Red Hat
Six Products Demo (RH6PD)
BRMS DEMO GUIDE
BRMS & JBPM

The Intelligent Integrated Enterprise
A JBoss Standard Operating Environment

Version History

Version	Author	Date	Comments
0.1	Derek Howlett	21st January 2013	First Draft
0.2	Derek Howlett	25th January 2013	Updated screen-shots of new UI
0.3	Derek Howlett	31st January 2013	Updated installation instructions

Table of Contents

Introduction

Welcome to the JBoss Middleware six products demonstration suite. If your new to JBoss middleware this suite of simple demos will allow you to get up and running with six of the most popular JBoss products very quickly.

This document describes the first of the Six Products Demos from JBoss Middleware. In this guide we will describe a business process for a simple car insurance quote. Essentially we will demonstrate how easy it is to build an I.T platform for selling car insurance. Although a simple representation of the actual insurance industry process it will cover all aspects of Enterprise architecture required to get to a fully functioning quote engine in place.

The car insurance quote is implemented with JBoss Business Rules Management System (BRMS). This version of BRMS also includes a powerful embeddable Business Process Management system which is used to control the business process .

Demo Installation - Setup and Configuration

Install Prerequisites

Component	Version Required.
JBoss Application Server	EAP 6.0.1
JBoss BRMS	BRMS 5.3.1
RH6PD Insurance Demo	RH6PD BRMS V0.1

Installation files

The RH6PD BRMS demo is delivered as a single ZIP file which must be expanded to reveal the following file :-

Component	Version Required
RH6PD JavaScript Web Application	webJsInterface.war
RH6PD BRMS DEMO EXPORT	repository.zip
RH6PD Data Model	Model.jar
RH6PD install directory	Install

Download the RH6PD-CARINSURANCE-DEMO ZIP file.

First you need to extract the demo distribution zip (rh6pd-carinsurance-demo-0.1.zip) to a location of your choosing . Once extracted, you will have the following folder structure:

- rh6pd-carinsurance-demo.zip

    ○ installs – Initially empty, but will be where you place the EAP & BRMS platform downloads from access.redhat.com.

    ○ projects – The project that comprise the demo.

    ○ support – Additional supporting files used by the demo.

    ○ target – Will be created by running rh6pd-install.sh. This directory will contain the fully configured BRMS runtime server.

    ○ rh6pd-install.sh – Script to install and configure the run time server environment.

    ○ start – start script to start BRMS once installed.

    ○ RH6PD BRMS Demo Guide.odt – This document.

Why not set an environment variable to point to your install directory. If your using the bash shell the configuration should be in your $HOME/.bashrc

```
$export RH6PD-INSTALL="$HOME/rh6pd-carinsurance-demo"
```

from now on the convention we will use is to refer to this install directory as:-

```
$RH6PD-INSTALL
```

Download JBoss Business Rules Management Platform (BRMS 5.3.1)

Download JBoss Enterprise BRMS Platform (JBoss BRMS 5.3.1 Deployable) from the Red Hat Customer Portal (https://access.redhat.com/jbossnetwork).

Download BRMS Platform:
1. Under *JBoss Enterprise Platforms*, select the *BRMS Platform* product.
2. Select version *5.3.1* in the *Version* field.
3. Download *JBoss BRMS 5.3.1 Deployable for EAP 6*
   *(*Please note that this is the deployable distribution, not the standalone one.)

Now copy `brms-p-5.3.1.GA-deployable-ee6.zip`, to the rh6pd-carinsurance-demo *installs* folder $RH6PD-INSTALL/installs.

```
$RH6PD-INSTALL/installs/brms-p-5.3.1.GA-deployable-ee6.zip
```

Download JBoss Application Server 6.0.1

Download EAP 6 Platform:
4. Under *JBoss Enterprise Platforms*, select the Application *Platform* product.
5. Select version 6.0.*1* in the *Version* field.
6. Download *JBoss Aplication Platform 6.0.1*

Now copy `jboss-eap-6.0.1.zip`, to the rh6pd-carinsurance-demo's *installs* folder. $RH6PD-INSTALL,

```
$RH6PD-INSTALL/installs/jboss-eap-6.0.1.GA.zip
```

Run the   rh6pd-carinsurance-demo Install Script

Now from the $RH6PD-INSTALL/rh6pd-carinsurance-demo folder run the *install* script:

```
$ ./install.sh
```

When the script completes you will have a new folder named *jboss-eap-6.0*, in the rh6pd-carinsurance-demo's $RH6PD-INSTALL/*target* folder.

The folder is a ready to run EAP 6 server with BRMS 5.3.1 with the following modifications have been made:

- The *admin* account is enabled (password is admin) in the *brms-roles.properties* and *brms-users.properties* file in rh6pd-carinsurance-demo/*target/jboss-eap-6.0/jboss-as/standalone/configuration*

- The security settings are modified by copying *components.xml* to :-
  $RH6PD-INSTALL/*target/jboss-eap-6.0/standalone/deployments*

- Adjusted server security domain to include *brms,* copied new *standalone.xml* to $RH6PD-INSTALL/*target/jboss-eap-6.0/jboss-as/standalone/configuration*

- The domain model jar Model.*jar* added to the console at $RH6PD-INSTALL/*target/jboss-eap-6.0/deployments/jbpm-gwt-console-server.war/WEB-INF/lib*

- Human Task users are enabled by copying a new *jbpm-human-task-war-web.xml* file to $RH6PD-INSTALL/*target/jboss-eap-6.0/jboss-as/standalone/deployments/jbpm-human-task.war/WEB-INF/web.xml*

- Both business-central-server and jbpm-human-task server needs a netty dependency, added a *MANIFEST.MF* to their repective *WEB-INF/classes/META-INF* directories

- The various testing dependencies will be unzipped and copied into rh6pd-carinsurance-demo/*support/lib/**

- Deploy RH6PD JavaScript demo web application – rh6pd-webJsInterface.war into the deployment directory. *The webJsInterface.war file has been unzipped and copied into the* rh6pd-carinsurance-demo/server/default/deploy directory.

Importing the RH6PD car insurance rules repository

In this final step you will need to start BRMS and import the car insurance repository.

Starting BRMS

Open a terminal window and execute the following command.

$RH6PD-INSTALL/target/ *jboss-eap-6.0/bin/standalone.sh*
or use the provided start script:-

       ./start

This will start the application server and deploy and start BRMS and the rh6pd car insurance demo web application.

Login to BRMS

Next navigate your browser to http://localhost:8080/jboss-brms*, log in as admin/admin the BRMS guvnor web interface is displayed.*

Import RH6PD Demo Repository

On the left hand navigation panel expand Administration

Select Import/Export. The import export form is displayed. Browse to the $RH6PD-INSTALL/*repository.zip file which is included in the demo distribution.*

Click Import.

Build the RH6PD Car insurance Package

Now using the side navigation panel , browse to the Knowlegde Base. Expand the packages org.jbpm.evaluation.carinsurnace to reveal the list of assets.
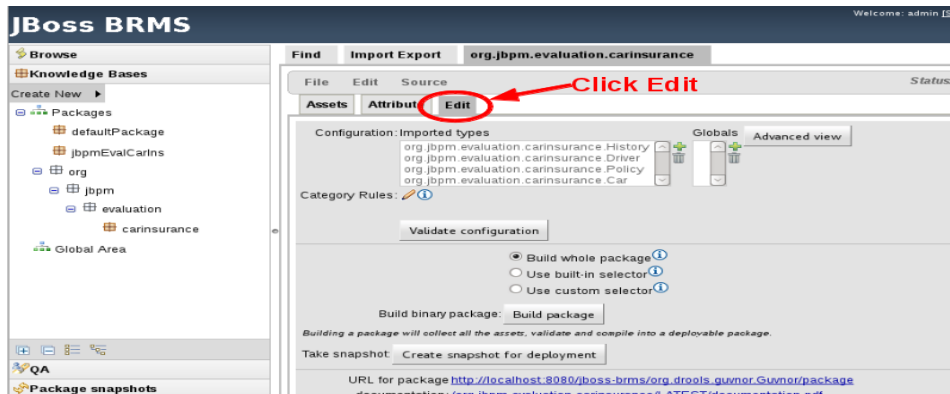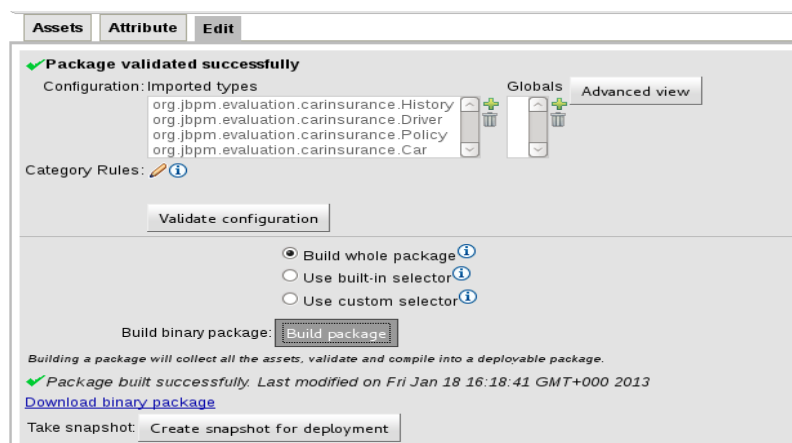

Once your in the edit area of the package you can now validate and build the package.

Click Validate configuration, wait for the small green tick and Package validated successfully to be displayed at the top.
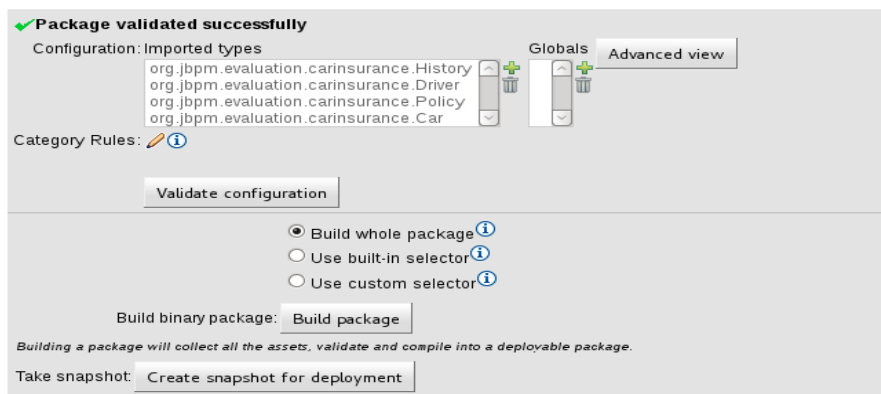
Now click build package to complete the build process.

An Overview of the Demo Components

The Data Model

A distinct and carefully designed data model can make your business and subsequent I.T systems very much easier to manage. Data modelling is often refereed to as the year two problem when systems are integrated together; there is a sudden realisation that the data on every system is formatted differently.

A well-designed and managed data model, which is implemented up front can save time, money and therefore reduce the costs of running your business.

For our six products demonstration we will design a very simple data model, make links between the component parts and deploy into a central business repository.

- **Note The RH6PD car insurance demo includes a data model, essentially a set of Java class files that represent the objects as facts. The Model must be present in the BRMS repository and the class path of the application server.**

The Customer (Driver)

This is the entity which represents our customer, for the car insurance quote our customer is defined by a simple object called "Driver".

Driver looks like the following: -

Driver

Description	Type
Name	String
Address	Object
Age	Integer
Social Security Number	String
Driving Licence Number	String
Credit Score	Integer

The Vehicle (Car)

This is the entity which represents the vehicle being insured, for our car insurance quote the vehicle is defined by a simple object called "Car".

Car looks like the following:-

Description	Type
Make	String
Model	String
Year	Integer

Registration Number	String

The Policy (The Result of the Insurance Application Process)

This is the entity which represents the insurance quote itself, this is defined by a simple object called "Policy".

Policy looks like the following:-

Description	Type
Policy Number	Integer
Policy Date	Date
Named Driver	String
Address	Object Address
Car Registration Number	String
Total Risk	Integer
Driver Risk	Integer
Car Risk	Integer
History Risk	Integer
Price	Currency
Discount	

The Business Process

This section describes in some detail the business process and the steps involved in generating an insurance quote. If your just looking for instructions on running the demo, please jump to the next chapter "Demo Script".

There are three sets of rules which are used to calculate the policy price, the risk of the drive, the risk of the car and finally the pricing rule used to match the driver and car to determine the final price.
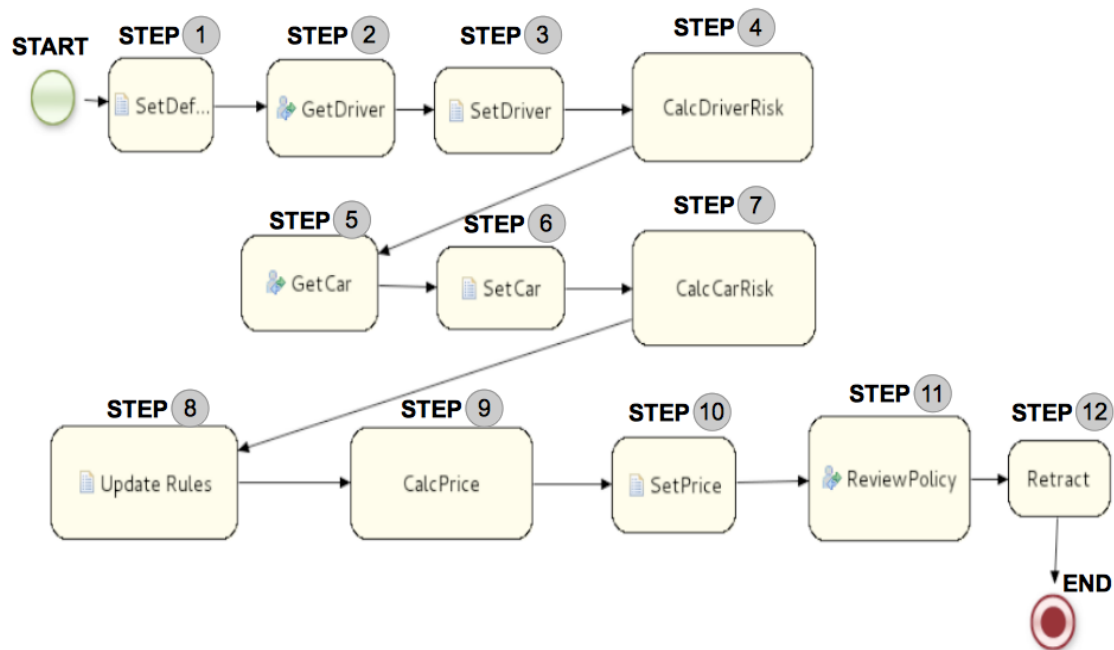
There are three human tasks used in the business process. Asking for information about the driver, asking for information on the car and finally the review police human task which we use to review all the details entered and display the final policy price.

Starting the business process

You can start this car insurance business process from the JBPM-CONSOLE simply by selecting the Quote222 and clicking start. You will be presented with a standard start form generated from the BPMN2.0 code. You will then need to complete the tasks from the tasks menu of the JBPM-CONSOLE.

Preferred....

Or by using the JavaScript application shipped with this demo. Go to http://localhost:8080/rh6pd and follow the instructions listed in the Demo Script section of this demo.

Step 1 - Script - Setting default environment variables

```
System.out.println("*** Starting Set Driver ***");

// Set variables
driverRisk="Set Driver: Not Set Yet";
driver = new Driver();
policy = new Policy();

// Set policy object with default values
policy.setDriverRisk(999);
policy.setPrice(999);

// Set driver object up with variable required for firing rules.
driver.setCreditScore(999);
driver.setName(name);
driver.setAge(Integer.parseInt(age));

//Insert into session.
kcontext.setVariable("driver",driver);
kcontext.setVariable("policy",policy);
kcontext.setVariable("name",name);
kcontext.setVariable("address1",address1);
kcontext.setVariable("address2",address2);
kcontext.setVariable("age",age);
kcontext.setVariable("driverRisk",driverRisk);


// Insert into working memory
insert(driver);
insert(policy);

System.out.println("*** Exiting  Set Driver ***");</script>
    </scriptTask>
    <scriptTask id="_17" name="SetDefaultVariables" >
      <script>

// Set Variables
name ="SetCar: Not Set Yet";
address1="SetCar: Not Set Yet";
address2="SetCar: Not Set Yet";
age="SetCar: Not Set Yet";

make    ="SetCar: Not Set Yet";
model  ="SetCar: Not Set Yet";
reg        ="SetCar: Not Set Yet";
year      =1999;

driverRisk="SetCar: Not Set Yet";
carRisk="SetCar: Not Set Yet";

// Driver
kcontext.setVariable("name",name);
kcontext.setVariable("address1",address1);
kcontext.setVariable("address2",address2);
kcontext.setVariable("age",age);

//Car
kcontext.setVariable("make",make);
kcontext.setVariable("model",model);
kcontext.setVariable("reg",reg);
kcontext.setVariable("year",year);

// Risks
kcontext.setVariable("carRisk",carRisk);
kcontext.setVariable("driverRisk",driverRisk)
```

Step 2 – Human Task - Get Driver Information

This is the step in the process where we collecting information about the driver. This task is a human task and is delegated to the human task server which generates a form for the user to complete.

```
<userTask id="_4" name="GetDriver" >
     <ioSpecification>
       <dataInput  id="_4_CommentInput"   name="Comment" />
       <dataInput  id="_4_SkippableInput" name="Skippable" />
       <dataInput  id="_4_TaskNameInput"  name="TaskName" />
       <dataInput  id="_4_GroupIdInput"   name="GroupId" />
       <dataInput  id="_4_PriorityInput"  name="Priority" />
       <dataOutput id="_4_NameOutput"     name="Name" />
       <dataOutput id="_4_AgeOutput"      name="Age" />
       <dataOutput id="_4_Address2Output" name="Address2" />
       <dataOutput id="_4_Address1Output" name="Address1" />
       <inputSet>
         <dataInputRefs>_4_CommentInput</dataInputRefs>
         <dataInputRefs>_4_SkippableInput</dataInputRefs>
         <dataInputRefs>_4_TaskNameInput</dataInputRefs>
         <dataInputRefs>_4_GroupIdInput</dataInputRefs>
         <dataInputRefs>_4_PriorityInput</dataInputRefs>
       </inputSet>
       <outputSet>
         <dataOutputRefs>_4_NameOutput</dataOutputRefs>
         <dataOutputRefs>_4_AgeOutput</dataOutputRefs>
         <dataOutputRefs>_4_Address2Output</dataOutputRefs>
         <dataOutputRefs>_4_Address1Output</dataOutputRefs>
       </outputSet>
     </ioSpecification>
    …...
   </userTask>
```

Step 3 – Script - Set environment variables for entered driver information

This step in the process is used to create the driver and policy objects, set driver variables to default values and the insert the driver and policy setup a number of environment variables to default values.

```
// Set variables
driverRisk="Set Driver: Not Set Yet";
driver = new Driver();
policy = new Policy();

// Set policy object with default values
policy.setDriverRisk(999);
policy.setPrice(999);

// Set driver object up with variable required for firing rules.
driver.setCreditScore(999);
driver.setName(name);
driver.setAge(Integer.parseInt(age));

//Insert into session.
kcontext.setVariable("driver",driver);
kcontext.setVariable("policy",policy);
kcontext.setVariable("name",name);
kcontext.setVariable("address1",address1);
kcontext.setVariable("address2",address2);
kcontext.setVariable("age",age);
kcontext.setVariable("driverRisk",driverRisk);


// Insert into working memory
insert(driver);
insert(policy);
```

Step 4 – Rule Execution - Run the rules for the driver risk calculation.

This step of the process simply calls the rules in the calcDriverRisk decision table. This is done with the following ruleflowgroup statement.

<businessRuleTask id="_6" name="CalcDriverRisk" g:ruleFlowGroup="calcDriverRisk" >

From the BRMS user interface, expand the rules tab to see the calc Driver risk table.

#	Description	ruleflow-group	from	to	risk
1		calcDriverRisk	17	21	10
2		calcDriverRisk	22	25	9
3		calcDriverRisk	26	35	8
4		calcDriverRisk	36	45	7
5		calcDriverRisk	46	55	6
6		calcDriverRisk	56	65	5
7		calcDriverRisk	66	75	4
8		calcDriverRisk	76	85	3
9		calcDriverRisk	86	95	2
10		calcDriverRisk	96	100	1

Once you have the table displayed you can see the actual rules generated by the decision table by selecting source tab then clicking View Source.

```
Viewing source for: calcDriverRisk                                    ✖

            Viewing source for: calcDriverRisk

1.   |//from row number: 1
2.   |rule "Row 1 calcDriverRisk"
3.   |    ruleflow-group "calcDriverRisk"
4.   |    dialect "mvel"
5.   |    when
6.   |        policy : Policy( this != null )
7.   |        driver : Driver( age >= 17 , age <= 21 )
8.   |    then
9.   |        policy.setDriverRisk( 10 );
10.  |end
11.  |
12.  |//from row number: 2
13.  |rule "Row 2 calcDriverRisk"
14.  |    ruleflow-group "calcDriverRisk"
15.  |    dialect "mvel"
16.  |    when
17.  |        policy : Policy( this != null )
18.  |        driver : Driver( age >= 22 , age <= 25 )
19.  |    then
```

Step 5 – Human Task - Get Car Information

This is the step in the process where we collecting information about the car. This task is a human task and is delegated to the human task server which generates a form for the user to complete.

```
<userTask id="_13" name="GetCar" >
    <ioSpecification>
      <dataInput id="_13_TaskNameInput" name="TaskName" />
      <dataInput id="_13_GroupIdInput" name="GroupId" />
      <dataOutput id="_13_YearOutput" name="Year" />
      <dataOutput id="_13_ModelOutput" name="Model" />
      <dataOutput id="_13_RegOutput" name="Reg" />
      <dataOutput id="_13_MakeOutput" name="Make" />
      <inputSet>
        <dataInputRefs>_13_TaskNameInput</dataInputRefs>
        <dataInputRefs>_13_GroupIdInput</dataInputRefs>
      </inputSet>
      <outputSet>
        <dataOutputRefs>_13_YearOutput</dataOutputRefs>
        <dataOutputRefs>_13_ModelOutput</dataOutputRefs>
        <dataOutputRefs>_13_RegOutput</dataOutputRefs>
        <dataOutputRefs>_13_MakeOutput</dataOutputRefs>
      </outputSet>
    </ioSpecification>....
</userTask>
```

Step 6 – Human Task - Set environment variables for entered car information

Once the human task for get Car has completed the resulting variables must be mapped back into the environment. The car object is created, variable are coppied in from the form and the policy object is updated.

```
System.out.println("create new car");
car = new Car();

// Set policy object with default values
System.out.println("set car risk to 999");
policy.setCarRisk(999);


// Set car object up with variable required for firing rules.
car.setMake(make);
car.setModel(model);
car.setReg(reg);

//Insert into session.
kcontext.setVariable("car",car);
kcontext.setVariable("make",make);
kcontext.setVariable("model",model);
kcontext.setVariable("reg",reg);
kcontext.setVariable("year",year);
kcontext.setVariable("carRisk",carRisk);
kcontext.setVariable("driverRisk",driverRisk);

// Insert into working memory
insert(car);
kcontext.getKnowledgeRuntime().update(kcontext.getKnowledgeRuntime().getFactHandle(policy), policy);
```

Step 7 – Rule Execution - Run the rules for the car risk

This step of the process simply calls the rules in the calcCarRisk decision table. This is done with the following ruleflowgroup statement.

```
<businessRuleTask id="_14" name="CalcCarRisk" g:ruleFlowGroup="calcCarRisk" >
    </businessRuleTask>
```

From the BRMS user interface, expand the rules tab to see the calc Car risk table.

	#	Description	ruleflow-group	make	risk
				▲	
➕ ⊟	1		calcCarRisk	FORD	1
➕ ⊟	2		calcCarRisk	BMW	10
➕ ⊟	3		calcCarRisk	HONDA	5

Once you have the table displayed you can see the actual rules generated by the decision table by selecting source tab then clicking View Source.

Viewing source for: calcCarRisk

Viewing source for: calcCarRisk

```
1.  |//from row number: 1
2.  |rule "Row 1 calcCarRisk"
3.  |    ruleflow-group "calcCarRisk"
4.  |    dialect "mvel"
5.  |    when
6.  |        policy : Policy( this != null )
7.  |        car : Car( make matches "FORD" )
8.  |    then
9.  |        policy.setCarRisk( 1 );
10. |end
11. |
12. |//from row number: 2
13. |rule "Row 2 calcCarRisk"
14. |    ruleflow-group "calcCarRisk"
15. |    dialect "mvel"
16. |    when
17. |        policy : Policy( this != null )
18. |        car : Car( make matches "BMW" )
19. |    then
```

Step 8 – Script – Update Rules

Setup the policy and update the rules engine to take into account

A feature of the BRMS rules engine is when the process is loaded for the first time all the rules defined in the car insurance package fire with the intent to calculate the price. Of course at the beginning of the process there is no data loaded into the facts. As we progress through the process we create a driver object which forces the rules to fire once again, this sets the driver risk. We then create the car object which again forces the rules to fire, this time setting the car risk.

At step 8 we have the driver and car risks set in the policy, but the rules engine doesn't have any way that the calcPrice rule now needs to be fired. So we execute the following statements at step 8 to retrieve the policy risk components and then force the policy rules to fire.

```
// Get variables
driverRiskInt  = policy.getDriverRisk();
driverRisk     = new Integer (driverRiskInt).toString();

kcontext.getVariable("policy"));

carRiskInt = policy.getCarRisk();
carRisk    = new Integer (carRiskInt).toString();


// Update working memory
kcontext.getKnowledgeRuntime().update(kcontext.getKnowledgeRuntime().getFactHand
le(policy), policy);
```

Step 9 – Rule Execution – Calculate Price

This step of the process simply calls the rules in the calcPrice decision table. This is done with the following ruleflowgroup statement.

```
<businessRuleTask id="_11" name="CalcPrice" g:ruleFlowGroup="calcPrice" >
    </businessRuleTask>
```

From the BRMS user interface, expand the rules tab to see the calc Price decision table.

#	Description	ruleflow-group	Driver Rsk	Car Risk	price
1		calcPrice	1	1	100
2		calcPrice	2	1	202
3		calcPrice	3	1	303
4		calcPrice	4	1	400
5		calcPrice	5	1	500
6		calcPrice	6	1	600
7		calcPrice	7	1	299
8		calcPrice	8	1	800
9		calcPrice	9	1	1400
10		calcPrice	10	1	2300
11		calcPrice	1	5	101
12		calcPrice	2	5	202
13		calcPrice	3	5	303
14		calcPrice	4	5	404
15		calcPrice	5	5	505

Once you have the table displayed you can see the actual rules generated by the decision table by selecting source tab then clicking View Source.

Viewing source for: calcPrice

```
57. |    then
58. |        policy.setPrice( 600 );
59. |end
60. |
61. |//from row number: 7
62. |rule "Row 7 calcPrice"
63. |    ruleflow-group "calcPrice"
64. |    dialect "mvel"
65. |    when
66. |        policy : Policy( driverRisk == 7 , carRisk == 1 )
67. |    then
68. |        policy.setPrice( 299 );
69. |end
70. |
71. |//from row number: 8
72. |rule "Row 8 calcPrice"
73. |    ruleflow-group "calcPrice"
74. |    dialect "mvel"
75. |    when
```

Step 10 – Script - Set Price

A script which sets up environment variables based on the result of calculating the price in the previous steps.

```
// set variables
price="Not Set Yet";
// get variables
price=new Integer(policy.getPrice()).toString();

driverRisk=new Integer(policy.getDriverRisk()).toString();
carRisk=new Integer(policy.getCarRisk()).toString();

// set variables.
kcontext.setVariable("name",name);
kcontext.setVariable("address1",address1);
kcontext.setVariable("address2",address2);
kcontext.setVariable("age",age);
kcontext.setVariable("driverRisk",driverRisk);
kcontext.setVariable("carRisk",carRisk);
kcontext.setVariable("price",price);
kcontext.setVariable("make",make);
kcontext.setVariable("model",model);
```

Step 11- Human Task – Review Policy

This step is used to generate a form which reviews the details of the policy. This includes driver and car information and the final calculated price.

Should should accept the quote to force the process to the final step 12. retract.

```
<userTask id="_B09AA1BF-19DA-4F57-B074-F36CB7E94178" name="ReviewPolicy" >
    <ioSpecification>
      <dataInput id="_B09AA1BF-19DA-4F57-B074-F36CB7E94178_NameInput" name="Name" />
      <dataInput id="_B09AA1BF-19DA-4F57-B074-F36CB7E94178_AgeInput" name="Age" />
      <dataInput id="_B09AA1BF-19DA-4F57-B074-F36CB7E94178_ModelInput" name="Model" />
      <dataInput id="_B09AA1BF-19DA-4F57-B074-F36CB7E94178_DriverRiskInput" name="DriverRisk" />
      <dataInput id="_B09AA1BF-19DA-4F57-B074-F36CB7E94178_Address2Input" name="Address2" />
      <dataInput id="_B09AA1BF-19DA-4F57-B074-F36CB7E94178_Address1Input" name="Address1" />
      <dataInput id="_B09AA1BF-19DA-4F57-B074-F36CB7E94178_PriceInput" name="Price" />
      <dataInput id="_B09AA1BF-19DA-4F57-B074-F36CB7E94178_CarRiskInput" name="CarRisk" />
      <dataInput id="_B09AA1BF-19DA-4F57-B074-F36CB7E94178_MakeInput" name="Make" />
      <dataInput id="_B09AA1BF-19DA-4F57-B074-F36CB7E94178_CommentInput" name="Comment" />
      <dataInput id="_B09AA1BF-19DA-4F57-B074-F36CB7E94178_SkippableInput" name="Skippable" />
      <dataInput id="_B09AA1BF-19DA-4F57-B074-F36CB7E94178_TaskNameInput" name="TaskName" />
      <dataInput id="_B09AA1BF-19DA-4F57-B074-F36CB7E94178_GroupIdInput" name="GroupId" />
      <dataInput id="_B09AA1BF-19DA-4F57-B074-F36CB7E94178_PriorityInput" name="Priority" />
      <inputSet>
…..
</userTask>
```

Step 12 – Rule Execution – Retract Objects.

The final step in the process is labelled as retract, here we call a group three of rules,

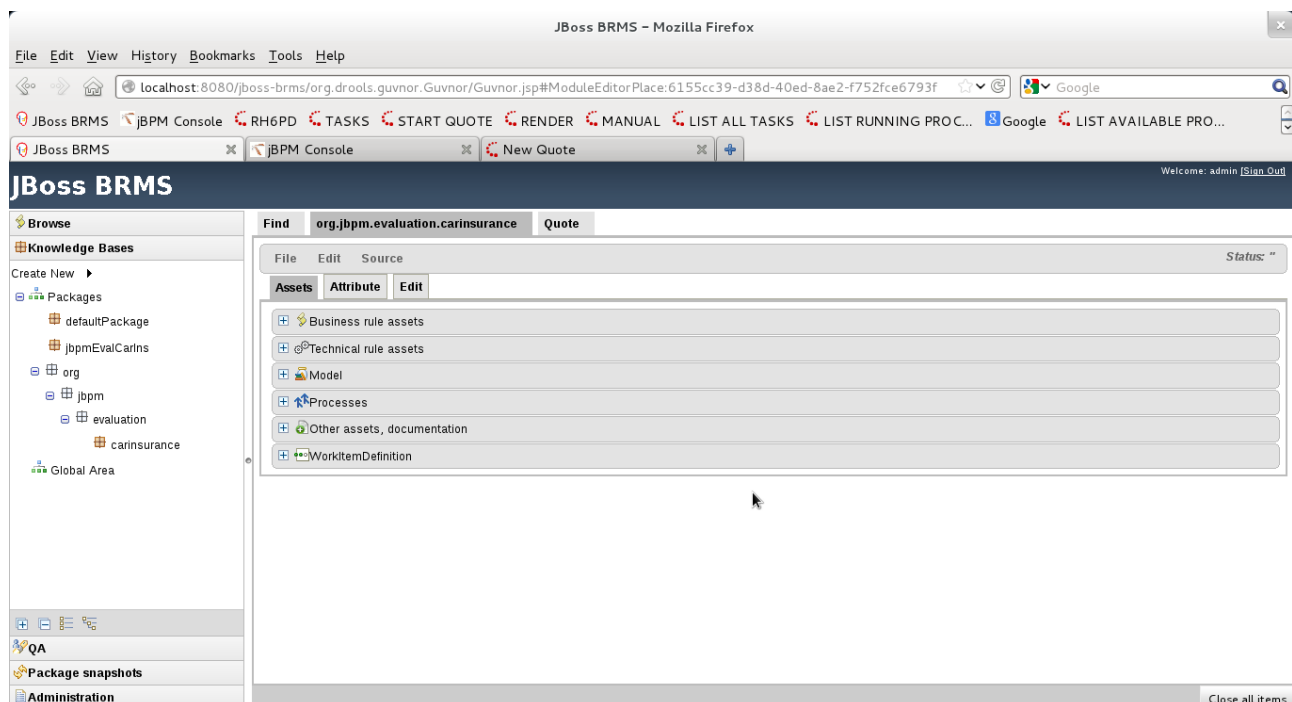| Rule	Description
RetractDriver	Removes the car object from the working memory of the rules engine.
RetractCar	Removes the car object from the working memory of the rules engine.
RetractPolicy	Removes the policy object from the working memory of the rules engine.

The Demo Script

Starting the demonstration

Web Browser Locations.

Connect to BRMS console	http://localhost:8080/jboss-brms	admin/admin
Connect to JBPM-console	http://localhost:8080/business-central/app.html	admin/admin
Open Home page for RH6PD	http://localhost:8080/rh6pd	
Start Quote Processs	http://localhost:8080/rh6pd/quote.html	
Retrive Tasks from Process	http://localhost:8080/rh6pd/tasks.html	

Navigate to the BRMS console http://localhost/jboss-brms

login as admin/admin

navigate to the car insurance package org.evaluation.jbpm.carinsurance

open the process show the process in the JBPM web designer
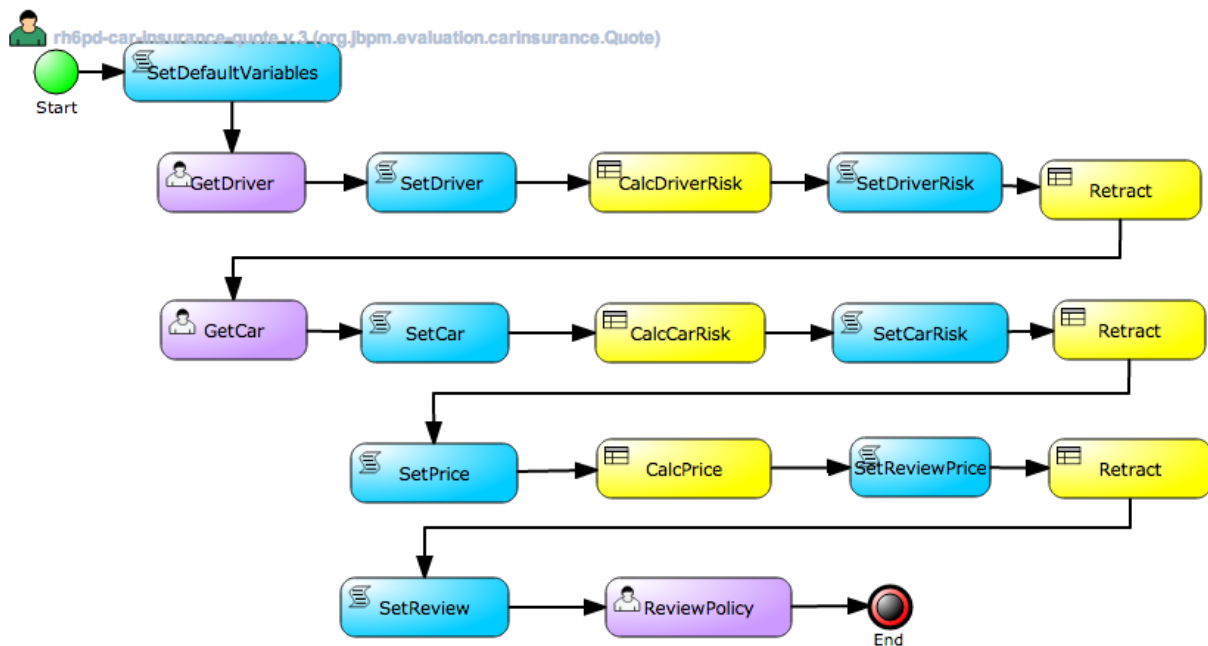
discuss the three human tasks

open rules tab

show calcDriverRisk to determine the risk of a 45 year old driver

	#	Description	ruleflow-group	from	to	risk
✚ ▢	1		calcDriverRisk	17	21	10
✚ ▢	2		calcDriverRisk	22	25	9
✚ ▢	3		calcDriverRisk	26	35	8
✚ ▢	4		calcDriverRisk	36	45	7
✚ ▢	5		calcDriverRisk	46	55	6
✚ ▢	6		calcDriverRisk	56	65	5
✚ ▢	7		calcDriverRisk	66	75	4
✚ ▢	8		calcDriverRisk	76	85	3
✚ ▢	9		calcDriverRisk	86	95	2
✚ ▢	10		calcDriverRisk	96	100	1

show calcCarRisk to determine the risk for a Ford Fiesta

	#	Description	ruleflow-group	make	risk
✚ ▢	1		calcCarRisk	FORD	1
✚ ▢	2		calcCarRisk	BMW	10
✚ ▢	3		calcCarRisk	HONDA	5

show calcPrice to determine the price for a for Driver risk of 7 and a car risk of 1

As you can see we are having a special deal on drivers with a risk of 7 and a car risk of 1, the price indicated shows 299. Watch out for this at the review policy step.

		#	Description	ruleflow-group	Driver Rsk	Car Risk	price
		1		calcPrice	1	1	100
		2		calcPrice	2	1	202
		3		calcPrice	3	1	303
		4		calcPrice	4	1	400
		5		calcPrice	5	1	500
		6		calcPrice	6	1	600
		7		calcPrice	7	1	299
		8		calcPrice	8	1	800
		9		calcPrice	9	1	1400
		10		calcPrice	10	1	2300
		11		calcPrice	1	5	101
		12		calcPrice	2	5	202
		13		calcPrice	3	5	303
		14		calcPrice	4	5	404
		15		calcPrice	5	5	505

Navigate to the JBPM-CONSOLE

Login as admin/admin – expand the process tab to reveal the demo process

Navigate to the RH6PD web application

Your are now at the RH6PD demo home page.

Click the START BRMS Demo button

You will be presented with the Quote process start screen.
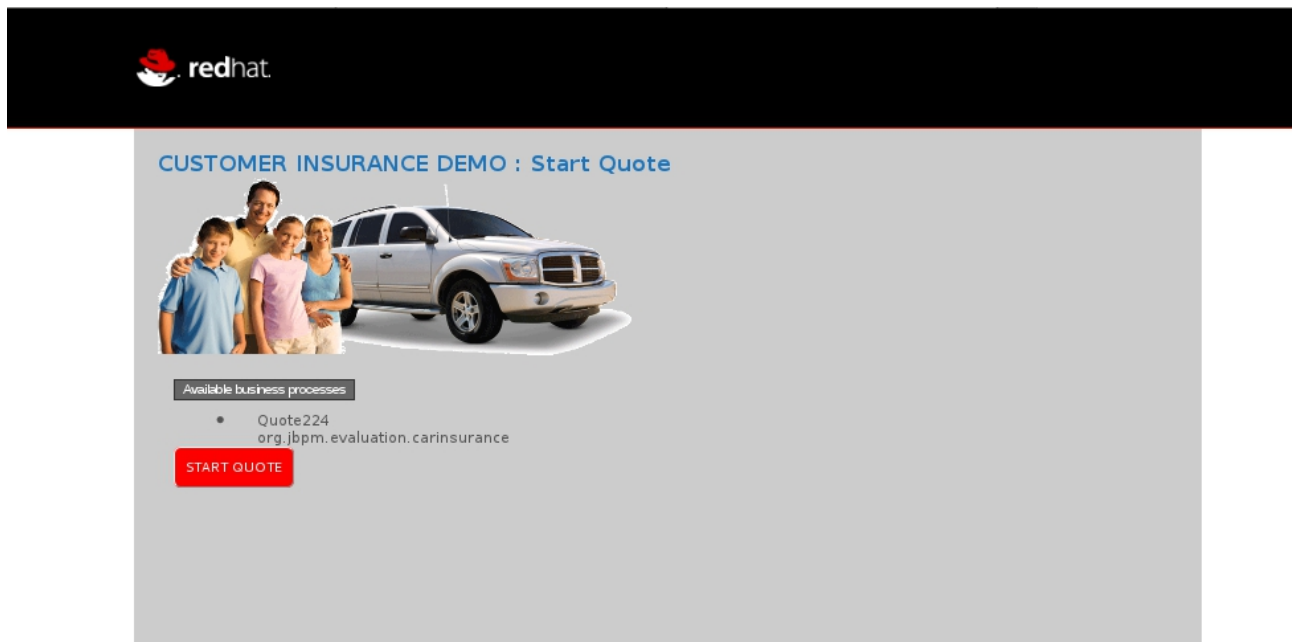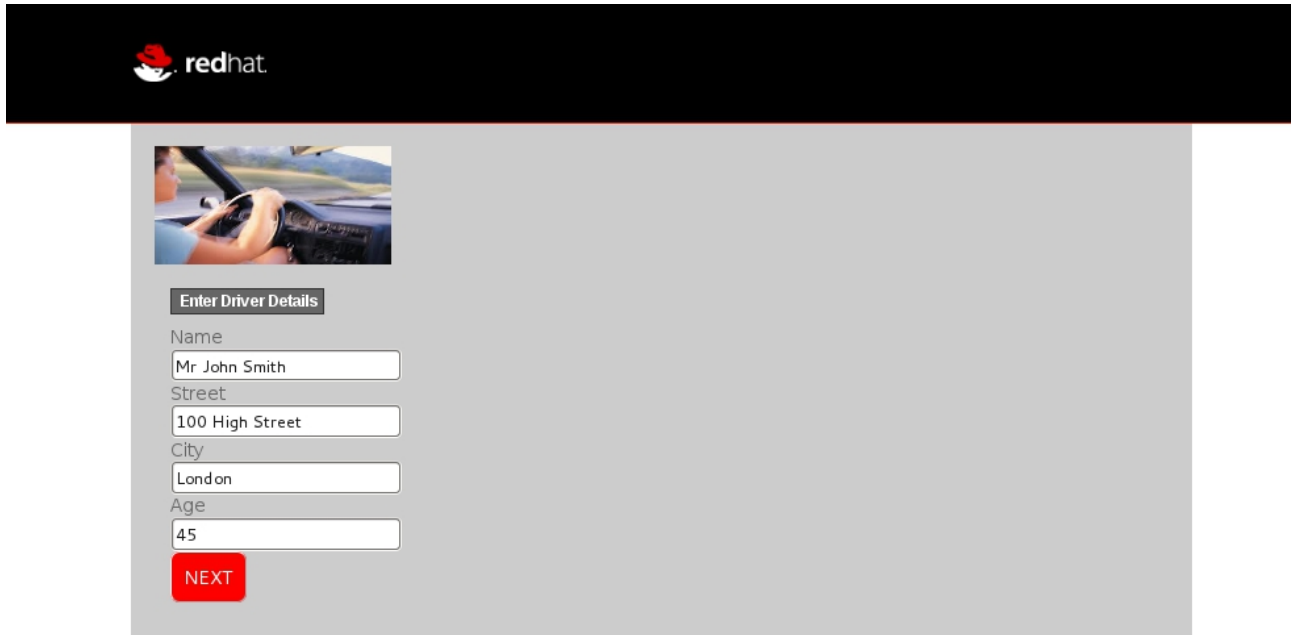
The Quote process start screen displays the current quote process available. If however you get the following message, this means you haven't authenticated with JBPM yet. Please go to the URL for the JBPM-CONSOLE and authenticate there.
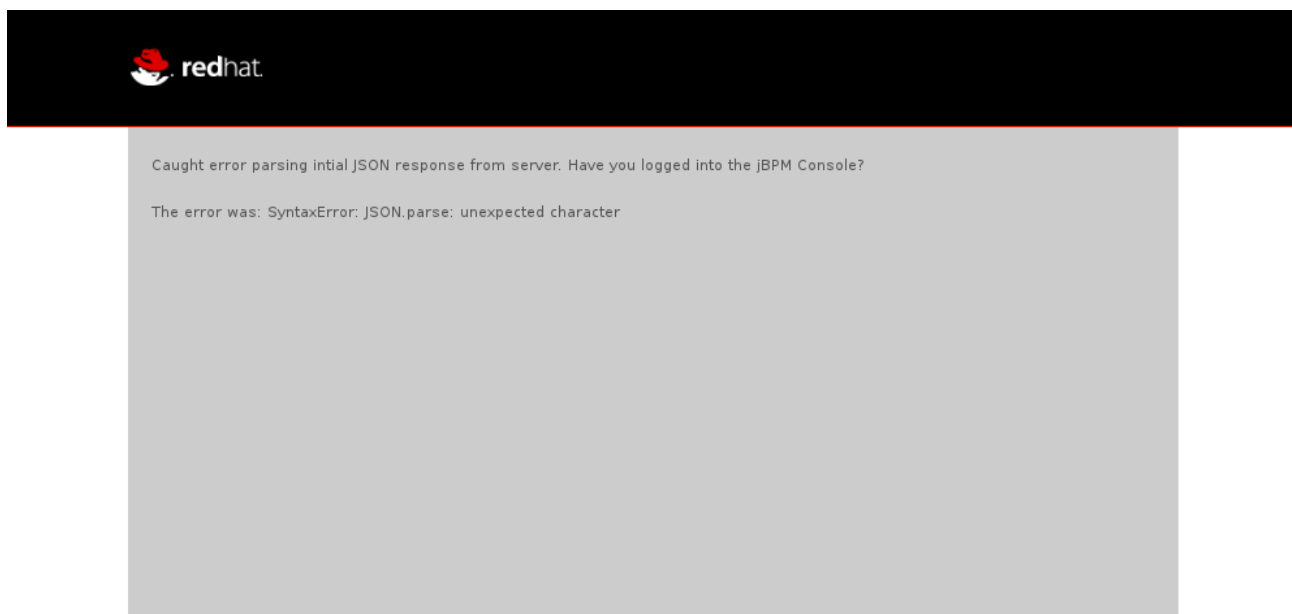
Get Driver Form

Clicking start on the Start Quote form will bring up the GetDriver form.

Once you have the Driver dialog on the screen, this is the point to switch tabs back to the JBPM-CONSOLE and this time selecting the running process and then the diagram box will bring up the process again but this time the image will have a red triangle indicating where you are in the process .
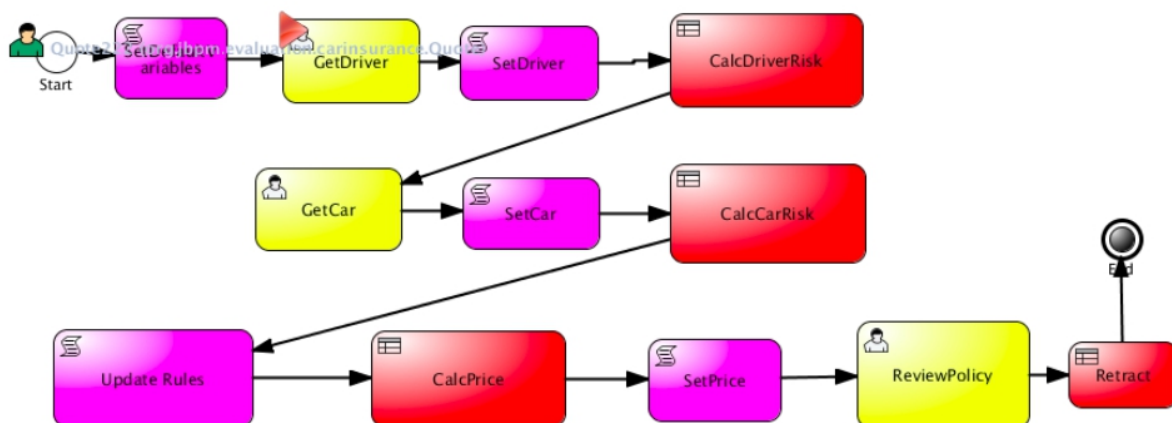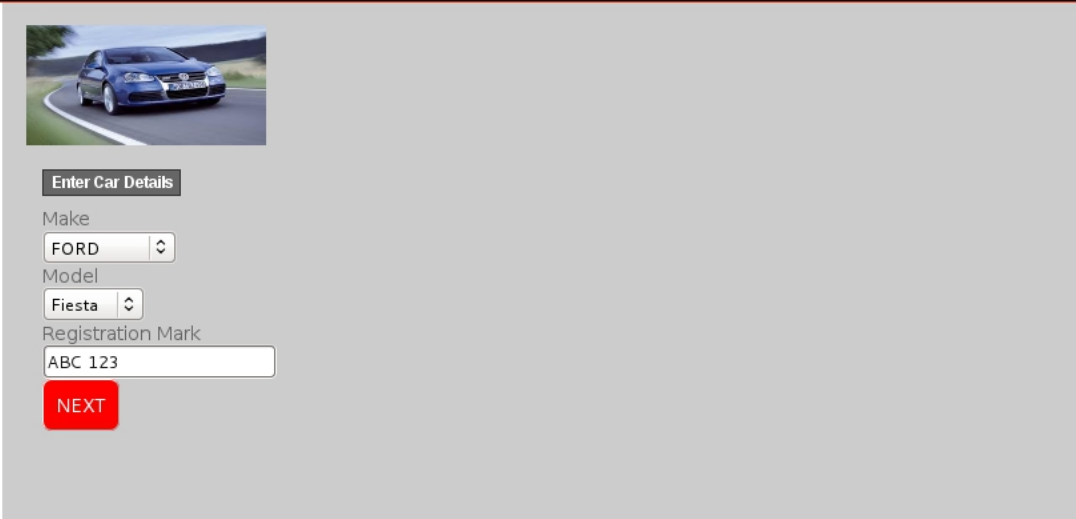
Get Car Form

Returning to the Driver form select default values have been inserted for ease of use. Currently a driver with an age with 45 is selected. Press NEXT to move to the Car form.
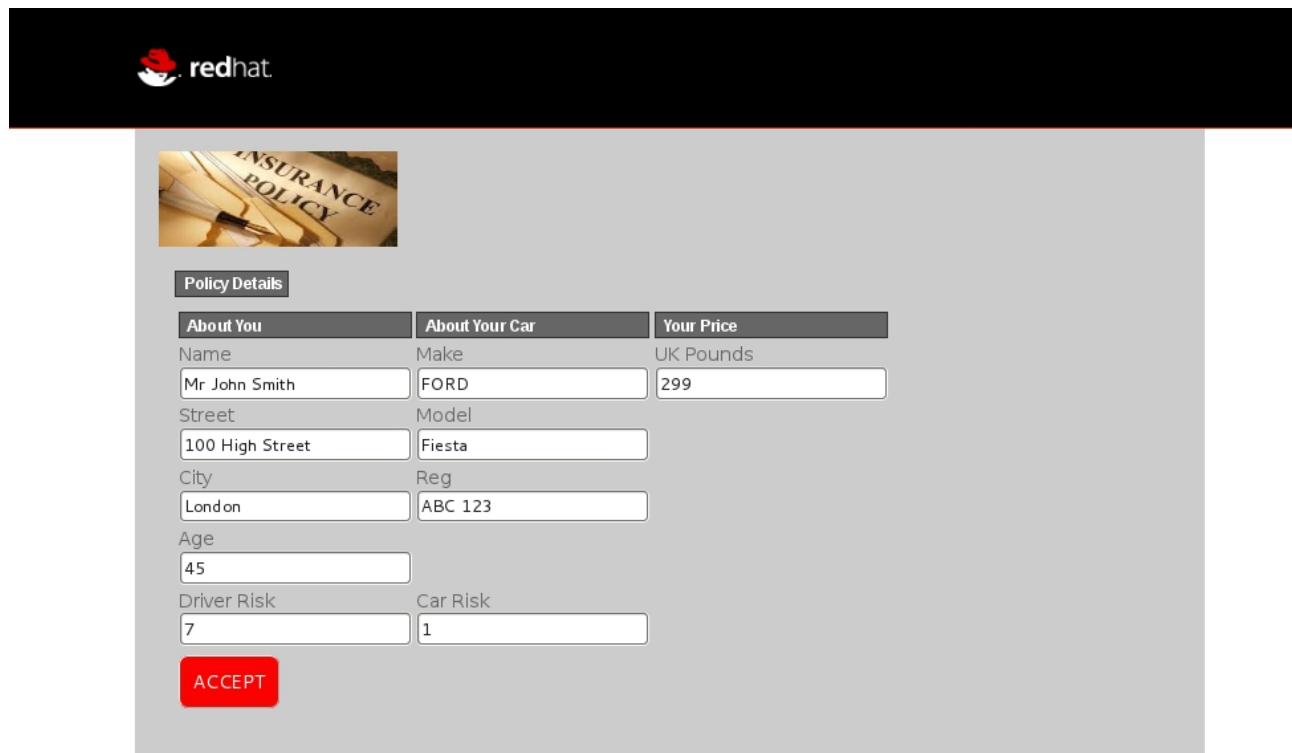
Now from the JBPM-CONSOLE you can now see the process has moved on to the getCar step.
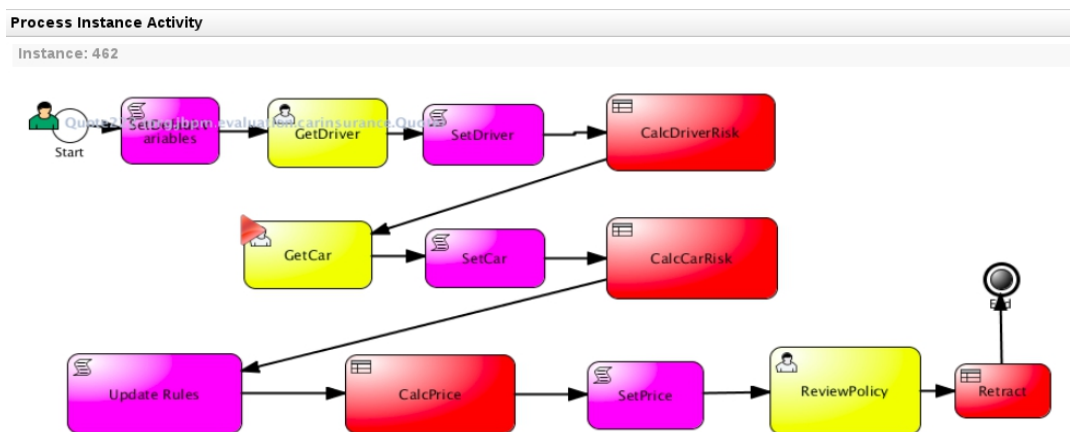
Select a FORD Fiesta from the drop down list and click NEXT.

Review Policy Form

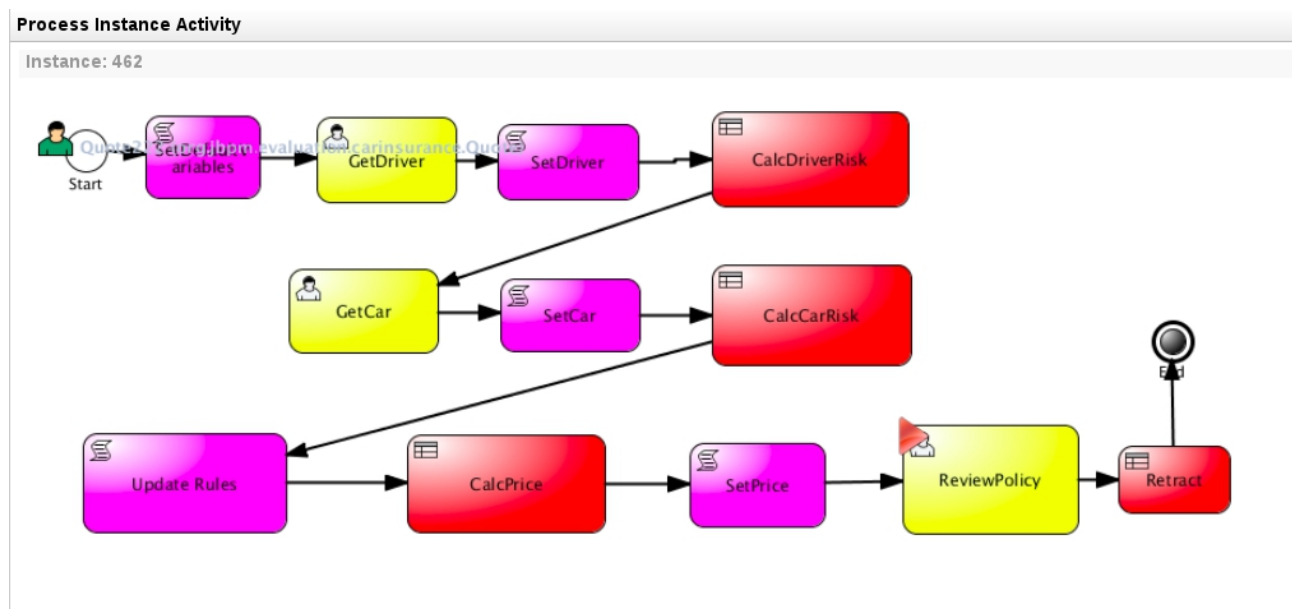The Final form displays the quote. This is the review policy step in the process.

Again you can see the process has advanced to the review policy step.

Retract Objects from Memory

From the window you started the BRMS from you should now see the log file detail the three retract statements.

```
11:33:58,524 INFO  [RepositoryServlet] PackageVersion: LATEST
11:33:58,524 INFO  [RepositoryServlet] PackageIsLatest: true
11:33:58,524 INFO  [RepositoryServlet] PackageIsSource: true
11:34:51,146 INFO  [STDOUT] Retracting Car
11:34:51,146 INFO  [STDOUT] Retracting Policy
11:34:51,146 INFO  [STDOUT] Retracting Driver
```

Once accepted the review policy will display the complete form.
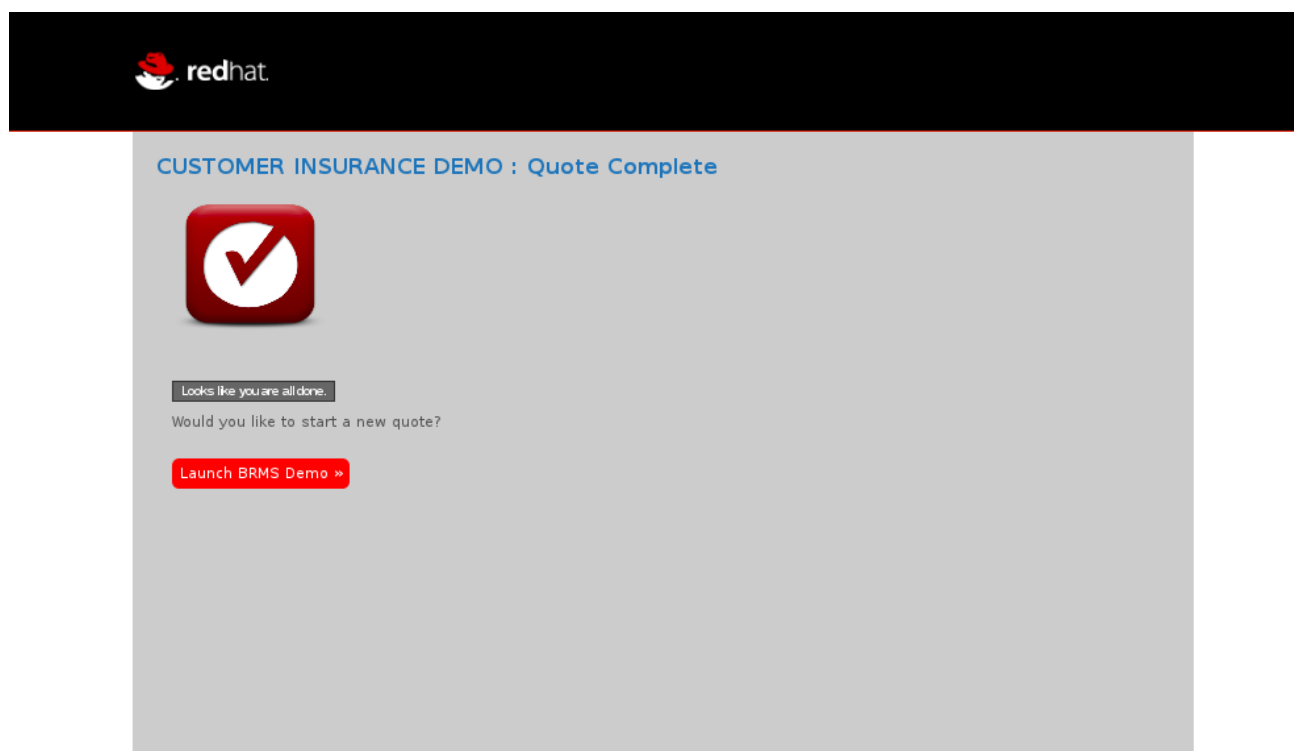
Click new_quote to step through the demo again, this time experiment with different driver ages and different car types to see the range of prices. You can edit the prices from the BRMS rules section, in the calcPrice decision table.