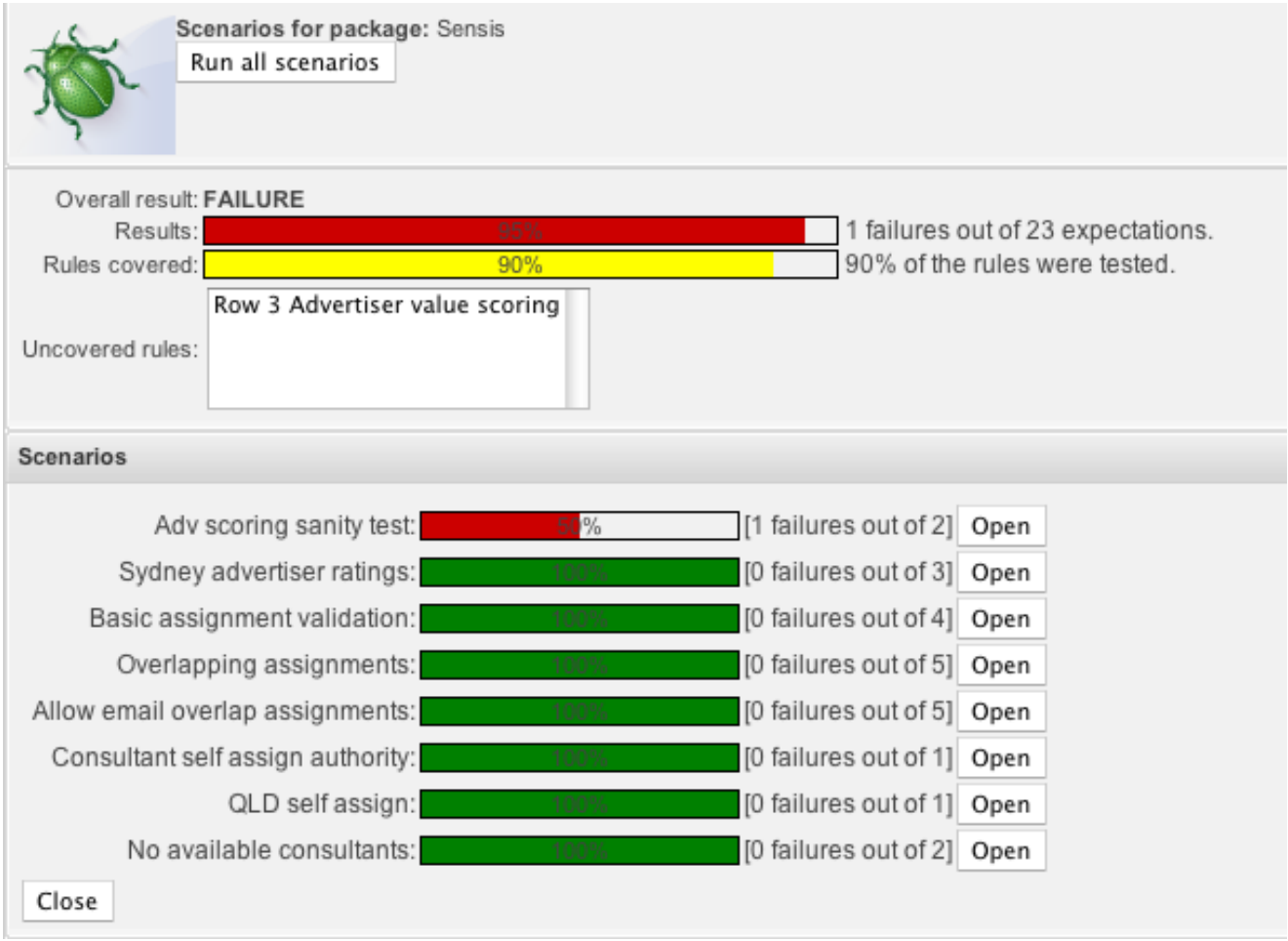


Figure 2.27. Running all scenarios

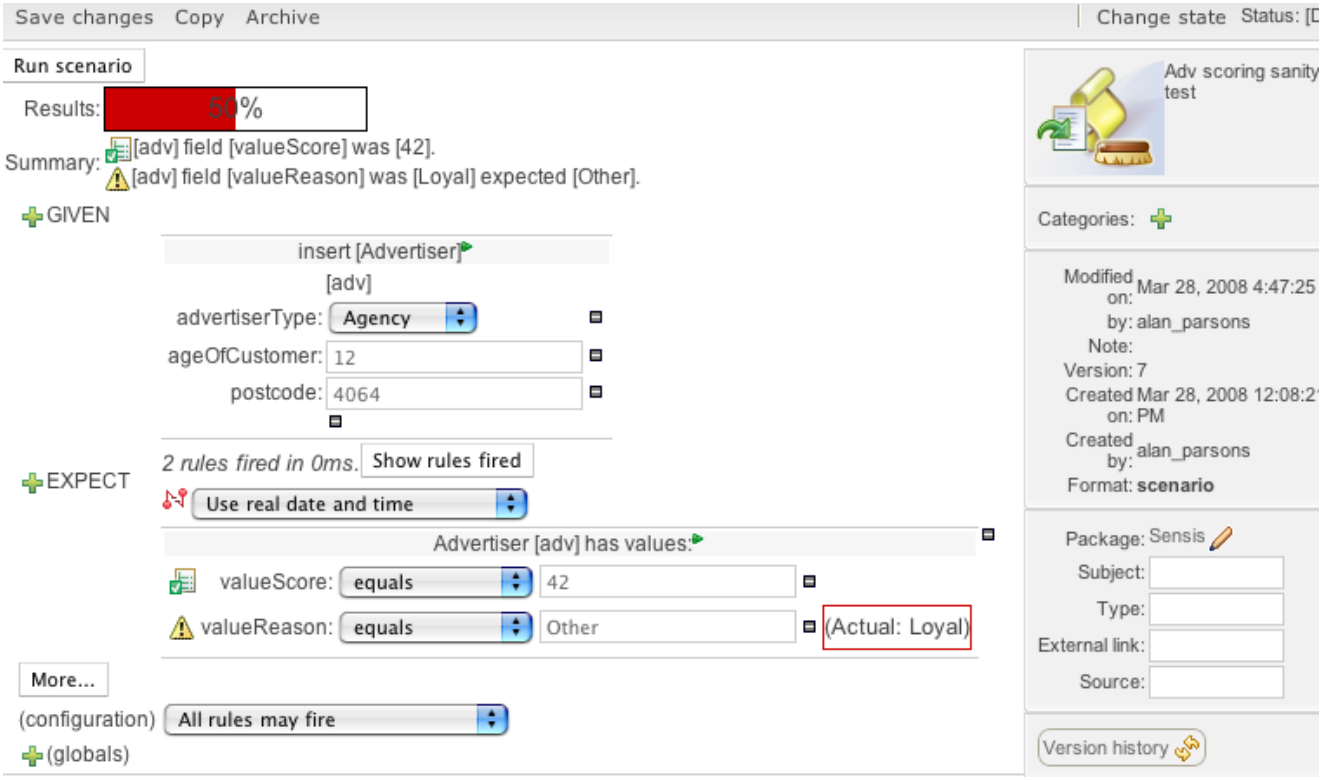


Figure 2.28. Running single scenario

- WebDAV file based interface to repository

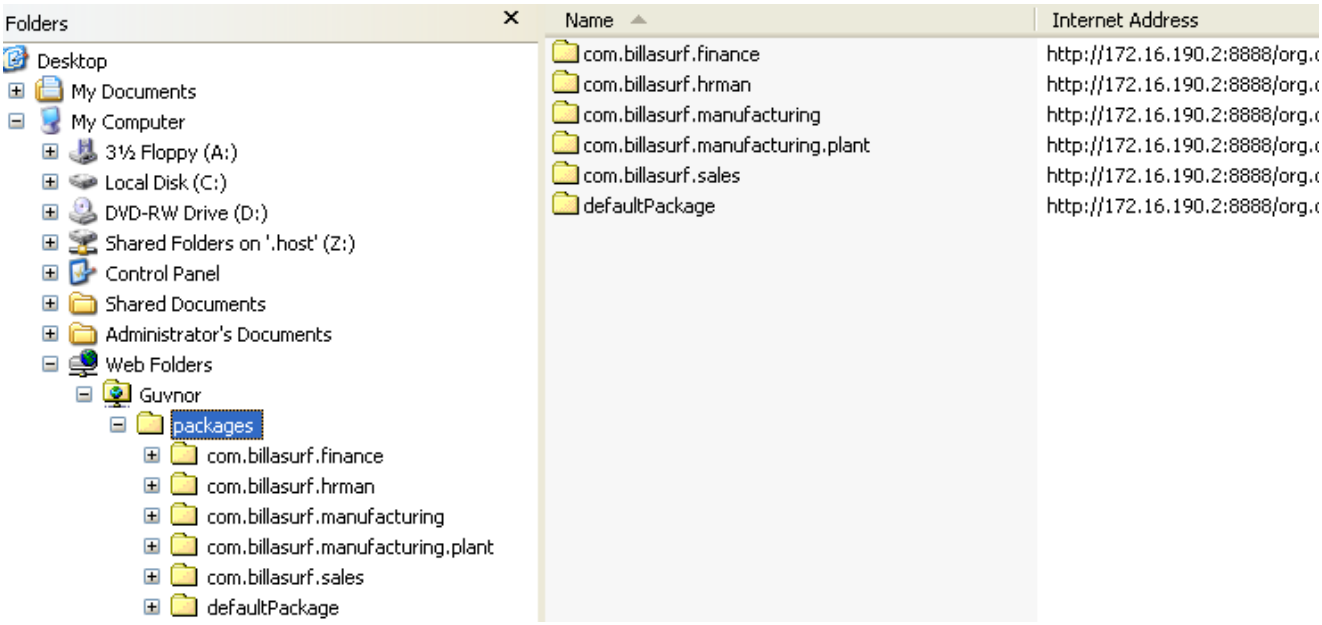


Figure 2.29. WebDAV

- Declarative modelling of types (types that are not in pojos)

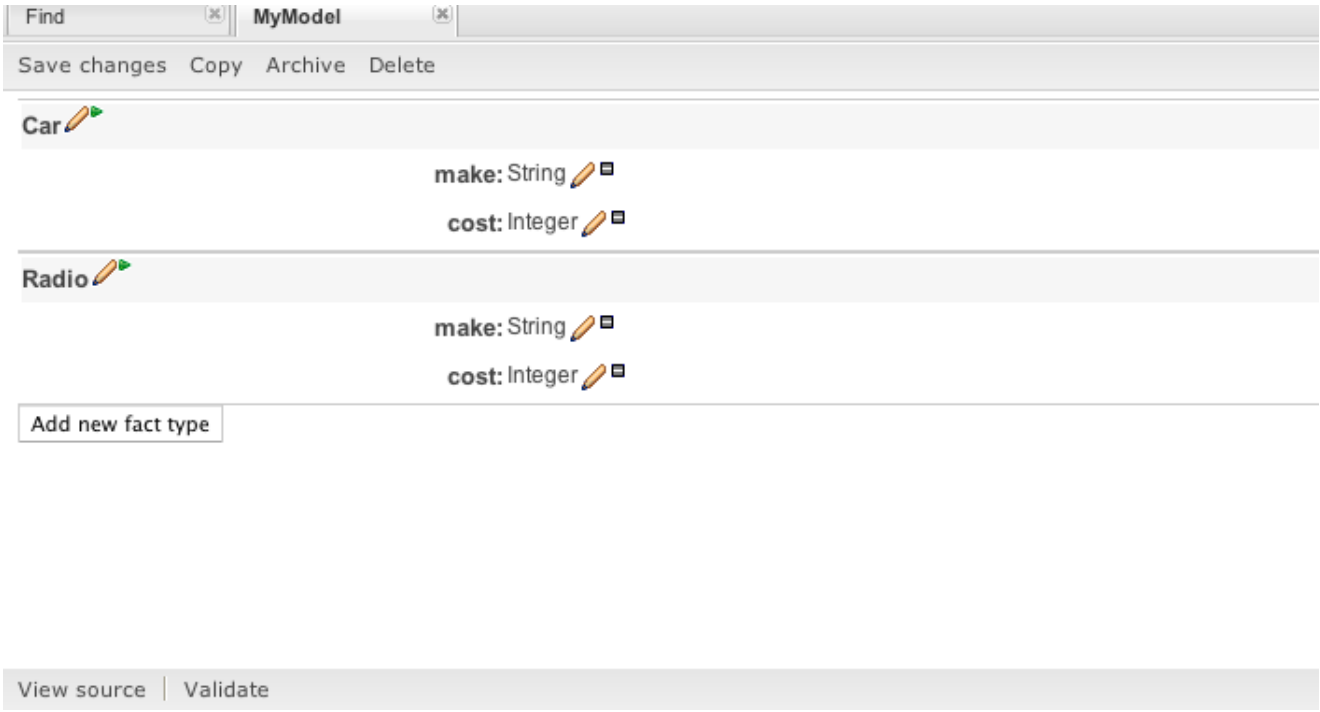


Figure 2.30. >Declarative modelling

This works with the new "declare" statement - you can now declare types in drl itself. You can then populate these without using a pojo (if you like). These types are then available in the rulebase.

- Fine grained security (lock down access to the app per package or per category). Users who only have category permissions have limited UI capability (ideal for business users)
- Execution server - access rules via XML or JSON for execution
- Category rules allows you to set 'parent rules' for a category. Any rules appearing in the given category will 'extend' the rule specified - ie inherit the conditions/LHS. The base rule for the category can be set on package configuration tab. RHS is not inherited, only the LHS
- Scenario runner detects infinite loops
- Scenario runner can show event trace that was recorded by audit logger
- DSL sentences in guided editor can now be set to show enums as a dropdown, dates as a date picker, booleans as a checkbox and use regular expressions to validate the inputs (DSL Widgets in Guvnor)
- Functions can be edited with text editor
- It is possible to add objects to global collections.
- Translations to English, Spanish, Chinese and Japanese

2.2.3. Drools Expert

2.2.3.1. Asymmetrical Rete algorithm implementation

Shadow proxies are no longer needed. Shadow proxies protected the engine from information change on facts, which if occurred outside of the engine's control it could not be modified or retracted.

2.2.3.2. `PackageBuilder` can now build multiple namespaces

You no longer need to confine one `PackageBuilder` to one package namespace. Just keeping adding your DRLs for any namespace and `getPackages()` returns an array of Packages for each of the used namespaces.

Example 2.26. Getting multiple packages

```
Package[] packages = pkgBuilder.getPackages();
```

2.2.3.3. `RuleBase` attachment to `PackageBuilder`

It is now possible to attach a `RuleBase` to a `PackageBuilder`, this means that rules are built and added to the rulebase at the same time. `PackageBuilder` uses the `Package` instances of the actual `RuleBase` as it's source, removing the need for additional `Package` creation and merging that happens in the existing approach.

Example 2.27. Attaching `RuleBase` to `PackageBuilder`

```
RuleBase ruleBase = RuleBaseFactory.newRuleBase();
PackageBuilder pkgBuilder = new PackageBuilder( ruleBase, null );
```

2.2.3.4. Binary marshalling of stateful sessions

Stateful sessions can now saved and resumed at a later date. Pre-loaded data sessions can now be created. Pluggable strategies can be used for user object persistence, i.e. hibernate or identity maps.

2.2.3.5. Type Declaration

Drools now supports a new base construct called Type Declaration. This construct fulfils two purposes: the ability to declare fact metadata, and the ability to dynamically generate new fact types local to the rule engine. The Guvnor modelling tool uses this underneath. One example of the construct is:

Example 2.28. Declaring `StockTick`

```
declare StockTick
  @role( event )
  @timestamp( timestampAttr )

  companySymbol : String
  stockPrice : double
  timestampAttr : long
end
```

2.2.3.6. Declaring Fact Metadata

To declare and associate fact metadata, just use the @ symbol for each metadata ID you want to declare. Example:

Example 2.29. Declaring metadata

```
declare StockTick
  @role( event )
end
```

2.2.3.7. Triggering Bean Generation

To activate the dynamic bean generation, just add fields and types to your type declaration:

Example 2.30. Declaring `Person`

```
declare Person
  name : String
  age : int
end
```

2.2.3.8. DSL improvements

A series of DSL improvements were implemented, including a completely new parser and the ability to declare matching masks for matching variables. For instance, one can constrain a phone number field to a 2-digit country code + 3-digit area code + 8-digit phone number, all connected

by a "-" (dash), by declaring the DSL map like: The phone number is {number:\d{2}-\d{3}-\d{8}} Any valid java regexp may be used in the variable mask.

2.2.3.9. `fireUntilHalt()`

Drools now supports "fireUntilHalt()" feature, that starts the engine in a reactive mode, where rules will be continually fired, until a halt() call is made. This is specially useful for CEP scenarios that require what is commonly known as "active queries".

2.2.3.10. Rule Base partitioning and multi-thread propagation

Drools ReteOO algorithm now supports an option to start the rule base in a multi-thread mode, where Drools ReteOO network is split into multiple partitions and rules are then evaluated concurrently by multiple threads. This is also a requirement for CEP where there usually are several independent rules running concurrently, with near realtime performance/throughput requirements and the evaluation of one can not interfere with the evaluation of others.

2.2.3.11. XSD Model Support

Drools now supports XSD models. Remember though the XSD model is generated as pojos local to the Drools classloader. A helper class is there to assist in the creation of the model in the packagebuilder. Once the data model is generated you'll typically use the JAXB dataloader to insert data.

2.2.3.12. Data Loaders

Drools now supports two data loaders, Smooks and JAXB. Smooks is an open source data transformation tool for ETL and JAXB a standard sun data mapping tool. Unit tests showing Smooks can be found here and JAXB here.

2.2.3.13. Type safe configuration

In addition to the ability of configuring options in drools through configuration files, system properties and by setting properties through the API `setProperty()` method, Drools-API now supports type safe configuration. We didn't want to add specific methods for each possible configuration methods for two reasons: it polutes the API and every time a new option is added to Drools, the API would have to change. This way, we followed a modular, class based configuration, where a new Option class is added to the API for each possible configuration, keeping the API stable, but flexible at the same time. So, in order to set configuration options now, you just need to use the enumerations or factories provided for each option. For instance, if you want to configure the knowledge base for assert behavior "equality" and to automatically remove identities from pattern matchings, you would just use the enums:

Example 2.31. Configuring

```
KnowledgeBaseConfiguration config =
 KnowledgeBaseFactory.newKnowledgeBaseConfiguration();
config.setOption( AssertBehaviorOption.EQUALITY );
config.setOption( RemoveIdentitiesOption.YES );
```

For options that don't have a predefined constant or can assume multiple values, a factory method is provided. For instance, to configure the alpha threshold to 5, just use the "get" factory method:

Example 2.32. Configuring alpha threshold

```
config.setOption( AlphaThresholdOption.get(5) );
```

As you can see, the same `setOption()` method is used for the different possible configurations, but they are still type safe.

2.2.3.14. New accumulate functions: collectSet and collectList

There are times when it is necessary to collect sets or lists of values that are derived from the facts attributes, but are not facts themselves. In such cases, it was not possible to use the collect CE. So, Drools now has two accumulate functions for such cases: collectSet for collecting sets of values (i.e., with no duplicate values) and collectList for collecting lists of values (i.e., allowing duplicate values):

Example 2.33. New accumulate functions

```
# collect the set of unique names in the working memory
$names : Set() from accumulate( Person( $n : name, $s : surname ),
                    collectSet( $n + " " + $s ) )

# collect the list of alarm codes from the alarms in the working memory
$codes : List() from accumulate( Alarm( $c : code, $s : severity ),
                     collectList( $c + $s ) )
```

2.2.3.15. New metadata for type declarations: @propertyChangeSupport

Facts that implement support for property changes as defined in the Javabean(tm) spec, now can be annotated so that the engine register itself to listen for changes on fact properties. The boolean parameter that was used in the insert() method in the Drools 4 API is deprecated and does not exist in the drools-api module.

Example 2.34. @propertyChangeSupport

```
declare Person
 @propertyChangeSupport
end
```

2.2.3.16. Batch Executor

Batch Executor allows for the scripting of of a Knowledge session using Commands, which can also re, both the `StatelessKnowledgeSession` and `StatefulKnowledgeSession` implement this interface Commands are created using the `CommandFactory` and executed using the "execute" method, such as the following insert Command:

Example 2.35. Using `CommandFactory`

```
ksession.execute( CommandFactory.newInsert( person ) );
```

Typically though you will want to execute a batch of commands, this can be achieved via the composite Command `BatchExecution`. `BatchExecutionResults` is now used to handle the results, some commands can specify "out" identifiers which it used to add the result to the `BatchExecutionResult`. Handily querries can now be executed and results added to the `BatchExecutionResult`. Further to this results are scoped to this execute call and return via the `BatchExecutionResults`:

Example 2.36. Using `BatchExecutionResult`

```
List<Command> cmds = new ArrayList<Command>();
cmds.add( CommandFactory.newSetGlobal( "list1", new ArrayList(), true ) );
cmds.add( CommandFactory.newInsert( new Person( "jon", 102 ), "person" ) );
cmds.add( CommandFactory.newQuery( "Get People" "getPeople" );

BatchExecutionResults                                    results                                    =
 ksession.execute( CommandFactory.newBatchExecution( cmds ) );
results.getValue( "list1" ); // returns the ArrayList
results.getValue( "person" ); // returns the inserted fact Person
results.getValue( "Get People" );// returns the query as a QueryResults instance.
end
```

The `CommandFactory` details the supported commands, all of which can marshalled using XStream and the `BatchExecutionHelper`. This can be combined with the pipeline to automate the scripting of a session.

Example 2.37. Using `PipelineFactory`

```
Action executeResultHandler = PipelineFactory.newExecuteResultHandler();
Action assignResult = PipelineFactory.newAssignObjectAsResult();
assignResult.setReceiver( executeResultHandler );
Transformer outTransformer =
 PipelineFactory.newXStreamToXmlTransformer( BatchExecutionHelper.newXStreamMarshaller() );
outTransformer.setReceiver( assignResult );
KnowledgeRuntimeCommand batchExecution = PipelineFactory.newBatchExecutor();
batchExecution.setReceiver( outTransformer );
Transformer inTransformer =
 PipelineFactory.newXStreamFromXmlTransformer( BatchExecutionHelper.newXStreamMarshaller() );
inTransformer.setReceiver( batchExecution );
Pipeline pipeline = PipelineFactory.newStatelessKnowledgeSessionPipeline( ksession );
pipeline.setReceiver( inTransformer );
```

Using the above for a rulset that updates the price of a Cheese fact, given the following xml to insert a Cheese instance using an out-identifier:

Example 2.38. Updating Cheese fact

```
<batch-execution>
<insert out-identifier='outStilton'>
  <org.drools.Cheese>
    <type>stilton</type>
    <price>25</price>
    <oldPrice>0</oldPrice>
  </org.drools.Cheese>
</insert>
</batch-execution>
```

We then get the following `BatchExecutionResults`:

Example 2.39. Updating Cheese fact

```
<batch-execution-results>
```

```
 <result identifier='outStilton'>
   <org.drools.Cheese>
     <type>stilton</type>
     <oldPrice>0</oldPrice>
     <price>30</price>
   </org.drools.Cheese>
 </result>
</batch-execution-results>
```

2.2.3.17. Marshalling

The `MarshallerFactory` is used to marshal and unmarshal `StatefulKnowledgeSessions`. At the simplest it can be used as follows:

Example 2.40. Using `MarshallerFactory`

```
// ksession is the StatefulKnowledgeSession
// kbase is the KnowledgeBase
ByteArrayOutputStream baos = new ByteArrayOutputStream();
Marshaller marshaller = MarshallerFactory.newMarshaller( kbase );
marshaller.marshall( baos, ksession );
baos.close();
```

However with marshalling you need more flexibility when dealing with referenced user data. To achieve this we have the `ObjectMarshallingStrategy` interface. Two implementations are provided, but the user can implement their own. The two supplied are `IdentityMarshallingStrategy` and `SerializeMarshallingStrategy`. `SerializeMarshallingStrategy` is the default, as used in the example above and it just calls the `Serializable` or `Externalizable` methods on a user instance. `IdentityMarshallingStrategy` instead creates an int id for each user object and stores them in a `Map` the id is written to the stream. When unmarshalling it simply looks to the `IdentityMarshallingStrategy` map to retrieve the instance. This means that if you use the `IdentityMarshallingStrategy` it's stateful for the life of the Marshaller instance and will create ids and keep references to all objects that it attempts to marshal.

Example 2.41. Code to use a `IdentityMarshallingStrategy`

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
Marshaller marshaller = MarshallerFactory.newMarshaller( kbase, new
 ObjectMarshallingStrategy[] { MarshallerFactory.newIdentityMarshallingStrategy() } );
```

```
marshaller.marshall( baos, ksession );
baos.close();
```

For added flexability we can't assume that a single strategy is suitable for this we have added the `ObjectMarshallingStrategyAcceptor` interface that each `ObjectMarshallingStrategy` has. The Marshaller has a chain of strategies and when it attempts to read or write a user object it iterates the strategies asking if they accept responsability for marshalling the user object. One one implementation is provided the `ClassFilterAcceptor`. This allows strings and wild cards to be used to match class names. The default is "*.*", so in the above the `IdentityMarshallingStrategy` is used which has a default "*.*" acceptor. But lets say we want to serialise all classes except for one given package, where we will use identity lookup, we could do the following:

Example 2.42. Using identity lookup

```
ByteArrayOutputStream baos = new ByteArrayOutputStream();
ObjectMarshallingStrategyAcceptor identityAceceptor =
 MarshallerFactory.newClassFilterAcceptor( new String[] { "org.domain.pkg1.*" } );
ObjectMarshallingStrategy identityStratetgy =
 MarshallerFactory.newIdentityMarshallingStrategy( identityAceceptor );
Marshaller marshaller = MarshallerFactory.newMarshaller( kbase, new
                        ObjectMarshallingStrategy[] { identityStrat
 MarshallerFactory.newSerializeMarshallingStrategy() } );
marshaller.marshall( baos, ksession );
baos.close();
```

2.2.3.18. Knowledge Agent

The `KnowlegeAgent` is created by the `KnowlegeAgentFactory`. The `KnowlegeAgent` provides automatic loading, caching and re-loading, of resources and is configured from a properties files. The `KnowledgeAgent` can update or rebuild this `KnowlegeBase` as the resources it uses are changed. The strategy for this is determined by the configuration given to the factory, but it is typically pull based using regular polling. We hope to add push based updates and rebuilds in future versions. The Following example constructs an agent that will build a new `KnowledgeBase` from the files specified in the path String. It will poll those files every 30 seconds to see if they are updated. If new files are found it will construct a new KnowledgeBase, instead of updating the existing one, due to the "newInstance" set to "true" (however currently only the value of "true" is supported and is hard coded into the engine):

Example 2.43. Constructing an agent

```
// Set the interval on the ResourceChangeScannerService if you are to use it and default of 60s is not desirable.
ResourceChangeScannerConfiguration sconf = ResourceFactory.getResourceChangeScannerService().newResourceChangeScannerConfiguration();
sconf.setProperty( "drools.resource.scanner.interval",
                   "30" ); // set the disk scanning interval to 30s, default is 60s
ResourceFactory.getResourceChangeScannerService().configure( sconf );
KnowledgeBase kbase = KnowledgeBaseFactory.newKnowledgeBase();
KnowledgeAgentConfiguration aconf = KnowledgeAgentFactory.newKnowledgeAgentConfiguration();
aconf.setProperty( "drools.agent.scanDirectories",
                   "true" ); // we want to scan directories, not just files, turning this on turns on file scanning
aconf.setProperty( "drools.agent.newInstance",
                   "true" ); // resource changes results in a new instance of the KnowledgeBase being built,
                             // this cannot currently be set to false for incremental building

KnowledgeAgent kagent = KnowledgeAgentFactory.newKnowledgeAgent( "test agent", // the name of the agent
                                                 kbase, // the KnowledgeBase to use, the Agent will also monitor any exist knowledge definitions
                                                 aconf );
kagent.applyChangeSet( ResourceFactory.newUrlResource( url ) ); // resource to the change-set xml for the resources to add
```

`KnowledgeAgents` can take a empty `KnowledgeBase` or a populated one. If a populated `KnowledgeBase` is provided, the `KnowledgeAgent` will iterate `KnowledgeBase` and subscribe to the `Resource` that it finds. While it is possible for the `KnowledgeBuilder` to build all resources found in a directory, that information is lost by the KnowledgeBuilder so those directories will not be continuously scanned. Only directories specified as part of the `applyChangeSet(Resource)` method are monitored.

2.2.4. Drools Flow

Drools 4.0 had simple "RuleFlow" which was for orchestrating rules. Drools 5.0 introduces a powerful (extensible) workflow engine. It allows users to specify their business logic using both rules and processes (where powerful interaction between processes and rules is possible) and offers a unified enviroment.

2.2.4.1. Process Instance view at a specific breakpoint

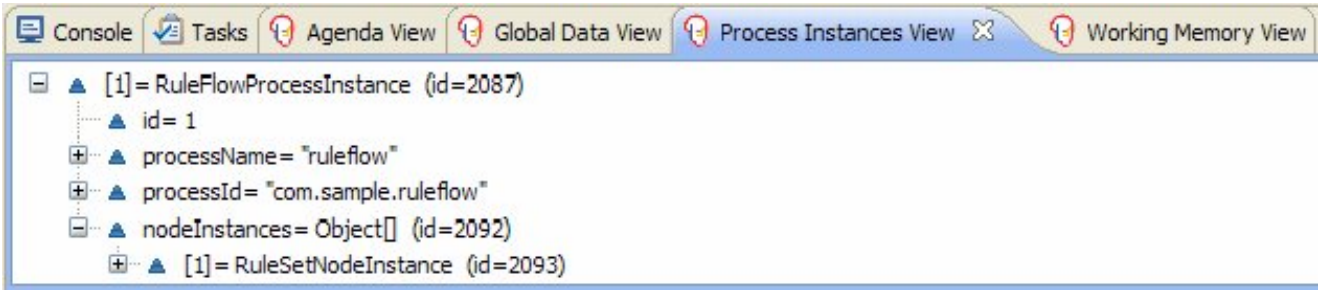


Figure 2.31. Rule Flow properties

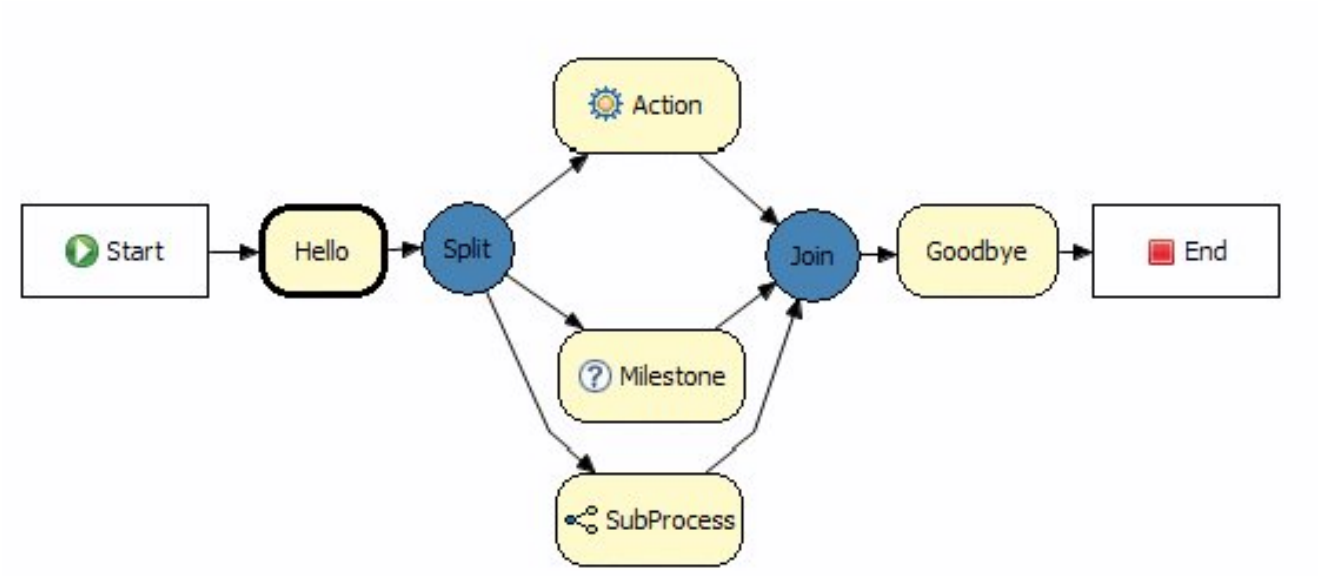


Figure 2.32. Current active nodes in a workflow in a specific breakpoint

2.2.4.2. New Nodes

Timers:

A timer node can be added which causes the execution of the node to wait for a specific period. Currently just uses JDK defaults of initial delay and repeat delay, more complex timers will be available in further milestones.

Human Task:

Processes can include tasks that need to be executed by human actors. Human tasks include parameters like taskname, priority, description, actorId, etc. The process engine can easily be integrated with existing human task component (like for example a WS-HumanTask implementation) using our pluggable work items (see below). Swimlanes and assignment rules are also supported.

The palette in the screenshot shows the two new components, and the workflow itself shows the human task in use. It also shows two "work items" which is explained in the next section:

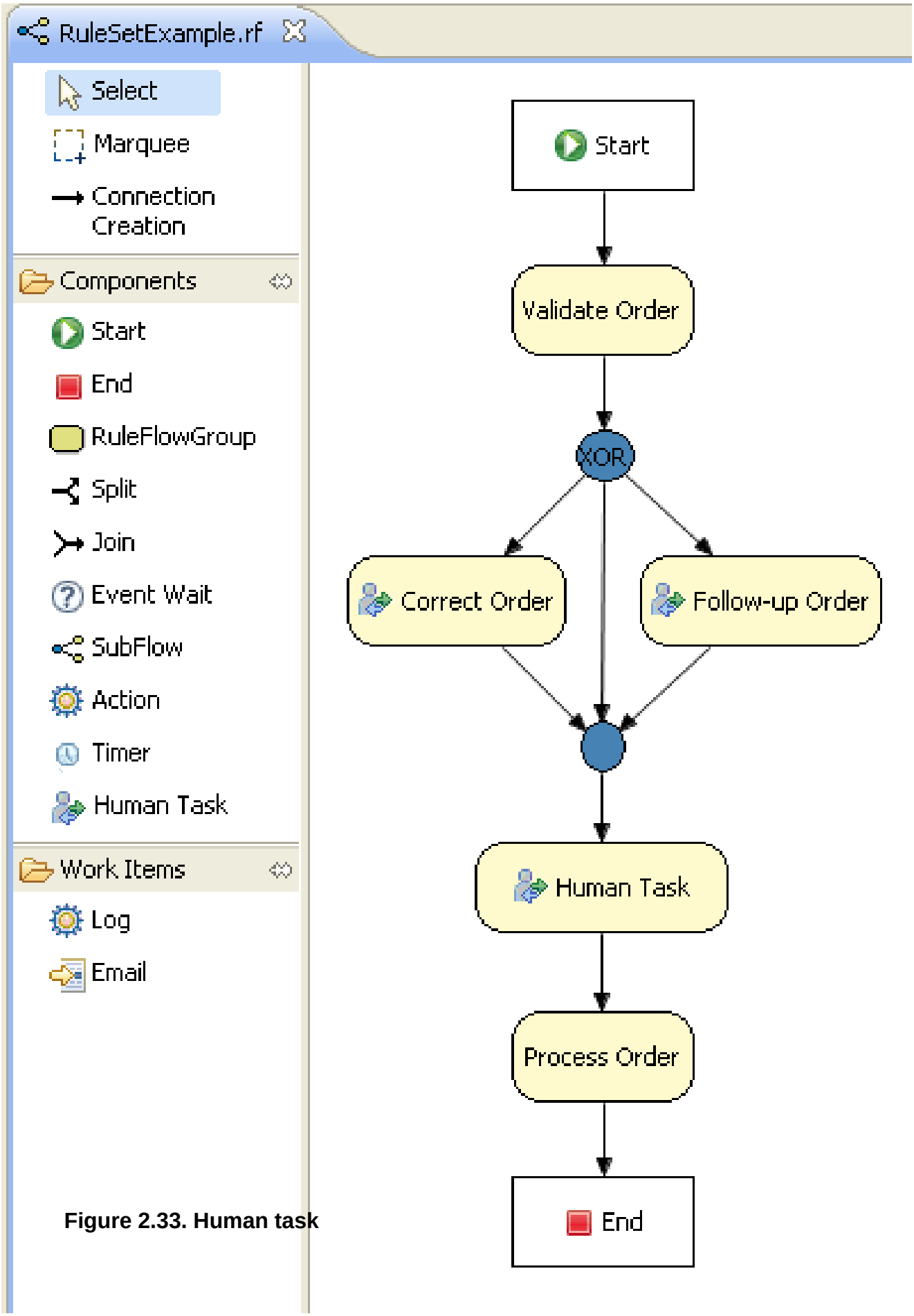


Figure 2.33. Human task

2.2.4.3. Domain Specific Work Items

Domain Specific Work Items are pluggable nodes that users create to facilitate custom task execution. They provide an api to specify a new icon in the palette and gui editor for the tasks properties, if no editor gui is supplied then it defaults to a text based key value pair form. The api then allows execution behaviour for these work items to be specified. By default the Email and Log work items are provided. The Drools flow Manual has been updated on how to implement these.

The below image shows three different work items in use in a workflow, "Blood Pressure", "BP Medication", "Notify GP":

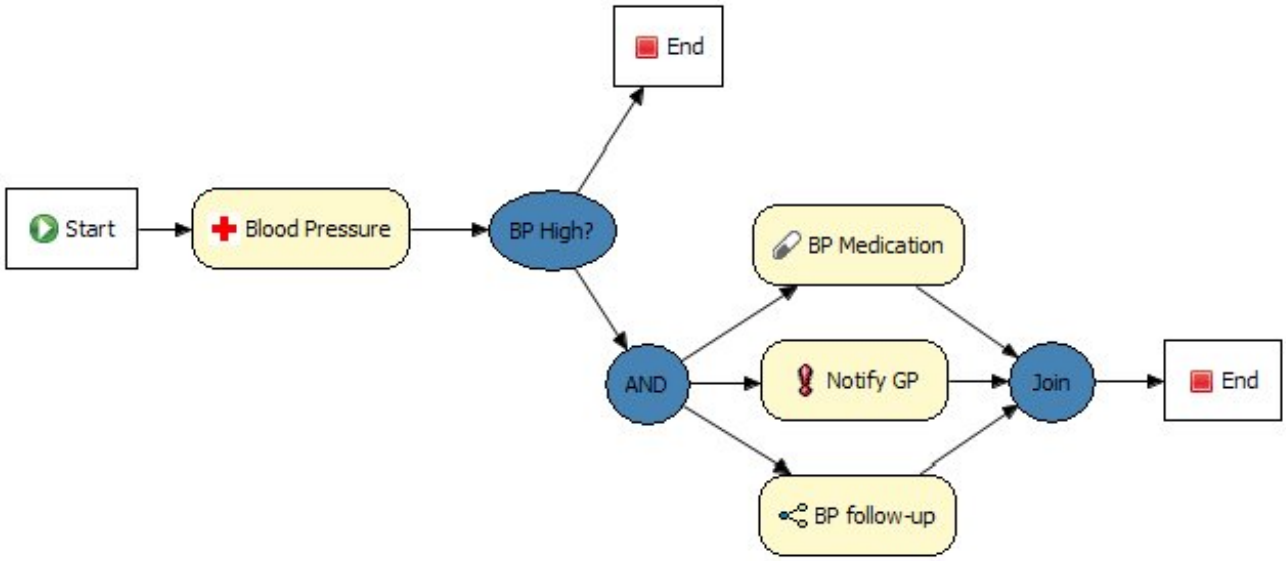


Figure 2.34. Work items

This one ows a new "Notification" work item:

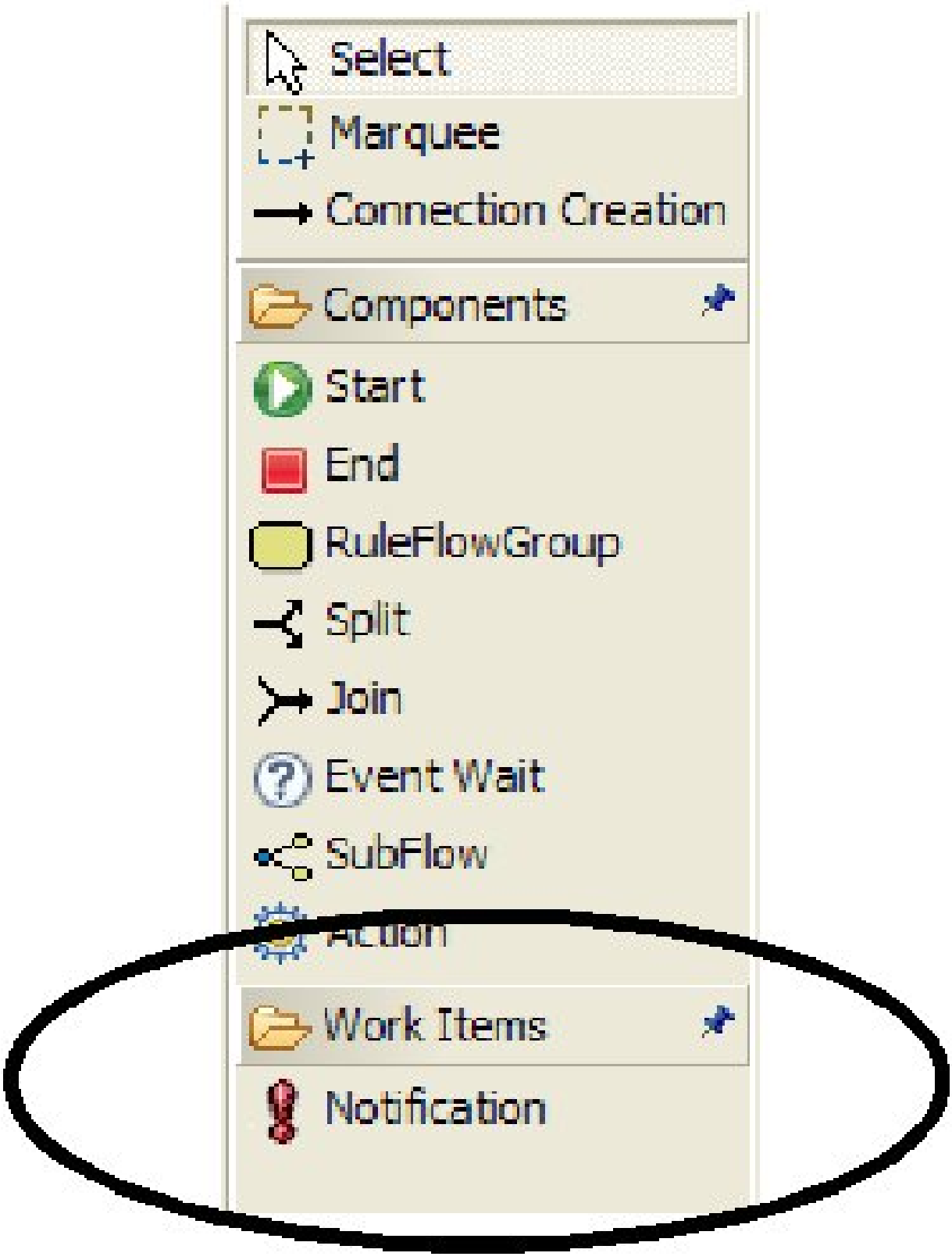


Figure 2.35. Notification

2.2.4.4. Extensible Process Definition Language (ePDL)

Drools 4.0 used Xstream to store it's content, which was not easily human writeable. Drools 5.0 introduced the ePDL which is a XML specific to our process language, it also allows for domain specific extensions which has been talked about in detail in this blog posting "Drools Extensible Process Definition Language (ePDL) and the Semantic Module Framework (SMF)". An example of the XML language, with a DSL extension in red, is shown below.

Example 2.44. Example of the XML language

```
<process  name="process name" id="process name" package-name="org.domain"
xmlns="http://drools.org/drools-4.0/process"
xmlns:mydsl="http://domain/org/mydsl"
xmlns:xs="http://www.w3.org/2001/XMLSchema-instance"
xs:schemaLocation="http://drools.org/drools-4.0/process drools-processes-4.0.xsd" >
<nodes>
  <start id="0" />

  <action id="1" dialect="java">
     list.add( "action node was here" );
  </action>

    <mydsl:logger id="2" type="warn">
       This is my message
    <mydsl:logger>

  <end id="3" />
</nodes>

<connections>
  <connection from="0 to="1" />
  <connection from="1" to="2" />
  <connection from="2" to="3" />
</connections>

</process>
```

2.2.4.5. Pluggable Nodes

The underlying nodes for the framework are completely pluggable making it simple to extend and to implement other execution models. We already have a partial implementation for OSWorkflow and are working with Deigo to complete this to provide a migration path for OSWorkflow users. Other enhancements include exception scopes, the ability to include on-entry and on-exit actions

on various node types, integration with our binary persistence mechanism to persist the state of long running processes, etc. Check out the Drools Flow documentation to learn more.

2.2.4.6. Human tasks

Human task management is very important in the context of processes. While we allow users to plug in any task component they prefer, we have developed a human task management component that supports the entire life cycle of human tasks based on the WS-HumanTask specification.

2.2.4.7. New functions to the Drools Flow language

- Event nodes that allow a process to respond to external events
- Exception handlers and exception handler scopes to handle exceptions that could be thrown
- A ForEach node allows instantiating a section of your flow multiple times, for each element in a collection
- Data type support has been extended
- Timers are integrated with common node types

As a result, new node types and properties have been added to the Drools Flow editor in Eclipse. You can also find examples of these new features in the integration tests (e.g. ProcessExceptionHandlerTest, ProcessTimerTest, etc.).

2.2.4.8. Work items

Our pluggable work item approach allows you to plug in domain-specific work in your process in a declarative manner. We plan to build a library of common work items and already provide an implementation for sending emails, finding files, archiving, executing system commands, logging and human tasks.

2.2.4.9. JPA

Improved support for persistence (JPA) and transactions (JTA).

Example 2.45. An example on how to use persistence and transactions in combination with processes

```
// create a new JPA-based session and specify the JPA entity manager factory
Environment env = KnowledgeBaseFactory.newEnvironment();
env.set( EnvironmentName.ENTITY_MANAGER_FACTORY,
 Persistence.createEntityManagerFactory( "emf-name" ) );
```

```
env.set( EnvironmentName.TRANSACTION_MANAGER,
 TransactionManagerServices.getTransactionManager() );

StatefulKnowledgeSession ksession =
      JPAKnowledgeService.newStatefulKnowledgeSession( kbase, null, env ); //
 KnowledgeSessionConfiguration may be null, and a default will be used
int sessionId = ksession.getId();

// if no transaction boundary is specified, the method invocation is executed in a new transaction
 automatically
ProcessInstance processInstance = ksession.startProcess( "org.drools.test.TestProcess" );

// the users can also specify the transaction boundary themselves
UserTransaction ut = (UserTransaction) new InitialContext().lookup( "java:comp/
UserTransaction" );
ut.begin();
ksession.insert( new Person( "John Doe" ) );
ksession.startProcess( "org.drools.test.TestProcess" );
ksession.fireAllRules();
ut.commit();
```

2.2.4.10. Variable Injection

Support direct access to process variables in both MVEL and Java in code constraints and actions, so if you have a variable called "person" in your process, you can now describe constraints like:

Example 2.46. Variable injection example

```
* [Java code constraint] return person.getAge() > 20;
* [MVEL action] System.out.println(person.name);
```

2.2.4.11. Miscellaneous Enhancements

- Process instances can now listen for external events by marking the event node property "external" as true. External events are signaled to the engine using `session.signalEvent(type, eventData)` More information on how to use events inside your processes can be found in the Drools Flow documentation here: https://hudson.jboss.org/hudson/job/drools/lastSuccessfulBuild/artifact/trunk/target/docs/drools-flow/html/ch03.html#d0e917
- Process instances are now safe for multi-threading (as multiple thread are blocked from working on the same process instance)

- Process persistence / transaction support has been further improved. Check out the drools-process/drools-process-enterprise project for more information.
- The human task component has been extended to support all kinds of data for input / output / exceptions during task execution.

Example 2.47. As a result, the life cycle methods of the task client have been extended to allow content data

```
taskClient.addTask(task, contentData, responseHandler)
taskClient.complete(taskId, userId, outputData,responseHandler)
taskFail.complete(taskId, userId, outputData,responseHandler)

long contentId = task.getTaskData().getDocumentContentId();
taskClient.getContent(contentId, responseHandler);
ContentData content = responseHandler.getContent();
```

- It is now possible to migrate old Drools4 RuleFlows (using the xstream format) to Drools5 processes (using readable xml) during compilation. Migration will automatically be performed when adding the RuleFlow to the KnowledgeBase when the following system property is set: `drools.ruleflow.port = true`
- The "Transform" work item allows you to easily transform data from one format to another inside processes. The code and an example can be found in the drools-process/drools-workitems directory.
- Function imports are now also supported inside processes.
- The history log - that keeps the history of all executed process instances in a database - has been extended so it is now caable of storing more detailed information for one specfic process instance. It is now possible to find out exactly which nodes were triggered during the execution of the process instance.
- A new type of join has been added, one that will wait until n of its m incoming connections have been completed. This n could either be hardcoded in the process or based on the value of a variable in the process.
- Improvements have been made to make persistence easier to configure. The persistence approach is based on a command service that makes sure that all the client invocations are executed inside a transaction and that the state is stored in the database after successful execution of the command. While this was already possible in M4 using the commands directly, we have extended this so that people can simply use the normal StatefulKnowledgeSession interface but simply can configure the persistence using configuration files. For more details, check out the chapter on persistence in the Drools Flow documentation.

2.2.5. Drools Fusion

Drools 5.0 brings to the rules world the full power of events processing by supporting a number of CEP features as well as supporting events as first class citizens in the rules engine.

2.2.5.1. Event Semantics

Events are (from a rules engine perspective) a special type of fact that has a few special characteristics:

- they are immutable
- they have strong time-related relationships
- they may have clear lifecycle windows
- they may be transparently garbage collected after it's lifecycle window expires
- they may be time-constrained
- they may be included in sliding windows for reasoning

2.2.5.2. Event Declaration

Any fact type can assume an event role, and its corresponding event semantics, by simply declaring the metadata for it.

Example 2.48. Both existing and generated beans support event semantics:

```
# existing bean assuming an event role
import org.drools.test.StockTick
declare StockTick
  @role( event )
end

# generated bean assuming an event role
declare Alarm
  @role( event )
  type : String
  timestamp : long
end
```

2.2.5.3. Entry-Point Stream Listeners

A new key "from entry-point" has been added to allow a pattern in a rule to listen on a stream, which avoids the overhead of having to insert the object into the working memory where it is potentially reasoned over by all rules.

```
$st : StockTick( company == "ACME", price > 10 ) from entry-point "stock stream"
```

Example 2.49. To insert facts into an entry point

```
WorkingMemoryEntryPoint entry = wm.getWorkingMemoryEntryPoint( "stock stream" );
entry.insert( ticker );
```

StreamTest shows a unit for this.

2.2.5.4. Event Correlation and New Operators

Event correlation and time based constraint support are requirements of event processing, and are completely supported by Drools 5.0. The new, out of the box, time constraint operators can be seen in these test case rules: test_CEP_TimeRelationalOperators.drl

As seen in the test above, Drools supports both: primitive events, that are point in time occurrences with no duration, and compound events, that are events with distinct start and end timestamps.

The complete list of operators are:

- coincides
- before
- after
- meets
- metby
- overlaps
- overlappedby
- during
- includes

- starts
- startedby
- finishes
- finishedby

2.2.5.5. Sliding Time Windows

Drools 5.0 adds support for reasoning over sliding windows of events. For instance:

```
StockTick( symbol == "RHAT" ) over window:time( 60 )
```

The above example will only pattern match the RHAT stock ticks that happened in the last 60 clock ticks, discarding any event older than that.

2.2.5.6. Session Clock

Enabling full event processing capabilities requires the ability to configure and interact with a session clock. Drools adds support for time reasoning and session clock configuration, allowing it to not only run real time event processing but also simulations, what-if scenarios and post-processing audit by replaying a scenario.

Example 2.50. The Clock is specified as part of the SessionConfiguration, a new class that is optionally specified at session creation time

```
SessionConfiguration conf = new SessionConfiguration();
conf.setClockType( ClockType.PSEUDO_CLOCK );
StatefulSession session = ruleBase.newStatefulSession( conf );
```

2.2.5.7. Event Garbage Collection

Since events usually have strong temporal relationships, it is possible to infer a logical time window when events can possibly match. The engine uses that capability to calculate when an event is no longer capable of matching any rule anymore and automatically retracts that event.

2.2.5.8. Time Units Support

Drools adopted a simplified syntax for time units, based on the ISO 8601 syntax for durations. This allows users to easily add temporal constraints to the rules writing time in well known units. Example:

```
SomeEvent( this after[1m,1h30m] $anotherEvent )
```

The above pattern will match if SomeEvent happens between 1 minute (1m) and 1 hour and 30 minutes after `$anotherEvent`.

2.2.5.9. Support to event expiration policy

added the ability to define a per-type event expiration policy. In the example bellow, the StockTick events will expire 10 minutes after they enter the system:

```
declare StockTick @role( event ) @expires( 10m ) end
```

2.2.5.10. Support to temporal operations over arbitrary dates.

Example 2.51. added the ability for point-in-time operators (before, after and coincides) to be used with any arbitrary date field

```
rule "Allow access"
when
WorkHours( $s : start, $e : end )
LogIn( time after $s, time before $e )
then
// allow access
end
```

2.2.6. Eclipse IDE

- Support multiple runtimes: The IDE now supports multiple runtimes. A Drools runtime is a collection of jars on your file system that represent one specific release of the Drools project jars. To create a runtime, you must either point the IDE to the release of your choice, or you can simply create a new runtime on your file system from the jars included in the Drools Eclipse plugin. Drools runtimes can be configured by opening up the Eclipse preferences and selecting the Drools -> Installed Drools Runtimes category, as shown below.

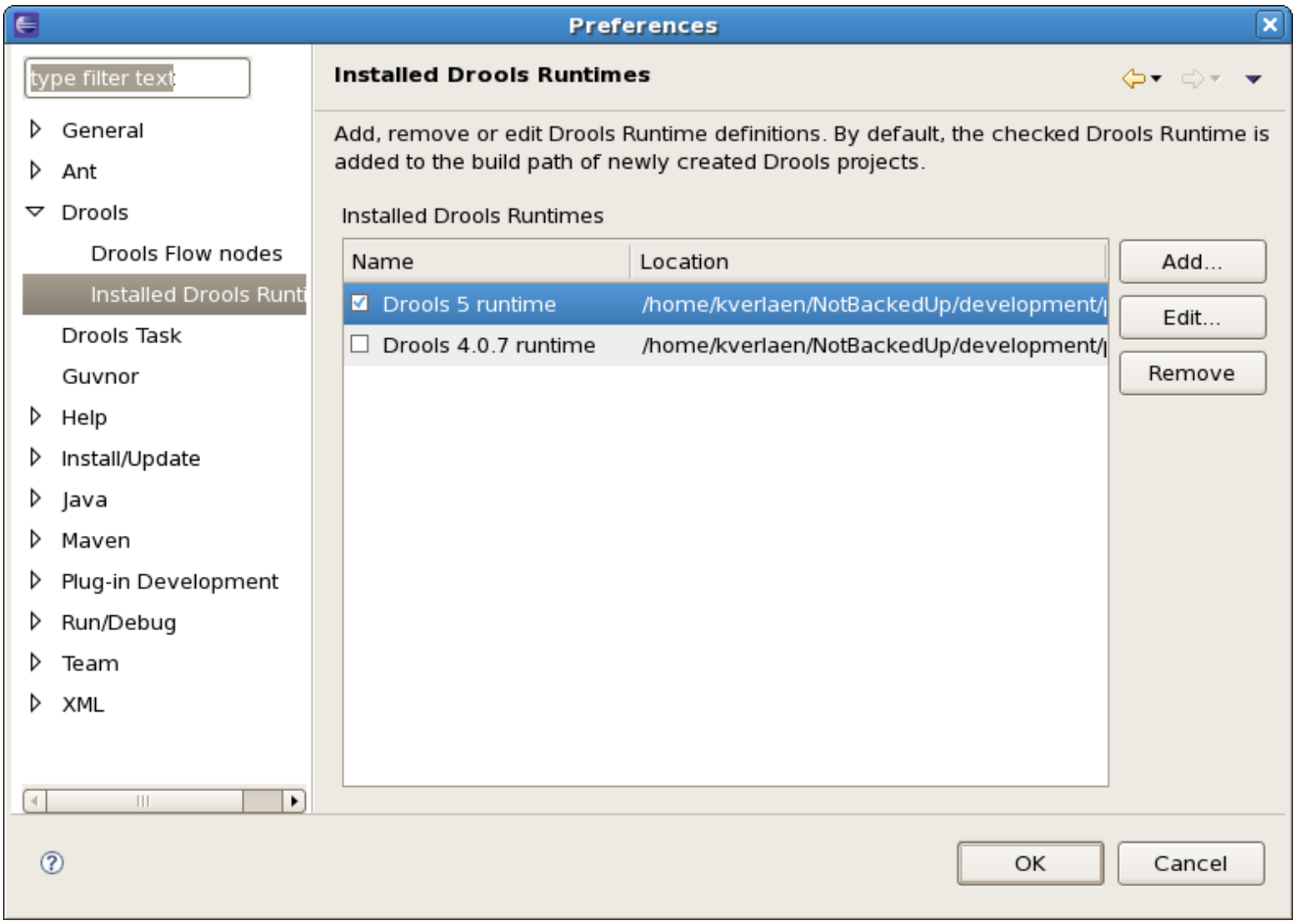


Figure 2.36. Run times

- Debugging of rules using the MVEL dialect has been fixed
- Drools Flow Editor
  - Process Skins allow you to define how the different RuleFlow nodes are visualized. We now support two skins: the default one which existed before and a BPMN skin that visualizes the nodes using a BPMN-like representation: http://blog.athico.com/2008/10/drools-flow-and-bpmn.html
  - An (X)OR split now shows the name of the constraint as the connection label
  - Custom work item editors now signal the process correctly that it has been changed

2.3. What is new in Drools 4.0

Drools 4.0 is a major update over the previous Drools 3.0.x series. A whole new set of features were developed which special focus on language expressiveness, engine performance and tools availability. The following is a list of the most interesting changes.

2.3.1. Language Expressiveness Enhancements

- New Conditional Elements: from, collect, accumulate and forall
- New Field Constraint operators: not matches, not contains, in, not in, memberOf, not memberOf
- New Implicit Self Reference field: this
- Full support for Conditional Elements nesting, for First Order Logic completeness.
- Support for multi-restrictions and constraint connectives && and ||
- Parser improvements to remove previous language limitations, like character escaping and keyword conflicts
- Support for pluggable dialects and full support for MVEL scripting language
- Complete rewrite of DSL engine, allowing for full l10n
- Fact attributes auto-vivification for return value restrictions and inline-eval constraints
- Support for nested accessors, property navigation and simplified collection, arrays and maps syntax
- Improved support for XML rules

2.3.2. Core Engine Enhancements

- Native support for primitive types, avoiding constant autoboxing
- Support for transparent optional Shadow Facts
- Rete Network performance improvements for complex rules
- Support for Rule-Flows
- Support for Stateful and Stateless working memories (rule engine sessions)
- Support for Asynchronous Working Memory actions
- Rules Engine Agent for hot deployment and BRMS integration
- Dynamic salience for rules conflict resolution
- Support for Parameterized Queries
- Support for halt command

- Support for sequential execution mode
- Support for pluggable global variable resolver

2.3.3. IDE Enhancements

- Support for rule break-points on debugging
- WYSIWYG support for rule-flows
- New guided editor for rules authoring
- Upgrade to support all new engine features

2.3.4. Business Rules Management System - BRMS

- New BRMS tool
- User friendly web interface with nice WEB 2.0 ajax features
- Package configuration
- Rule Authoring easy to edit rules both with guided editor (drop-down menus) and text editor
- Package compilation and deployment
- Easy deployment with Rule Agent
- Easy to organize with categories and search assets
- Versioning enabled, you can easily replace yours assets with previously saved
- JCR compliant rule assets repository

2.3.5. Miscellaneous Enhancements

- Slimmed down dependencies and smaller memory footprint

2.4. Upgrade tips from Drools 3.0.x to Drools 4.0.x

As mentioned before Drools 4.0 is a major update over the previous Drools 3.0.x series. Unfortunately, in order to achieve the goals set for this release, some backward compatibility issues were introduced, as discussed in the mail list and blogs.

This section of the manual is a work in progress and will document a simple how-to on upgrading from Drools 3.0.x to Drools 4.0.x.

2.4.1. API changes

There are a few API changes that are visible to regular users and need to be fixed.

2.4.1.1. Working Memory creation

Drools 3.0.x had only one working memory type that worked like a stateful working memory. Drools 4.0.x introduces separate APIs for Stateful and Stateless working memories that are called now Rule Sessions. In Drools 3.0.x, the code to create a working memory was:

Example 2.52. Drools 3.0.x: Working Memory Creation

```
WorkingMemory wm = rulebase.newWorkingMemory();
```

In Drools 4.0.x it must be changed to:

Example 2.53. Drools 4.0.x: Stateful Rule Session Creation

```
StatefulSession wm = rulebase.newStatefulSession();
```

The StatefulSession object has the same behavior as the Drools 3.0.x WorkingMemory (it even extends the WorkingMemory interface), so there should be no other problems with this fix.

2.4.1.2. Working Memory Actions

Drools 4.0.x now supports pluggable dialects and has built-in support for Java and MVEL scripting language. In order to avoid keyword conflicts, the working memory actions were renamed as showed bellow:

Table 2.1. Working Memory Actions equivalent API methods

Drools 3.0.x	Drools 4.0.x
WorkingMemory.assertObject()	WorkingMemory.insert()
WorkingMemory.assertLogicalObject()	WorkingMemory.insertLogical()
WorkingMemory.modifyObject()	WorkingMemory.update()

2.4.2. Rule Language Changes

The DRL Rule Language also has some backward incompatible changes as detailed bellow.

2.4.2.1. Working Memory Actions

The Working Memory actions in rule consequences were also changed in a similar way to the change made in the API. The following table summarizes the change:

Table 2.2. Working Memory Actions equivalent DRL commands

Drools 3.0.x	Drools 4.0.x
assert()	insert()
assertLogical()	insertLogical()
modify()	update()

2.4.2.2. Primitive support and unboxing

Drools 3.0.x did not had native support for primitive types and consequently, it auto-boxed all primitives in it's respective wrapper classes. That way, any use of a boxed variable binding required a manual unbox.

Drools 4.0.x has full support for primitive types and does not wrap values anymore. So, all previous unwrap method calls must be removed from the DRL.

Example 2.54. Drools 3.0.x manual unwrap

```
rule "Primitive int manual unbox"
when
   $c : Cheese( $price : price )
then
   $c.setPrice( $price.intValue() * 2 )
end
```

The above rule in 4.0.x would be:

Example 2.55. Drools 4.0.x primitive support

```
rule "Primitive support"
when
   $c : Cheese( $price : price )
then
   $c.setPrice( $price * 2 )
end
```

2.4.3. Drools Update Tool

The Drools Update tools is a simple program to help with the upgrade of DRL files from Drools 3.0.x to Drools 4.0.x.

At this point, its main objective is to upgrade the memory action calls from 3.0.x to 4.0.x, but expect it to grow over the next few weeks covering additional scenarios. It is important to note that it does

not make a dumb text search and replace in rules file, but it actually parses the rules file and try to make sure it is not doing anything unexpected, and as so, it is a safe tool to use for upgrade large sets of rule files.

The Drools update tool can be found as a maven project in the following source repository http://anonsvn.labs.jboss.com/labs/jbossrules/trunk/experimental/drools-update/ you just need to check it out, and execute the maven clean install action with the project's pom.xml file. After resolve all the class path dependencies you are able to run the toll with the following command:

```
java -cp $CLASSPATH org.drools.tools.update.UpdateTool -f <filemask> [-d <basedir>] [-s <sufix>]
```

The program parameters are very easy to understand as following.

- -h,--help, Shows a very simple list the usage help
- -d your source base directory
- -f pattern for the files to be updated. The format is the same as used by ANT: * = single file, directory ** = any level of subdirectories EXAMPLE: src/main/resources/**/*.drl = matches all DRL files inside any subdirectory of /src/main/resources
- -s,--sufix the sufix to be added to all updated files

2.4.4. DSL Grammars in Drools 4.0

It is important to note that the DSL template engine was rewritten from scratch to improve flexibility. One of the new features of DSL grammars is the support to Regular Expressions. This way, now you can write your mappings using regexp to have additional flexibility, as explained in the DSL chapter. Although, now you have to escape characters with regexp meaning. Example: if previously you had a matching like:

Example 2.56. Drools 3.0.x mapping

```
[when][]- the {attr} is in [ {values} ]={attr} in ( {values} )
```

Now, you need to escape '[' and ']' characters, as they have special meaning in regexps. So, the same mapping in Drools 4.0 would be:

Example 2.57. Drools 4.0.x mapping with escaped characters

```
[when][]- the {attr} is in \[ {values} \]={attr} in ( {values} )
```

2.4.5. Rule flow Update for 4.0.2

The Rule flow feature was updated for 4.0.2, and now all your ruleflows must declare a package name.

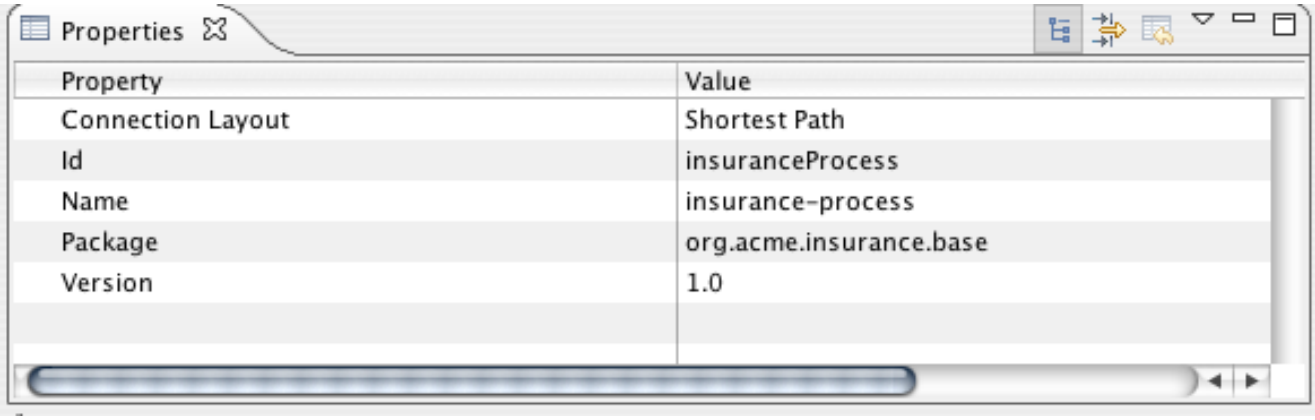


Figure 2.37. Rule Flow properties

Chapter 3. Installation and Setup (Core and IDE)

3.1. Installing and using

Drools provides an Eclipse-based IDE (which is optional), but at its core only Java 1.5 (Java SE) is required.

A simple way to get started is to download and install the Eclipse plug-in - this will also require the Eclipse GEF framework to be installed (see below, if you don't have it installed already). This will provide you with all the dependencies you need to get going: you can simply create a new rule project and everything will be done for you. Refer to the chapter on the Rule Workbench and IDE for detailed instructions on this. Installing the Eclipse plug-in is generally as simple as unzipping a file into your Eclipse plug-in directory.

Use of the Eclipse plug-in is not required. Rule files are just textual input (or spreadsheets as the case may be) and the IDE (also known as the Rule Workbench) is just a convenience. People have integrated the rule engine in many ways, there is no "one size fits all".

Alternatively, you can download the binary distribution, and include the relevant jars in your projects classpath.

3.1.1. Dependencies and jars

Drools is broken down into a few modules, some are required during rule development/compiling, and some are required at runtime. In many cases, people will simply want to include all the dependencies at runtime, and this is fine. It allows you to have the most flexibility. However, some may prefer to have their "runtime" stripped down to the bare minimum, as they will be deploying rules in binary form - this is also possible. The core runtime engine can be quite compact, and only require a few 100 kilobytes across 2 jar files.

The following is a description of the important libraries that make up JBoss Rules

- drools-api.jar - this provides the interfaces and factories. Drools-api also helps clearly show what is intended as a user api and what is just an engine api.
- drools-core.jar - this is the core engine, runtime component. Contains both the RETE engine and the LEAPS engine. This is the only runtime dependency if you are pre-compiling rules (and deploying via Package or RuleBase objects).
- drools-compiler.jar - this contains the compiler/builder components to take rule source, and build executable rule bases. This is often a runtime dependency of your application, but it need not be if you are pre-compiling your rules. This depends on drools-core

- drools-jsr94.jar - this is the JSR-94 compliant implementation, this is essentially a layer over the drools-compiler component. Note that due to the nature of the JSR-94 specification, not all features are easily exposed via this interface. In some cases, it will be easier to go direct to the Drools API, but in some environments the JSR-94 is mandated.
- drools-decisiontables.jar - this is the decision tables 'compiler' component, which uses the drools-compiler component. This supports both excel and CSV input formats.

There are quite a few other dependencies which the above components require, most of which are for the drools-compiler, drools-jsr94 or drools-decisiontables module. Some key ones to note are "POI" which provides the spreadsheet parsing ability, and "antlr" which provides the parsing for the rule language itself.

NOTE: if you are using Drools in J2EE or servlet containers and you come across classpath issues with "JDT", then you can switch to the janino compiler. Set the system property "drools.compiler": For example: -Ddrools.compiler=JANINO.

For up to date info on dependencies in a release, consult the README_DEPENDENCIES.txt file, which can be found in the lib directory of the download bundle, or in the root of the project directory.

3.1.2. Runtime

The "runtime" requirements mentioned here are if you are deploying rules as their binary form (either as KnowledgePackage objects, or KnowledgeBase objects etc). This is an optional feature that allows you to keep your runtime very light. You may use drools-compiler to produce rule packages "out of process", and then deploy them to a runtime system. This runtime system only requires drools-core.jar and drools-api for execution. This is an optional deployment pattern, and many people do not need to "trim" their application this much, but it is an ideal option for certain environments.

3.1.3. Installing IDE (Rule Workbench)

The rule workbench (for Eclipse) requires that you have Eclipse 3.4 or greater, as well as Eclipse GEF 3.4 or greater. You can install it either by downloading the plug-in or, or using the update site.

Another option is to use the JBoss IDE, which comes with all the plug-in requirements pre packaged, as well as a choice of other tools separate to rules. You can choose just to install rules from the "bundle" that JBoss IDE ships with.

3.1.3.1. Installing GEF (a required dependency)

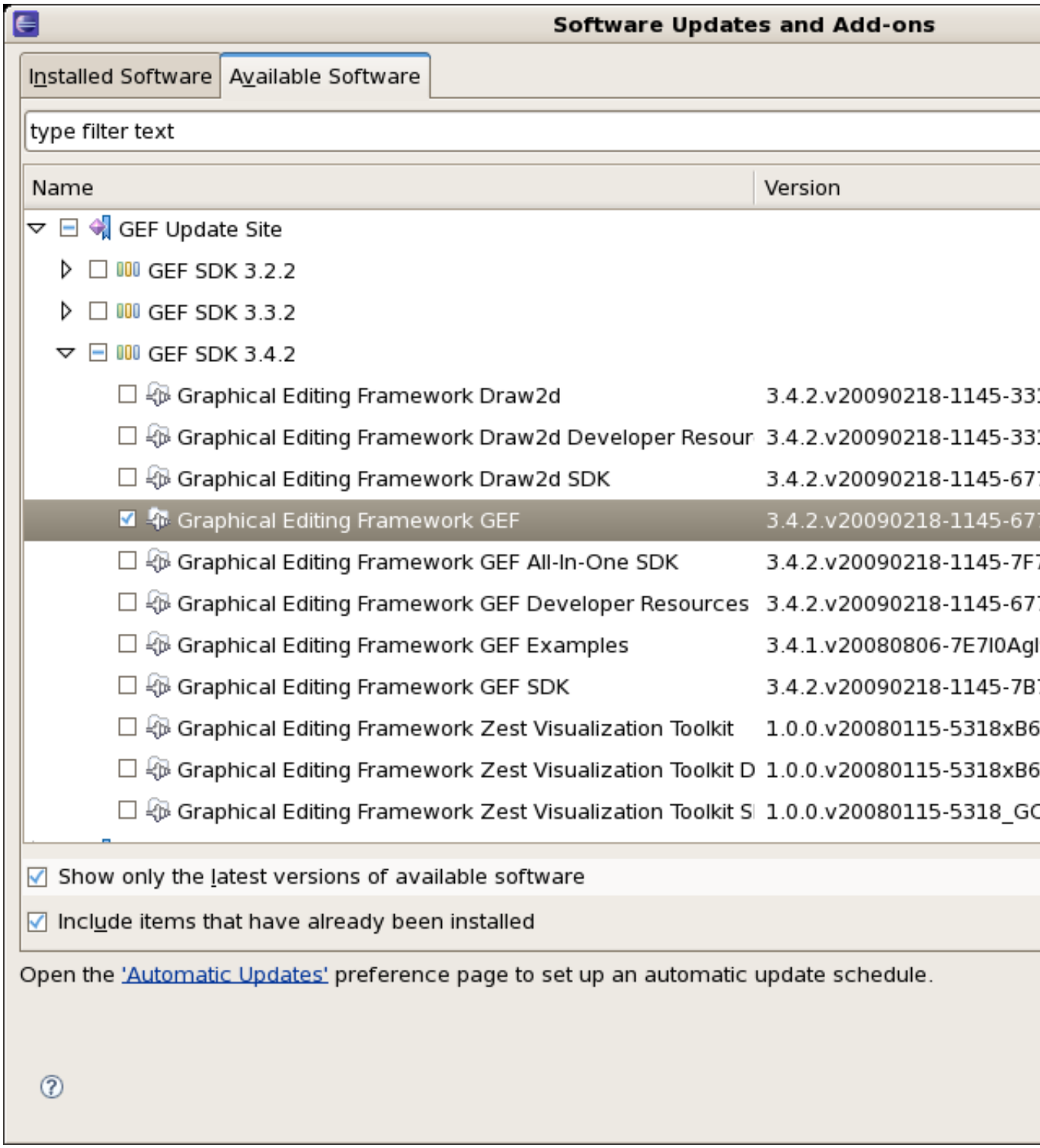
GEF is the Eclipse Graphical Editing Framework, which is used for graph viewing components in the plug-in.

If you don't have GEF installed, you can install it using the built in update mechanism (or downloading GEF from the Eclipse.org website not recommended). JBoss IDE has GEF already, as do many other "distributions" of Eclipse, so this step may be redundant for some people.

Open the Help->Software updates...->Available Software->Add Site... from the help menu. Location is:

```
http://download.eclipse.org/tools/gef/updates/releases/
```

Next you choose the GEF plug-in:



Press next, and agree to install the plug-in (an Eclipse restart may be required). Once this is completed, then you can continue on installing the rules plug-in.

3.1.3.2. Installing GEF from zip file

To install from the zip file, download and unzip the file. Inside the zip you will see a plug-in directory, and the plug-in jar itself. You place the plug-in jar into your Eclipse applications plug-in directory, and restart Eclipse.

3.1.3.3. Installing Drools plug-in from zip file

Download the Drools Eclipse IDE plugin from the link below. Unzip the downloaded file in your main eclipse folder (do not just copy the file there, extract it so that the feature and plugin jars end up in the features and plugin directory of eclipse) and (re)start Eclipse.

http://www.jboss.org/drools/downloads.html

To check that the installation was successful, try opening the Drools perspective: Click the 'Open Perspective' button in the top right corner of your Eclipse window, select 'Other...' and pick the Drools perspective. If you cannot find the Drools perspective as one of the possible perspectives, the installation probably was unsuccessful. Check whether you executed each of the required steps correctly: Do you have the right version of Eclipse (3.4.x)? Do you have Eclipse GEF installed (check whether the org.eclipse.gef_3.4.*.jar exists in the plugins directory in your eclipse root folder)? Did you extract the Drools Eclipse plugin correctly (check whether the org.drools.eclipse_*.jar exists in the plugins directory in your eclipse root folder)? If you cannot find the problem, try contacting us (e.g. on irc or on the user mailing list), more info can be found no our homepage here:

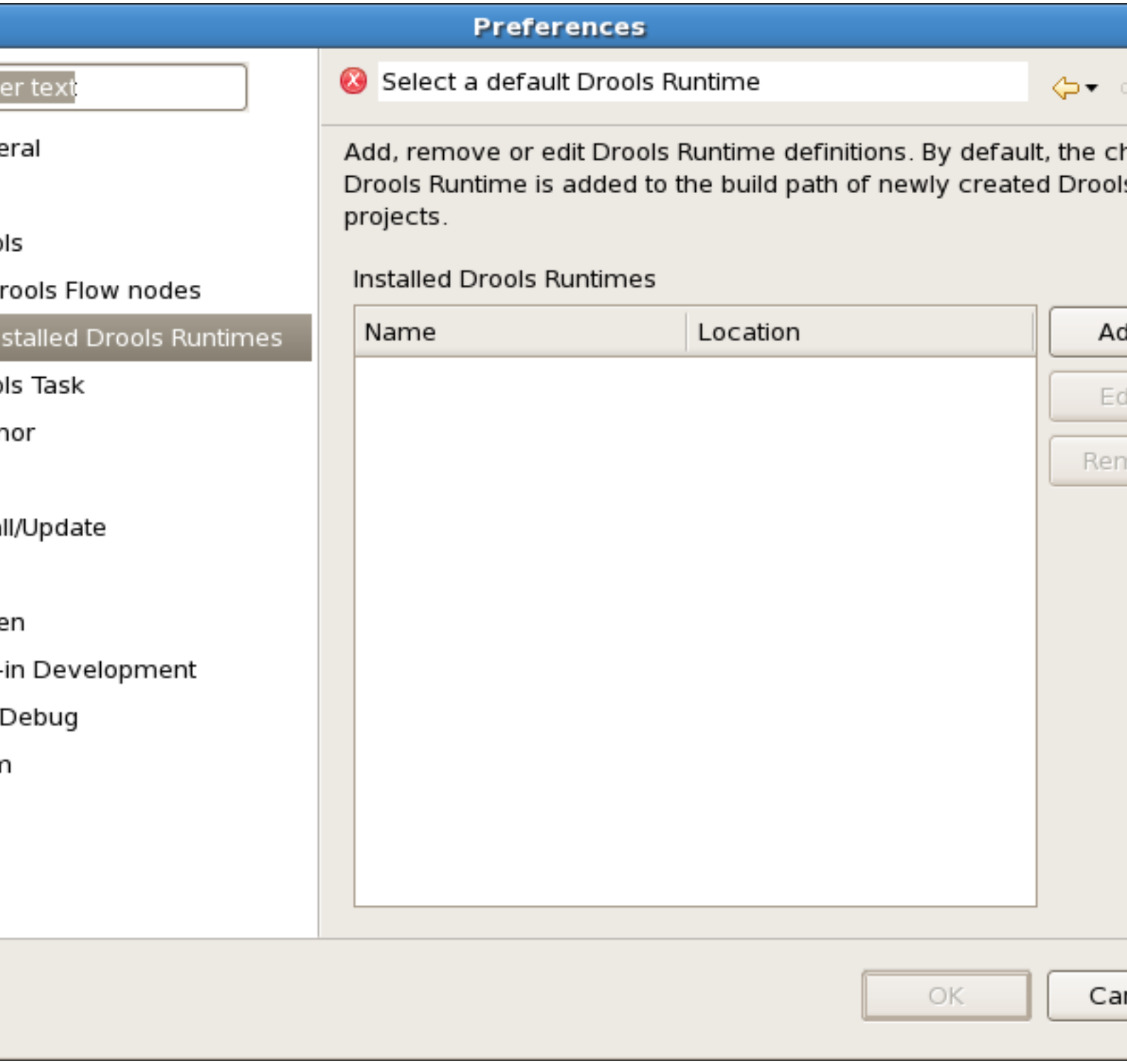
http://www.jboss.org/drools/

3.1.3.4. Drools Runtimes

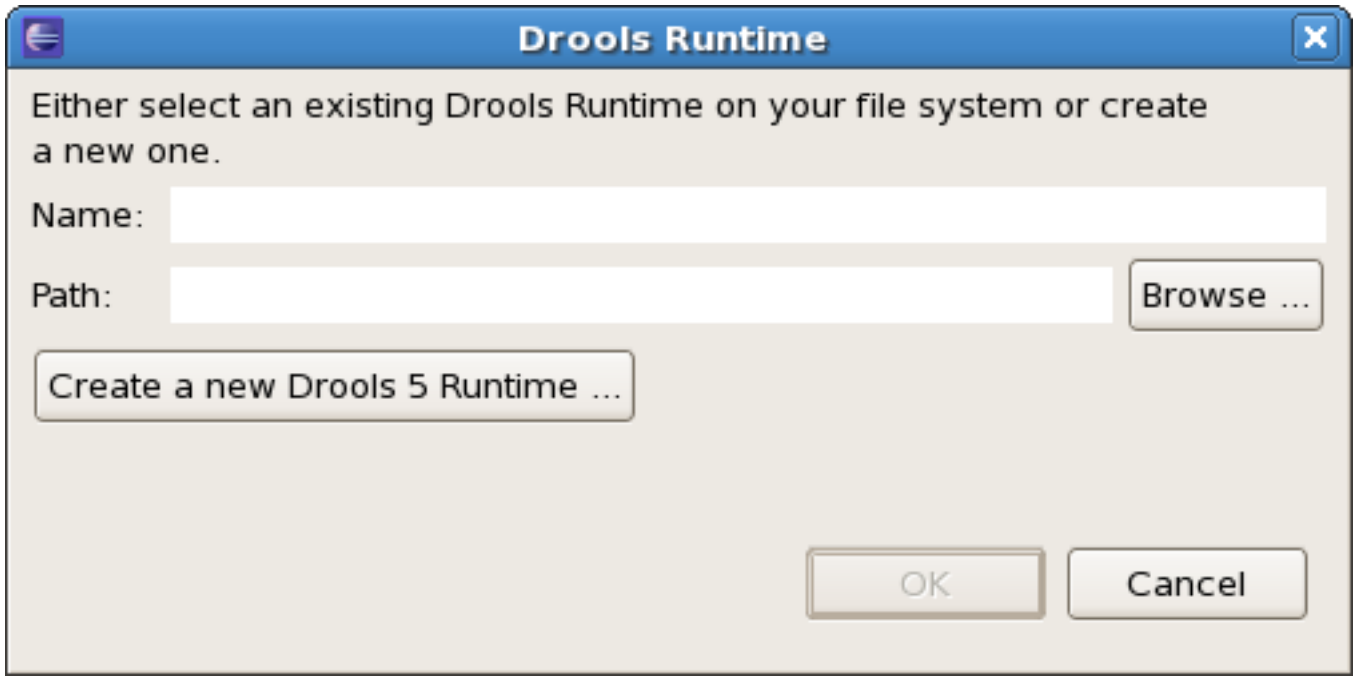
A Drools runtime is a collection of jars on your file system that represent one specific release of the Drools project jars. To create a runtime, you must point the IDE to the release of your choice. If you want to create a new runtime based on the latest Drools project jars included in the plugin itself, you can also easily do that. You are required to specify a default Drools runtime for your Eclipse workspace, but each individual project can override the default and select the appropriate runtime for that project specifically.

3.1.3.4.1. Defining a Drools runtime

You are required to define one or more Drools runtimes using the Eclipse preferences view. To open up your preferences, in the menu Window select the Preferences menu item. A new preferences dialog should show all your preferences. On the left side of this dialog, under the Drools category, select "Installed Drools runtimes". The panel on the right should then show the currently defined Drools runtimes. If you have not yet defined any runtimes, it should like something like the figure below.



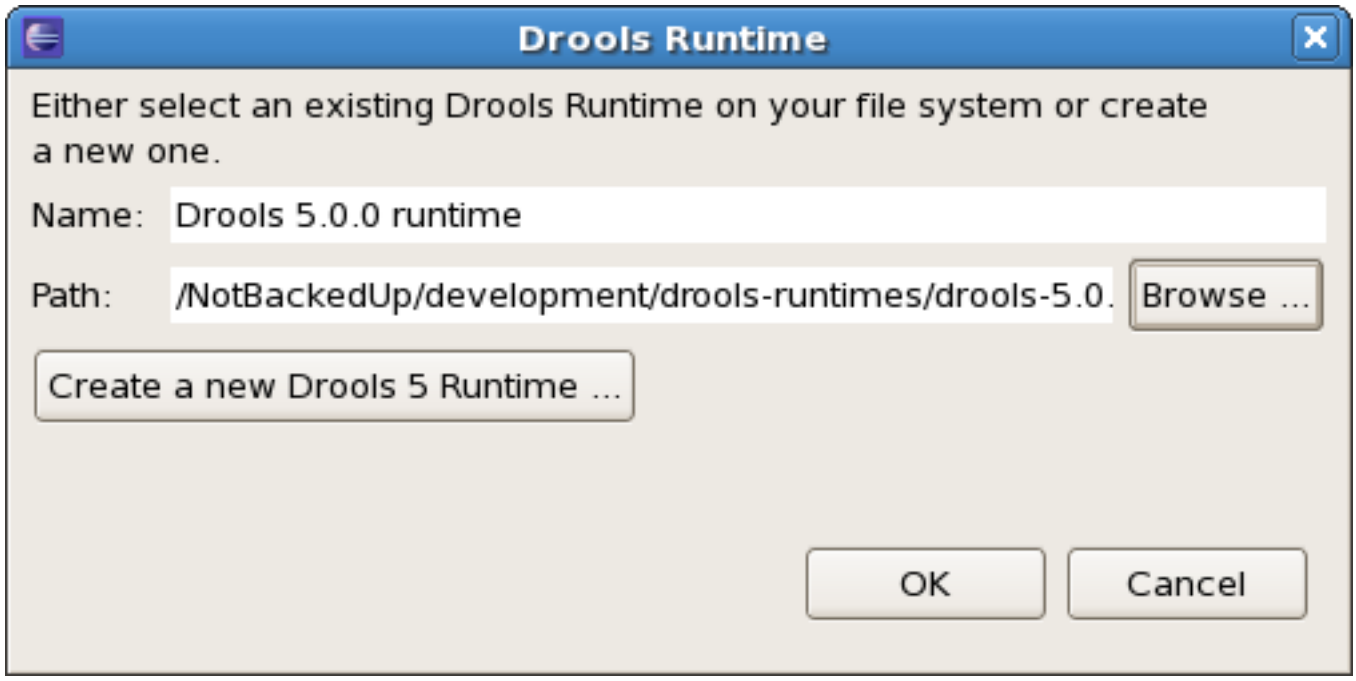
To define a new Drools runtime, click on the add button. A dialog as shown below should pop up, requiring the name for your runtime and the location on your file system where it can be found.



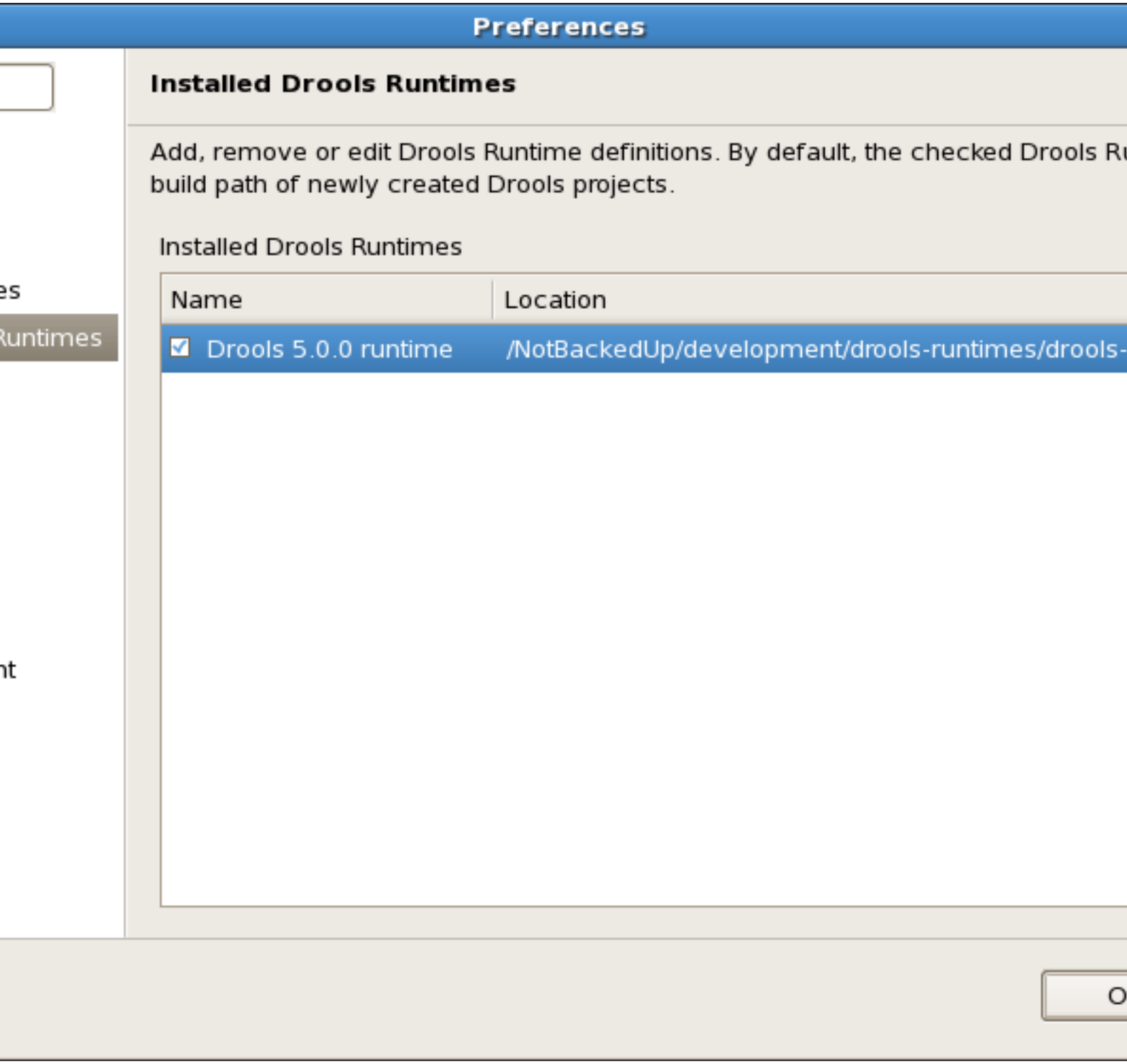
In general, you have two options:

1. If you simply want to use the default jars as included in the Drools Eclipse plugin, you can create a new Drools runtime automatically by clicking the "Create a new Drools 5 runtime ..." button. A file browser will show up, asking you to select the folder on your file system where you want this runtime to be created. The plugin will then automatically copy all required dependencies to the specified folder. After selecting this folder, the dialog should look like the figure shown below.

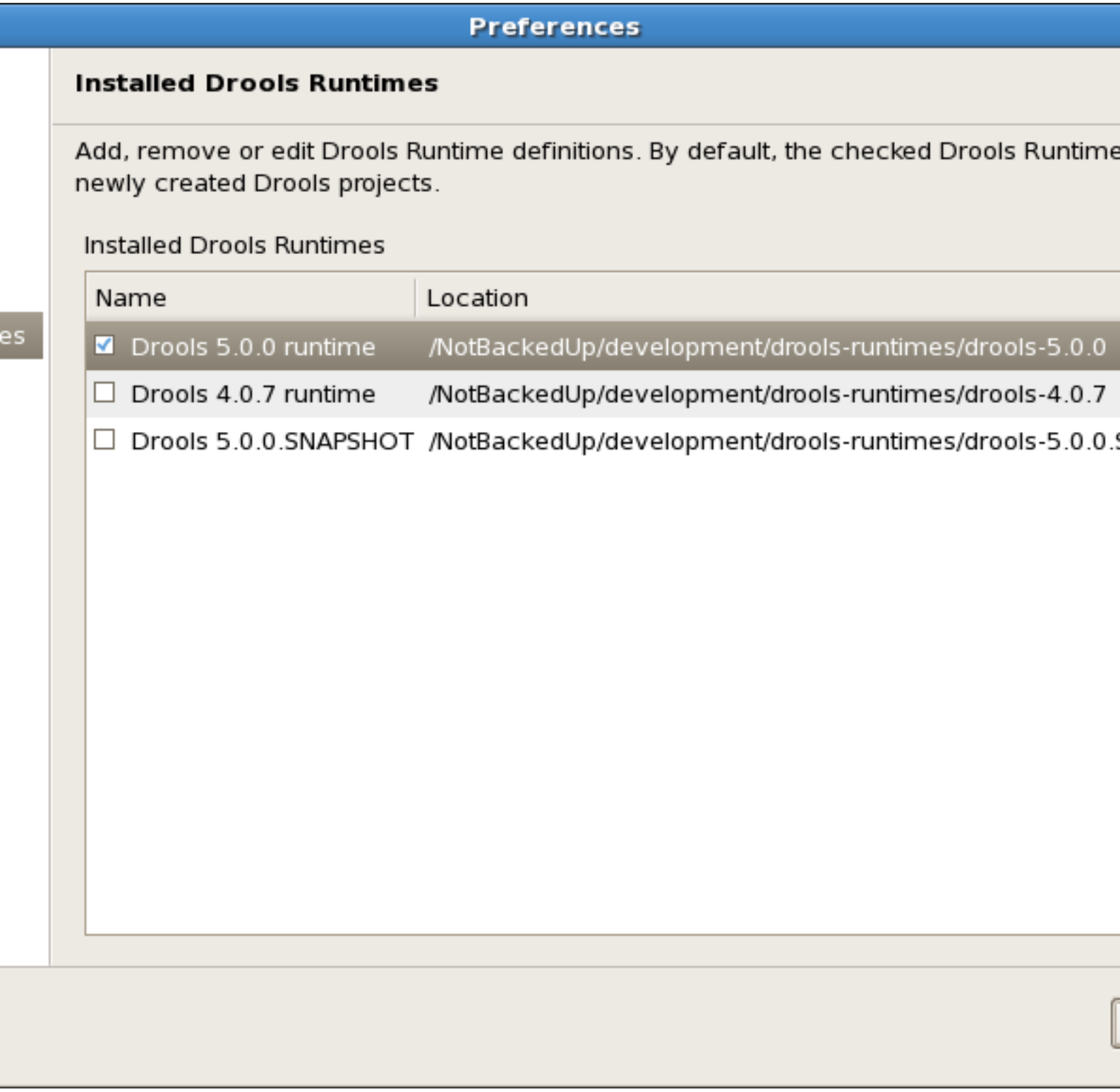
2. If you want to use one specific release of the Drools project, you should create a folder on your file system that contains all the necessary Drools libraries and dependencies. Instead of creating a new Drools runtime as explained above, give your runtime a name and select the location of this folder containing all the required jars.



After clicking the OK button, the runtime should show up in your table of installed Drools runtimes, as shown below. Click on checkbox in front of the newly created runtime to make it the default Drools runtime. The default Drools runtime will be used as the runtime of all your Drools project that have not selected a project-specific runtime.



You can add as many Drools runtimes as you need. For example, the screenshot below shows a configuration where three runtimes have been defined: a Drools 4.0.7 runtime, a Drools 5.0.0 runtime and a Drools 5.0.0.SNAPSHOT runtime. The Drools 5.0.0 runtime is selected as the default one.

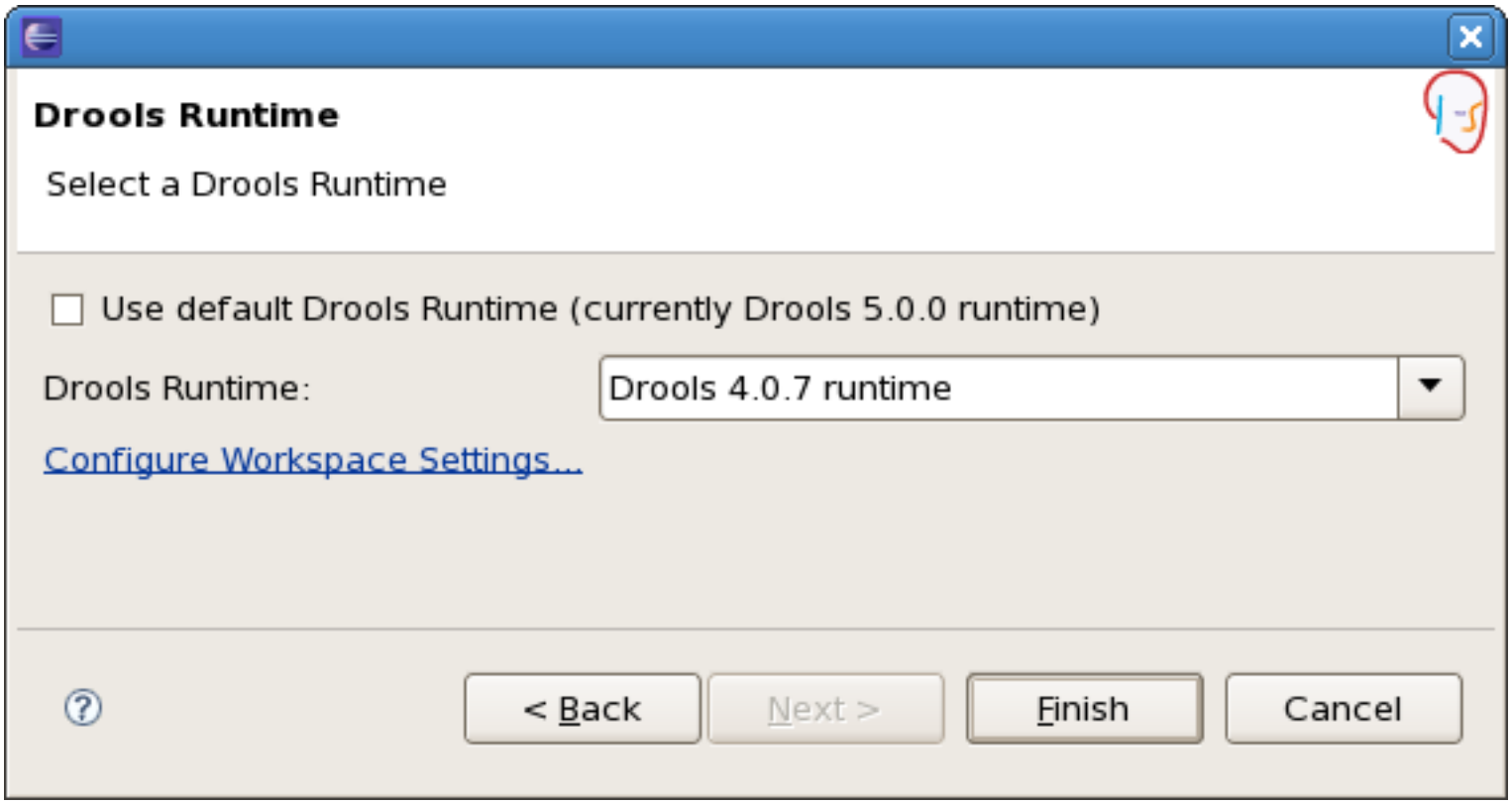


Note that you will need to restart Eclipse if you changed the default runtime and you want to make sure that all the projects that are using the default runtime update their classpath accordingly.

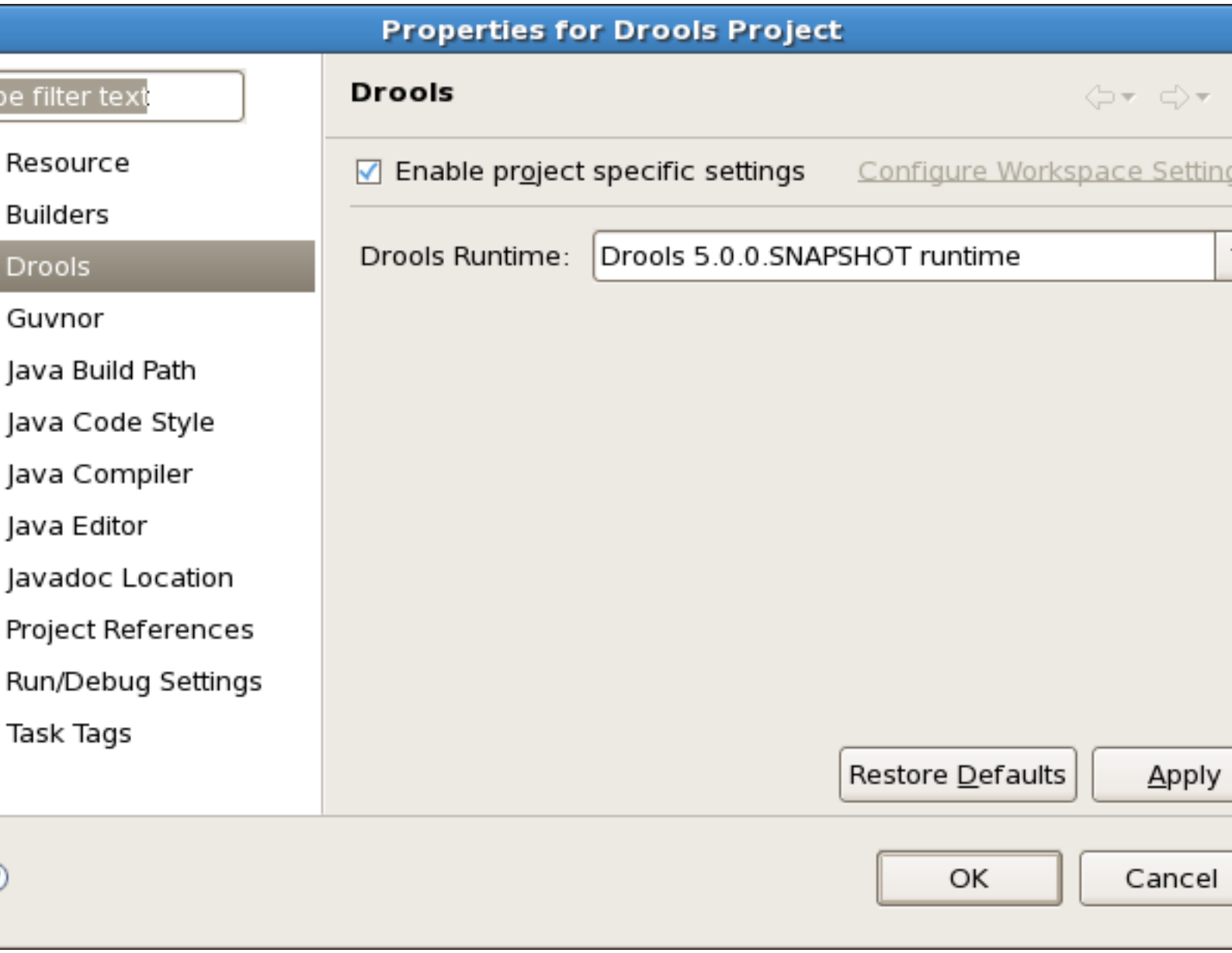
3.1.3.4.2. Selecting a runtime for your Drools project

Whenever you create a Drools project (using the New Drools Project wizard or by converting an existing Java project to a Drools project using the "Convert to Drools Project" action that is shown when you are in the Drools perspective and you right-click an existing Java project), the plugin will automatically add all the required jars to the classpath of your project.

When creating a new Drools project, the plugin will automatically use the default Drools runtime for that project, unless you specify a project-specific one. You can do this in the final step of the New Drools Project wizard, as shown below, by deselecting the "Use default Drools runtime" checkbox and selecting the appropriate runtime in the drop-down box. If you click the "Configure workspace settings ..." link, the workspace preferences showing the currently installed Drools runtimes will be opened, so you can add new runtimes there.



You can change the runtime of a Drools project at any time by opening the project properties (right-click the project and select Properties) and selecting the Drools category, as shown below. Check the "Enable project specific settings" checkbox and select the appropriate runtime from the drop-down box. If you click the "Configure workspace settings ..." link, the workspace preferences showing the currently installed Drools runtimes will be opened, so you can add new runtimes there. If you deselect the "Enable project specific settings" checkbox, it will use the default runtime as defined in your global preferences.



3.2. Setup from source

As Drools is an open source project, instructions for building from source are part of the manual ! Building from source means you can stay on top with the latest features. Whilst aspects of Drools are quite complicated, many users have found ways to become contributors.

Drools works with JDK1.5 and above. you will need also need to have the following tools installed. Minimum requirement version numbers provided.

- Eclipse 3.4

  http://www.eclipse.org/

- Subversion Client 1.4

http://subversion.tigris.org

http://tortoisesvn.tigris.org - recommended win32 client

- Maven 2.0.9

  http://maven.apache.org/

- Ant 1.7.0

  http://ant.apache.org

Ensure the executables for ant, maven and java are in your path. The examples given illustrative and are for a win32 system:

```
Path=D:\java\j2sdk1.5.0_8\bin;D:\java\apache-ant-1.7\bin;D:\java\maven-2.0.9\bin
```

Following environment variables will also need to be set. The examples given illustrative and are for a win32 system::

```
JAVA_HOME=D:\java\j2sdk1.5.0_8
ANT_HOME=D:\java\apache-ant-1.7
MAVEN_HOME=D:\java\maven-2.0.9
```

Past releases used to have an ant based build mechanism, but now maven is mandatory, although Ant is used internally in maven for document building proposes

3.3. Source Checkout

Drools is available from two Subversion repositories.

- Anonymous SVN

  http://anonsvn.jboss.org/repos/labs/labs/jbossrules/trunk/

- Developers secured SVN

  https://svn.jboss.org/repos/labs/labs/jbossrules/trunk/

To checkout Drools source code just execute the following command.

```
fmeyer:~/jboss $ svn checkout http://anonsvn.jboss.org/repos/labs/labs/jbossrules/trunk/ trunk
```

And wait to complete the files download.

```
A    trunk/drools-repository
```

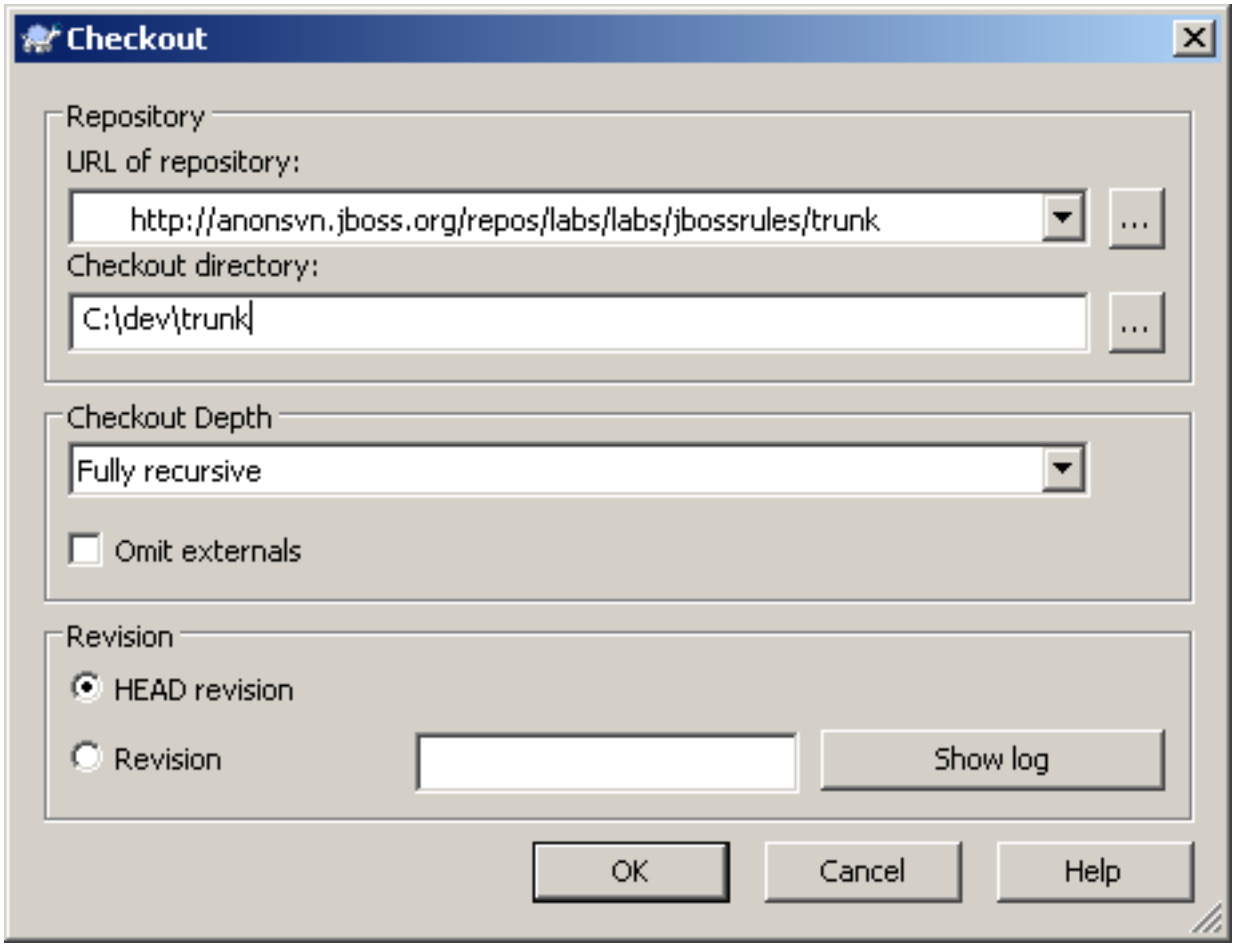
```
A    trunk/drools-repository/.classpath
A    trunk/drools-repository/.project
A    trunk/drools-repository/doc
A    trunk/drools-repository/doc/repository_layout.jpeg
A    trunk/drools-repository/doc/high_level_design.jpeg
A    trunk/drools-repository/doc/javadoc
A    trunk/drools-repository/doc/javadoc/serialized-form.html
A    trunk/drools-repository/doc/javadoc/index-all.html
A    trunk/drools-repository/doc/javadoc/stylesheet.css
A    trunk/drools-repository/doc/javadoc/allclasses-frame.html
A    trunk/drools-repository/doc/javadoc/package-list
A    trunk/drools-repository/doc/javadoc/overview-tree.html
A    trunk/drools-repository/doc/javadoc/org
A    trunk/drools-repository/doc/javadoc/org/drools
A    trunk/drools-repository/doc/javadoc/org/drools/repository
A    trunk/drools-repository/doc/javadoc/org/drools/repository/class-use
A    trunk/drools-repository/doc/javadoc/org/drools/repository/class-use/RuleSet.html
A                    trunk/drools-repository/doc/javadoc/org/drools/repository/class-use/
RulesRepositoryException.html
A    trunk/drools-repository/doc/javadoc/org/drools/repository/class-use/RulesRepository.html
A    trunk/drools-repository/doc/javadoc/org/drools/repository/RuleSet.html

....

snip

....

A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/benchmark/waltz
A          trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/benchmark/waltz/
waltz.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/benchmark/manners
A      trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/benchmark/manners/
manners.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/benchmark/waltzdb
A        trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/benchmark/waltzdb/
waltzdb.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples
A                  trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/
TroubleTicketWithDSL.drl
A                  trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/
TroubleTicket.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway
```

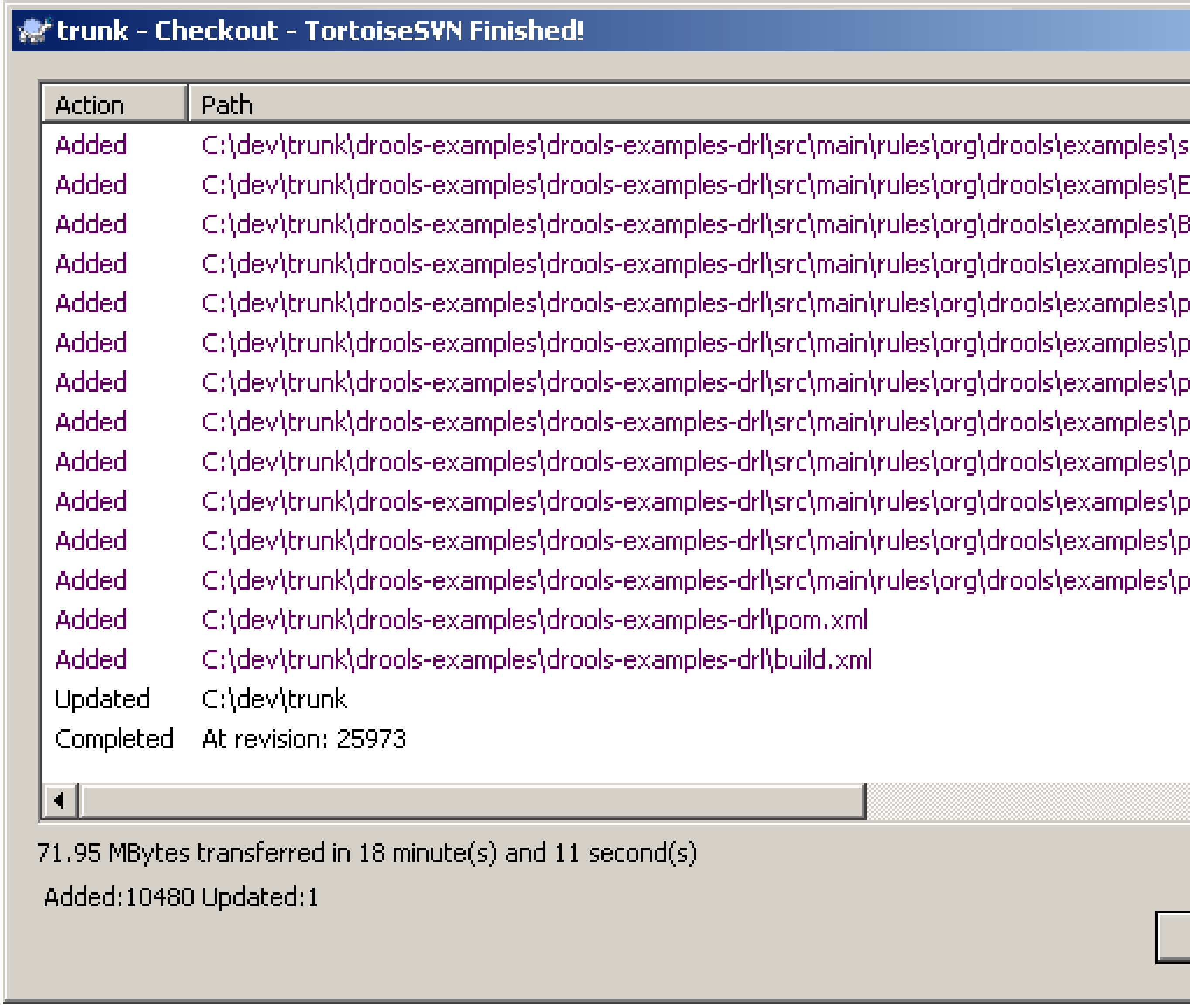
```
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/
calculate.rfm
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/
generation.rf
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/
calculate.rf
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/
registerNeighbor.rfm
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/
killAll.rfm
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/
registerNeighbor.rf
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/
conway-agendagroup.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/
killAll.rf
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/
conway-ruleflow.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/conway/
generation.rfm
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/ticketing.dsl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/
StateExampleUsingSalience.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/golf.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/
LogicalAssertionsNotPingPong.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/
StateExampleDynamicRule.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/sudoku
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/sudoku/
sudoku.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/
HelloWorld.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/
ExamplePolicyPricing.xls
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/
HonestPolitician.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/Fibonacci.drl
A    trunk/drools-examples/drools-examples-drl/src/main/rules/org/drools/examples/
StateExampleUsingAgendGroup.drl
A    trunk/drools-examples/drools-examples-drl/pom.xml
A    trunk/drools-examples/drools-examples-drl/build.xml
U    trunk
```

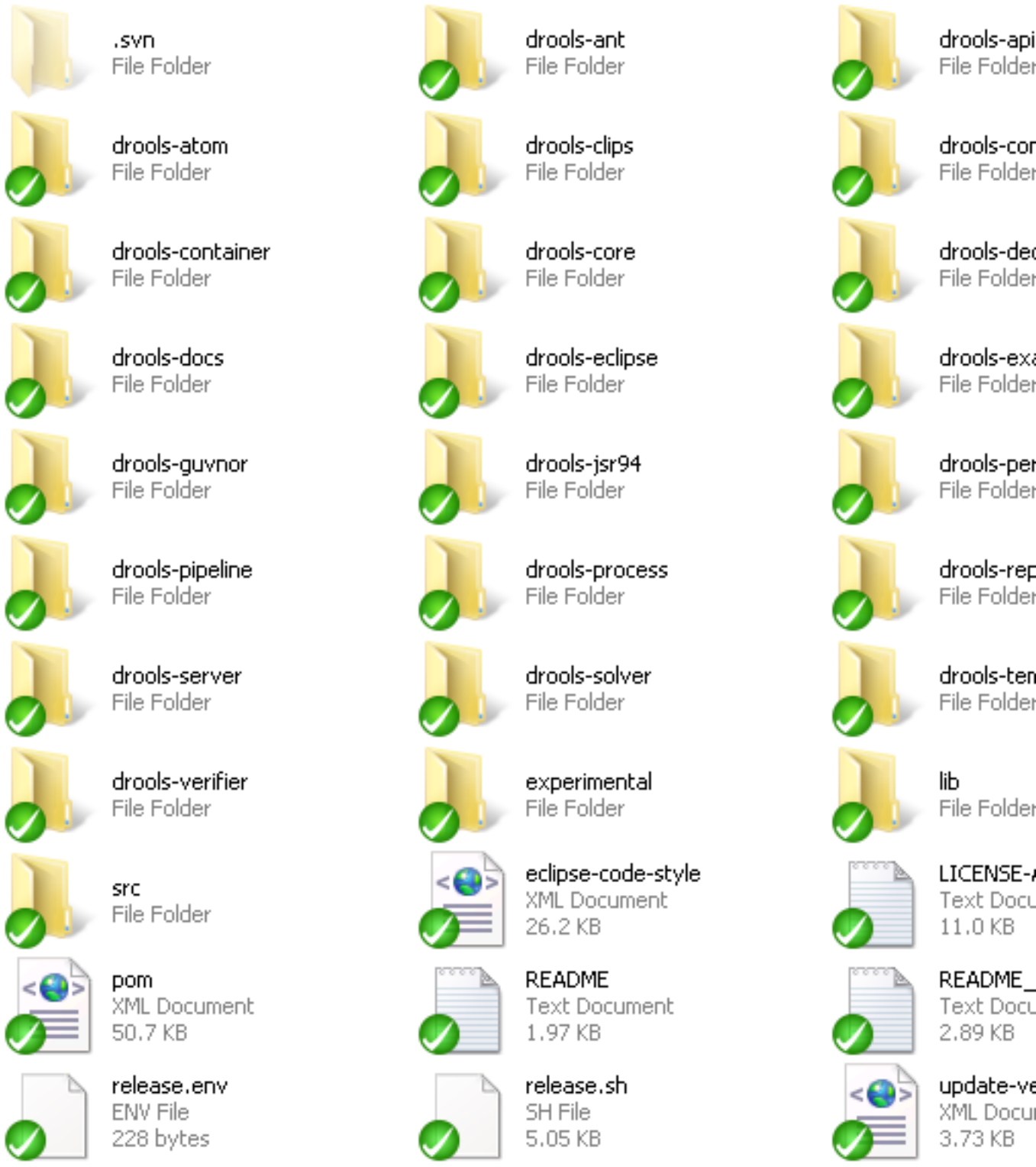
```
Checked out revision 13656.
```

Although, we highly recommend command line tools to work with repository you can also use both Eclipse's integrated SVN client or TortoiseSVN

Setup TortoiseSVN to checkout from the subversion repository and click OK. Once the checkout has finished you should see the folders as shown below.




trunk - Checkout - TortoiseSVN Finished!
Action
Path
Added
C:\dev\trunk\drools-examples\drools-examples-drl\src\main\rules\org\drools\examples\
Added
C:\dev\trunk\drools-examples\drools-examples-drl\pom.xml
Added
C:\dev\trunk\drools-examples\drools-examples-drl\build.xml
Updated
C:\dev\trunk
Completed
At revision: 25973
71.95 MBytes transferred in 18 minute(s) and 11 second(s)
Added:10480 Updated:1


.svn
File Folder
drools-ant
File Folder
drools-api
File Folder
drools-atom
File Folder
drools-clips
File Folder
drools-con
File Folder
drools-container
File Folder
drools-core
File Folder
drools-doc
File Folder
drools-docs
File Folder
drools-eclipse
File Folder
drools-exa
File Folder
drools-guvnor
File Folder
drools-jsr94
File Folder
drools-per
File Folder
drools-pipeline
File Folder
drools-process
File Folder
drools-rep
File Folder
drools-server
File Folder
drools-solver
File Folder
drools-ten
File Folder
drools-verifier
File Folder
experimental
File Folder
lib
File Folder
src
File Folder
eclipse-code-style
XML Document
26.2 KB
LICENSE-A
Text Docu
11.0 KB
pom
XML Document
50.7 KB
README
Text Document
1.97 KB
README_
Text Docu
2.89 KB
release.env
ENV File
228 bytes
release.sh
SH File
5.05 KB
update-ve
XML Docu
3.73 KB

3.4. Build

3.4.1. Building the Source

Now that we have the source the next step is to build and install the source. Since version 3.1 Drools uses Maven 2 to build the system. There are two profiles available which enable the associated modules "documentation" and "Eclipse"; this enables quicker building of the core modules for developers. The Eclipse profile will download Eclipse into the drools-Eclipse folder, which is over 100MB download (It depends on your operating system), however this only needs to be done once; if you wish you can move that Eclipse download into another location and specify it with -DlocalEclipseDrop=/folder/jboss-drools/local-Eclipse-drop-mirror. The following builds all the jars, the documentation and the Eclipse zip with a local folder specified to avoid downloading Eclipse:

```
mvn -Declipse -Ddocumentation clean install -DlocalEclipseDrop=/folder/jboss-drools/local-Eclipse-drop-mirror
```

You can produce distribution builds, which puts everything into zips, as follows:

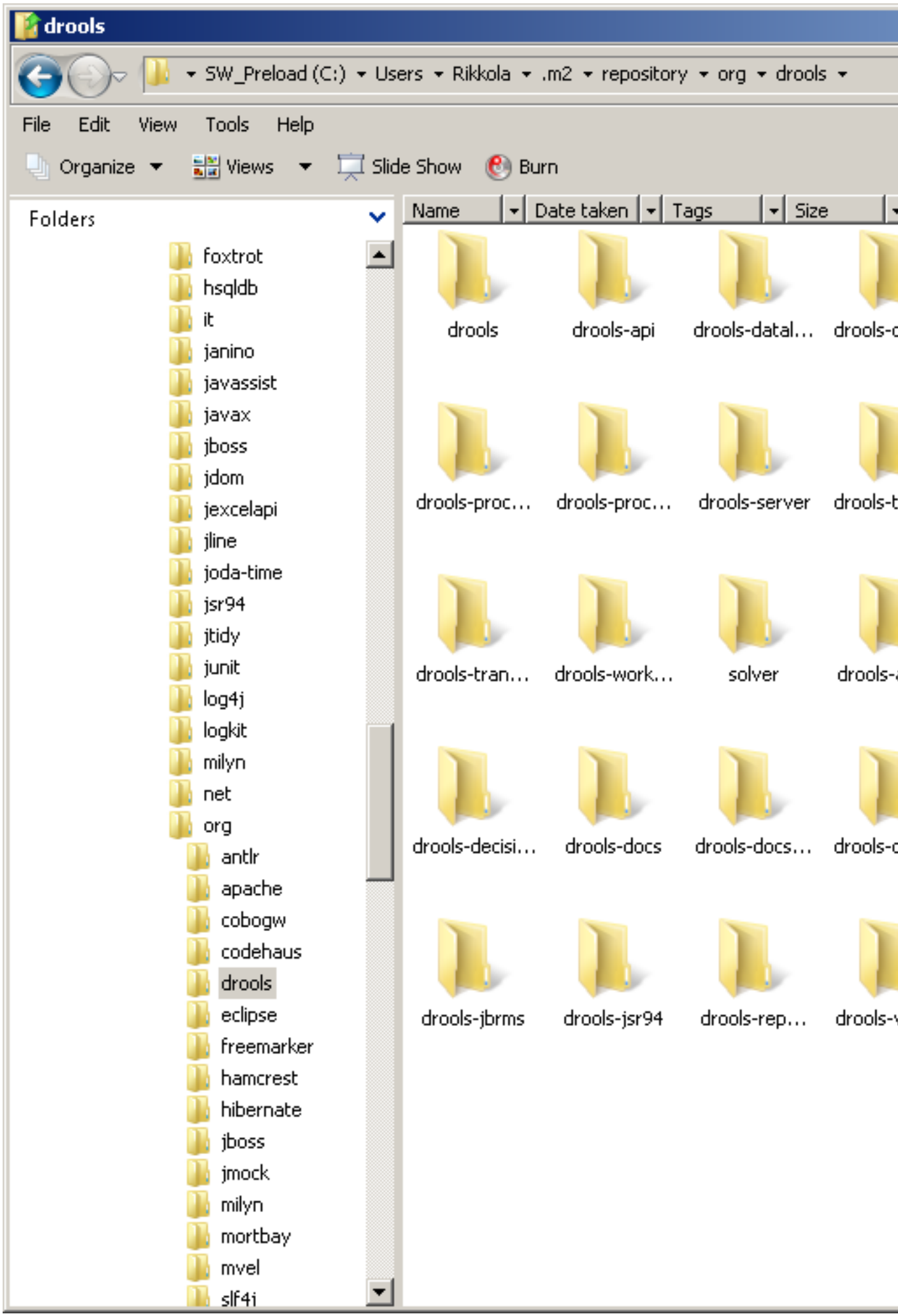
```
mvn -Declipse -Ddocumentation clean install -DlocalEclipseDrop=/folder/jboss-drools/local-Eclipse-drop-mirror
mvn -Ddocumentation -Declipse -DskipTests package javadoc:javadoc assembly:assembly -DlocalEclipseDrop=/folder/jboss-drools/local-Eclipse-drop-mirror
```

Note that install must be done first as javadoc:javadoc won't work unless the jars are in the local maven repo, but the tests can be skipped on the second run. assembly:assembly fails unless you increase the available memory to Maven, on windows the following command worked well: set MAVEN_OPTS=-Xmx512m

Type `mvn clean` to clear old artifacts, and then test and built the source, and report on any errors.

The resulting jars are put in the /target directory from the top level of the project.

As maven builds each module it will install the resulting jars in the local Maven 2 repository automatically. Where it can be easily used from other project `pom.xml` or copied else where.


drools
SW_Preload (C:) ▸ Users ▸ Rikkola ▸ .m2 ▸ repository ▸ org ▸ drools
File Edit View Tools Help
Organize Views Slide Show Burn
Folders
foxtrot
hsqldb
it
janino
javassist
javax
jboss
jdom
jexcelapi
jline
joda-time
jsr94
jtidy
junit
log4j
logkit
milyn
net
org
antlr
apache
cobogw
codehaus
drools
eclipse
freemarker
hamcrest
hibernate
jboss
jmock
milyn
mortbay
mvel
slf4j
Name Date taken Tags Size
drools
drools-api
drools-datal...
drools-proc...
drools-proc...
drools-server
drools-tran...
drools-work...
solver
drools-decisi...
drools-docs
drools-docs...
drools-jbrms
drools-jsr94
drools-rep...

3.4.2. Building the Manual

The building of the manual is now integrated into the maven build process, and is built by either using the profile (-Ddocumentation) switch or cding into the main directory.

Drools uses Docbook for this manual. Maven is used to build documents and this build produces three different formats, all sharing the same images directory.

- html_single

  The entire manual in a single html document

- html

  The manual is split into multiple documents and placed in a frameset. The left frame provides navigation

- Eclipse

  Documentation suitable for including in an Eclipse plug-in

The manual can be generated from the project `pom.xml` by calling `mvn package` in the drools-docs directory or adding the -Ddocumentation switch when you build the sources. Documentation is generated into each drools-docs subdirectory's `target/` directory. Running `mvn -Ddocumentation package assembly:assembly` in the Drools project root generates and copies the documentation into a zip file. This zip file is located in the root folders `target/` directory.

```
[trikkola@trikkola trunk]$ mvn -Ddocumentation clean package assembly:assembly
[INFO] Scanning for projects...
[INFO] Reactor build order:
[INFO]   Drools
[INFO]   Drools :: API
[INFO]   Drools :: Core
[INFO]   Drools :: Compiler
[INFO]   Drools :: Templates
[INFO]   Drools :: Decision Tables
[INFO]   Drools :: JSR-94 API Module
[INFO]   Drools :: Pipeline :: Transformer :: Smooks
[INFO]   Drools :: Pipeline :: Transformer :: JAXB
[INFO]   Drools :: Pipeline :: Transformer :: XStream
[INFO]   Drools :: Pipeline :: Transformer :: JXLS
[INFO]   Drools :: Pipeline :: Messenger :: JMS
[INFO]   Drools :: Pipeline
[INFO]   Drools :: Process :: WorkItems
[INFO]   Drools :: Process :: Task
[INFO]   Drools :: Process :: BAM
[INFO]   Drools :: Process
```

```
[INFO]   Drools :: Persistence :: JPA
[INFO]   Drools :: Server
[INFO]   Drools :: Verifier
[INFO]   Drools :: Ant Task
[INFO]   Drools :: Repository
[INFO]   Drools :: Guvnor
[INFO]   Drools :: Microcontainer
[INFO]   Drools :: Clips
[INFO]   Drools :: Planner parent
[INFO]   Drools :: Planner core
[INFO]   Drools :: Planner examples
[INFO] Searching repository for plugin with prefix: 'assembly'.
WAGON_VERSION: 1.0-beta-2
[INFO] ------------------------------------------------------------------------
[INFO] Building Drools
[INFO]    task-segment: [clean, package]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean]
[INFO] [site:attach-descriptor]
[INFO] Preparing source:jar
[WARNING] Removing: jar from forked lifecycle, to prevent recursive invocation.
[INFO] No goals needed for project - skipping
[INFO] [source:jar {execution: default}]
[INFO] Preparing source:test-jar
[WARNING] Removing: jar from forked lifecycle, to prevent recursive invocation.
[WARNING] Removing: test-jar from forked lifecycle, to prevent recursive invocation.
[INFO] No goals needed for project - skipping
[INFO] [source:test-jar {execution: default}]
[INFO] ------------------------------------------------------------------------
[INFO] Building Drools :: API
[INFO]    task-segment: [clean, package]
[INFO] ------------------------------------------------------------------------
[INFO] [clean:clean]
[INFO] Deleting directory /home/trikkola/jboss-drools/trunk/drools-api/target

...snip ...

[INFO]
[INFO] ------------------------------------------------------------------------
[INFO] Reactor Summary:
[INFO] ------------------------------------------------------------------------
[INFO] Drools ..................................... SUCCESS [59.889s]
[INFO] Drools :: API .............................. SUCCESS [4.832s]
[INFO] Drools :: Core ............................. SUCCESS [11.027s]
```

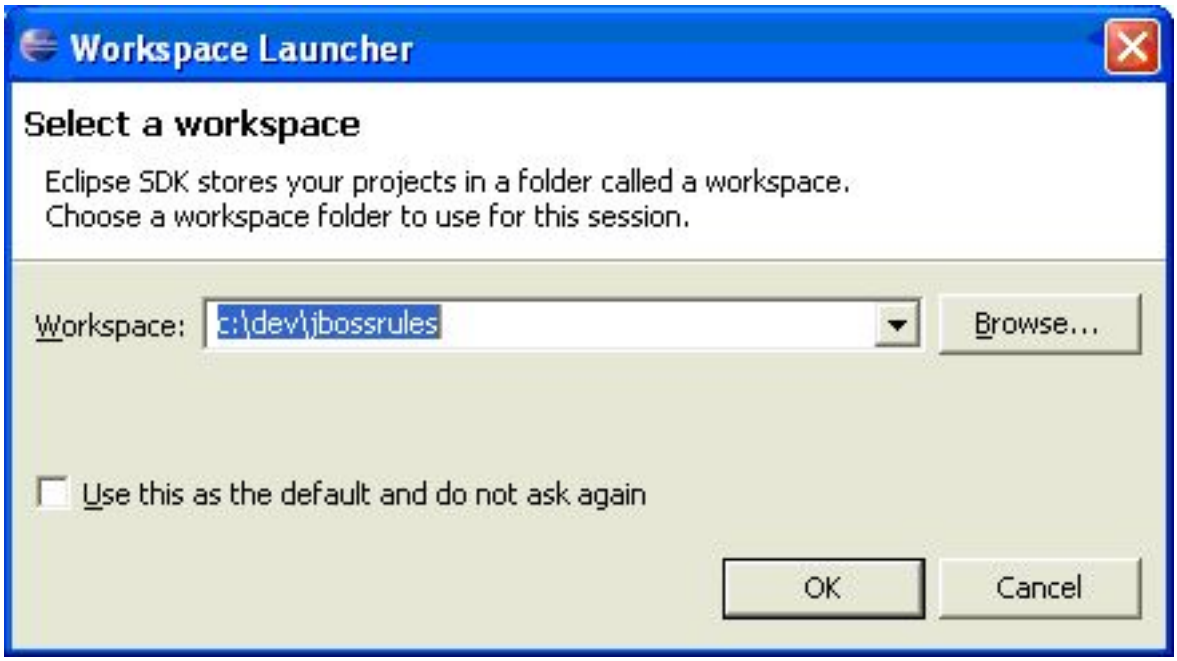
```
[INFO] Drools :: Compiler ..... SUCCESS [10.400s]
[INFO] Drools :: Templates ..... SUCCESS [1.018s]
[INFO] Drools :: Decision Tables ..... SUCCESS [1.179s]
[INFO] Drools :: JSR-94 API Module ..... SUCCESS [1.001s]
[INFO] Drools :: Pipeline :: Transformer :: Smooks ..... SUCCESS [0.651s]
[INFO] Drools :: Pipeline :: Transformer :: JAXB ..... SUCCESS [0.711s]
[INFO] Drools :: Pipeline :: Transformer :: XStream ..... SUCCESS [0.465s]
[INFO] Drools :: Pipeline :: Transformer :: JXLS ..... SUCCESS [0.481s]
[INFO] Drools :: Pipeline :: Messenger :: JMS ..... SUCCESS [0.879s]
[INFO] Drools :: Pipeline ..... SUCCESS [0.006s]
[INFO] Drools :: Process :: WorkItems ..... SUCCESS [1.526s]
[INFO] Drools :: Process :: Task ..... SUCCESS [3.104s]
[INFO] Drools :: Process :: BAM ..... SUCCESS [0.580s]
[INFO] Drools :: Process ..... SUCCESS [0.005s]
[INFO] Drools :: Persistence :: JPA ..... SUCCESS [0.958s]
[INFO] Drools :: Server ..... SUCCESS [2.216s]
[INFO] Drools :: Verifier ..... SUCCESS [1.836s]
[INFO] Drools :: Ant Task ..... SUCCESS [0.722s]
[INFO] Drools :: Repository ..... SUCCESS [3.925s]
[INFO] Drools :: Guvnor ..... SUCCESS [19.850s]
[INFO] Drools :: Microcontainer ..... SUCCESS [0.676s]
[INFO] Drools :: Clips ..... SUCCESS [1.464s]
[INFO] Drools :: Planner parent ..... SUCCESS [0.527s]
[INFO] Drools :: Planner core ..... SUCCESS [2.209s]
[INFO] Drools :: Planner examples ..... SUCCESS [4.689s]
[INFO] ------------------------------------------------------------------------
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESSFUL
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 2 minutes 24 seconds
[INFO] Finished at: Tue Apr 07 15:11:14 EEST 2009
[INFO] Final Memory: 48M/178M
[INFO] ------------------------------------------------------------------------>
```

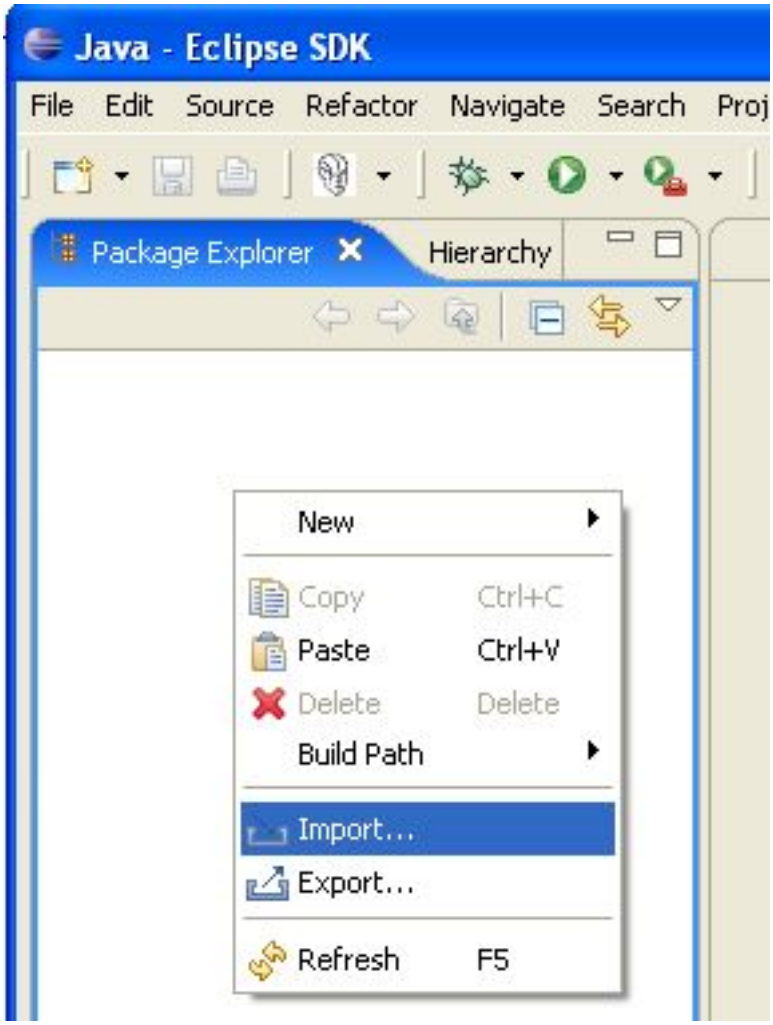
The generated manual can be found in the `target/drools-docs-$VERSION.jar` file, a compressed archive with all formats.

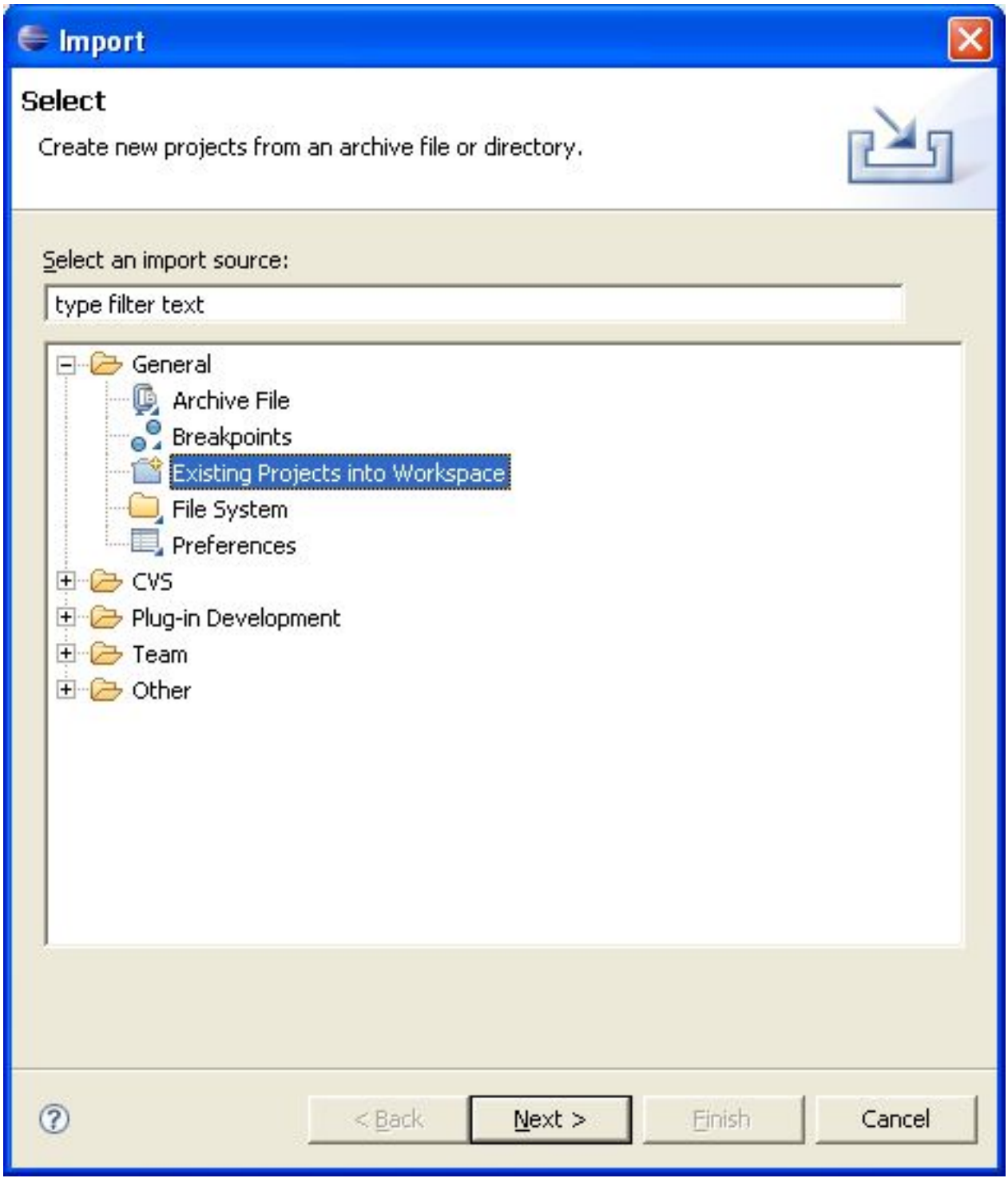
3.5. Eclipse

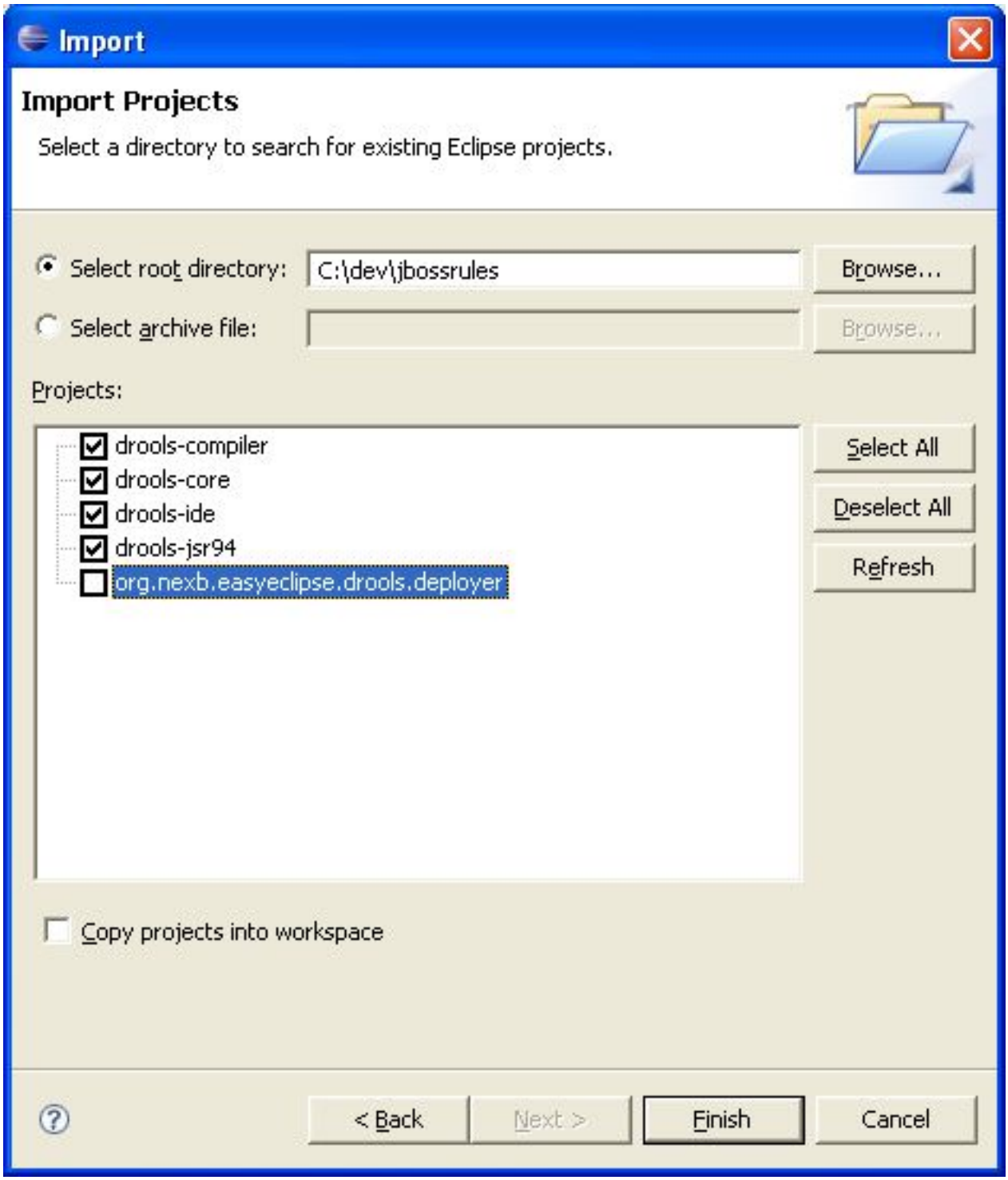
3.5.1. Importing Eclipse Projects

With the Eclipse project files generated they can now be imported into Eclipse. When starting Eclipse open the workspace in the root of your subversion checkout.

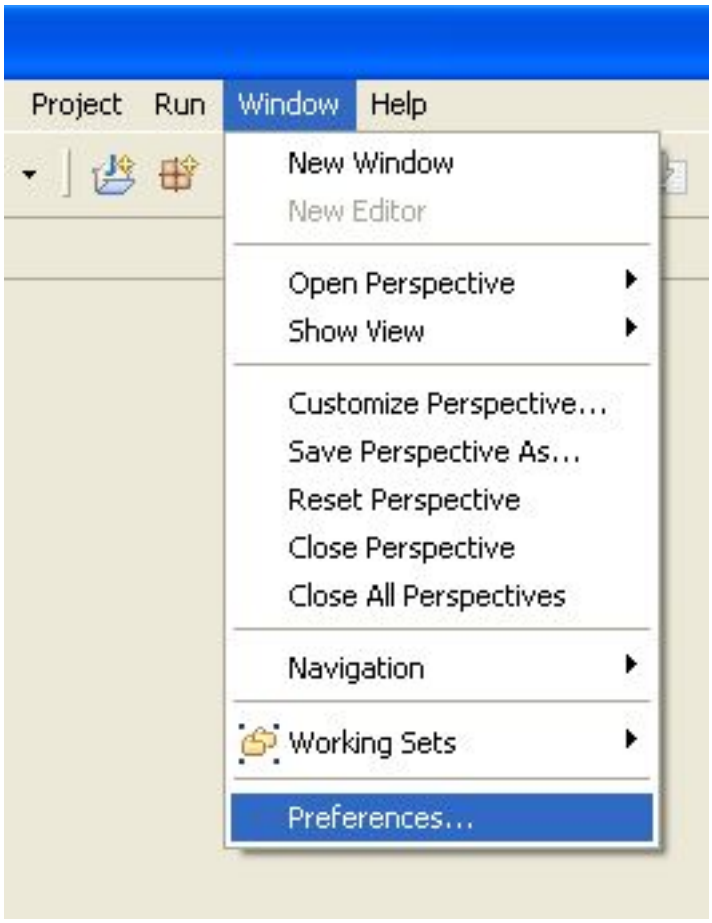

Workspace Launcher
Select a workspace
Eclipse SDK stores your projects in a folder called a workspace.
Choose a workspace folder to use for this session.
Workspace:
c:\dev\jbossrules
Browse...
Use this as the default and do not ask again
OK
Cancel

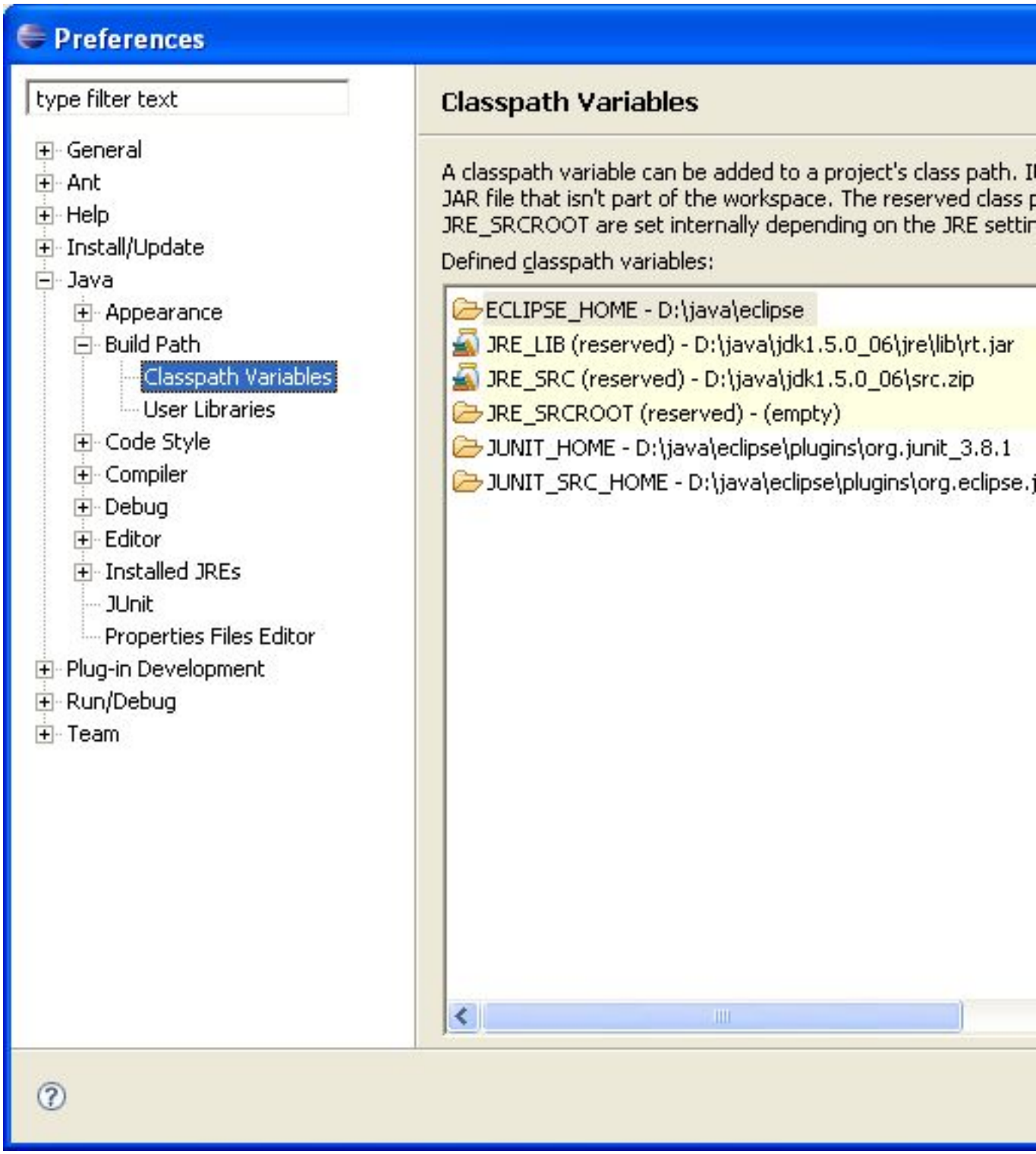

Java - Eclipse SDK
File
Edit
Source
Refactor
Navigate
Search
Proj
Package Explorer
Hierarchy
New
Copy
Ctrl+C
Paste
Ctrl+V
Delete
Delete
Build Path
Import...
Export...
Refresh
F5

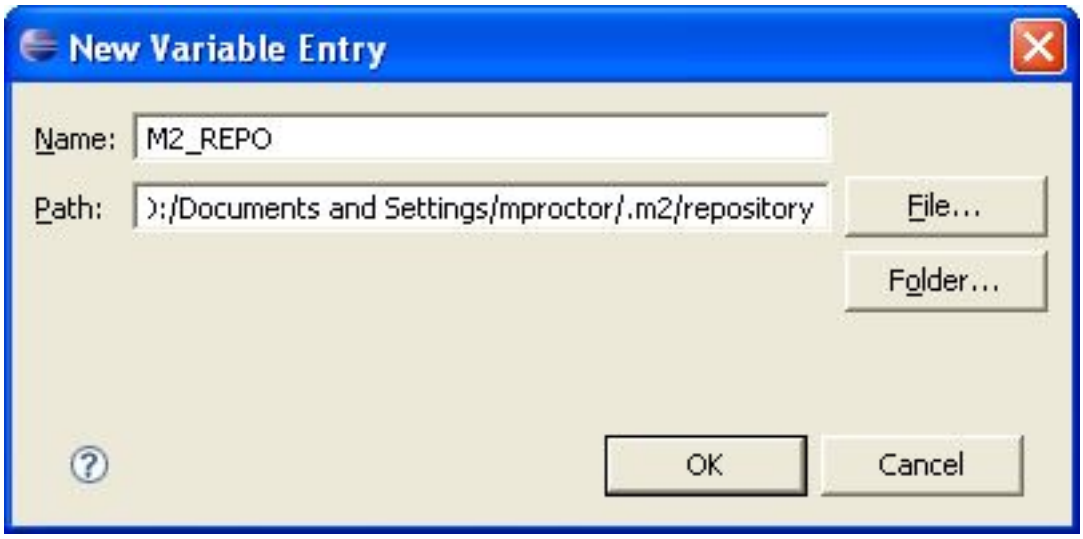

Import
Select
Create new projects from an archive file or directory.
Select an import source:
type filter text
General
Archive File
Breakpoints
Existing Projects into Workspace
File System
Preferences
CVS
Plug-in Development
Team
Other
< Back
Next >
Finish
Cancel

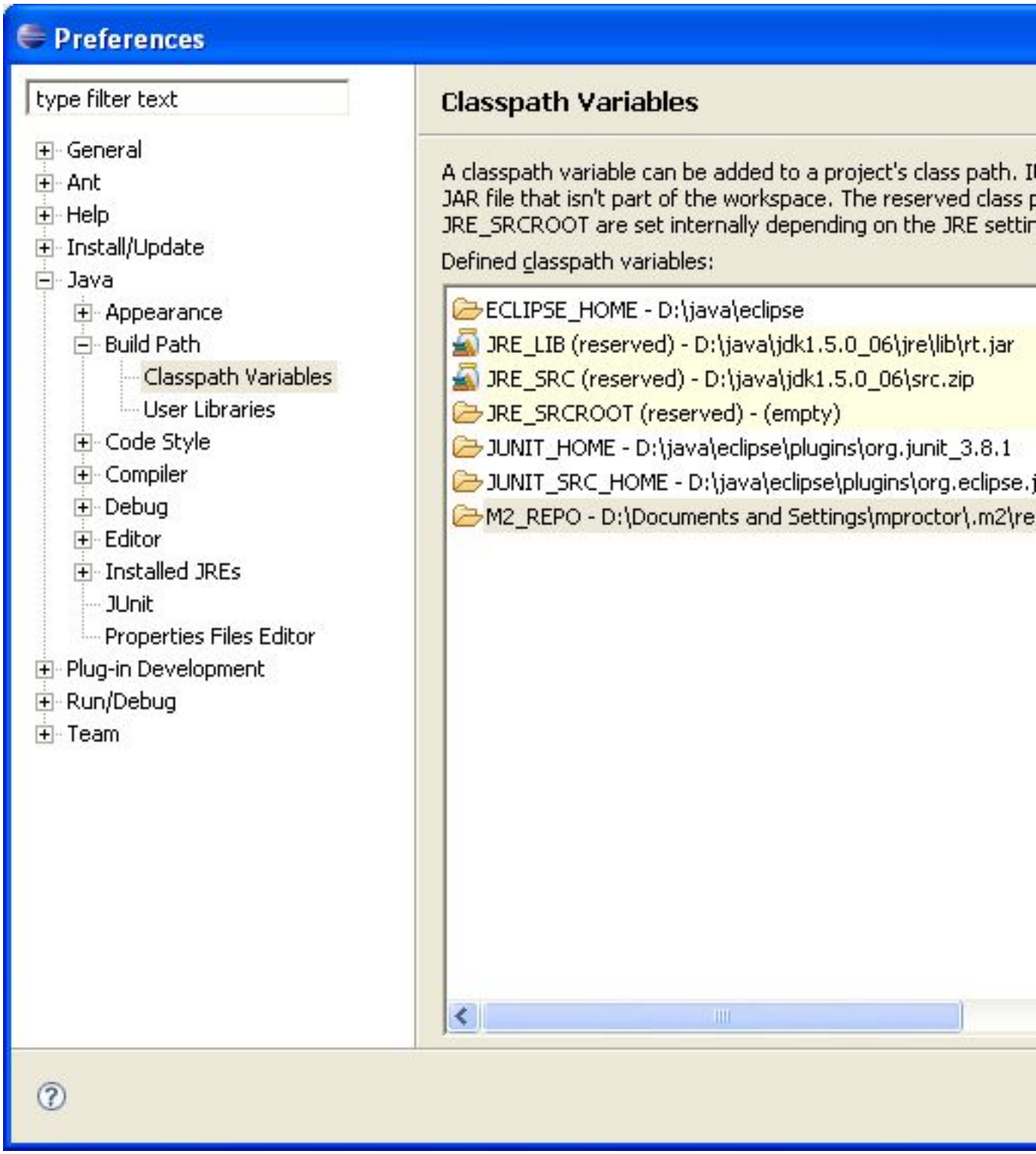


When calling `mvn install` all the project dependencies were downloaded and added to the local Maven repository. Eclipse cannot find those dependencies unless you tell it where that repository is. To do this setup an M2_REPO classpath variable.


Project
Run
Window
Help
New Window
New Editor
Open Perspective
Show View
Customize Perspective...
Save Perspective As...
Reset Perspective
Close Perspective
Close All Perspectives
Navigation
Working Sets
Preferences...


Preferences
type filter text
General
Ant
Help
Install/Update
Java
Appearance
Build Path
Classpath Variables
User Libraries
Code Style
Compiler
Debug
Editor
Installed JREs
JUnit
Properties Files Editor
Plug-in Development
Run/Debug
Team
Classpath Variables
A classpath variable can be added to a project's class path. I
JAR file that isn't part of the workspace. The reserved class p
JRE_SRCROOT are set internally depending on the JRE settin
Defined classpath variables:
ECLIPSE_HOME - D:\java\eclipse
JRE_LIB (reserved) - D:\java\jdk1.5.0_06\jre\lib\rt.jar
JRE_SRC (reserved) - D:\java\jdk1.5.0_06\src.zip
JRE_SRCROOT (reserved) - (empty)
JUNIT_HOME - D:\java\eclipse\plugins\org.junit_3.8.1
JUNIT_SRC_HOME - D:\java\eclipse\plugins\org.eclipse.


New Variable Entry
Name:
M2_REPO
Path:
):/Documents and Settings/mproctor/.m2/repository
File...
Folder...
OK
Cancel


Preferences
type filter text
General
Ant
Help
Install/Update
Java
Appearance
Build Path
Classpath Variables
User Libraries
Code Style
Compiler
Debug
Editor
Installed JREs
JUnit
Properties Files Editor
Plug-in Development
Run/Debug
Team
Classpath Variables
A classpath variable can be added to a project's class path. I
JAR file that isn't part of the workspace. The reserved class p
JRE_SRCROOT are set internally depending on the JRE settin
Defined classpath variables:
ECLIPSE_HOME - D:\java\eclipse
JRE_LIB (reserved) - D:\java\jdk1.5.0_06\jre\lib\rt.jar
JRE_SRC (reserved) - D:\java\jdk1.5.0_06\src.zip
JRE_SRCROOT (reserved) - (empty)
JUNIT_HOME - D:\java\eclipse\plugins\org.junit_3.8.1
JUNIT_SRC_HOME - D:\java\eclipse\plugins\org.eclipse.j
M2_REPO - D:\Documents and Settings\mproctor\.m2\re

Index