OpenRules®

Open Source Business Decision Management System

Release 6.1.3

Getting Started

OpenRules, Inc.

www.openrules.com

December-2011

Table of Contents

INTRODUCTION

Business Decision Management

OpenRules® was started in 2003 by OpenRules, Inc. as an open source Business Rules Management System (BRMS) and since then became one of the most popular BRMS on the market. It is oriented to subject matter experts (business analysts, not programmers) allowing them to represent, test, and maintain their business logic. Today OpenRules® is a winner of several software awards and is used worldwide by multi-billion dollar corporations, major banks, insurers, health care providers, government agencies, online stores, universities, and many other institutions.

OpenRules® offers the following decision making components:

- Rule Repository for management of enterprise-level decision rules
- Rule Engine for rules execution
- Rule Learner for rule discovery and predictive analytics
- Rule Solver for solving constraint satisfaction and optimization problems
- Finite State Machines for event processing and "connecting the dots"
- Rules Dialog for building rules-based Web questionnaires.

Integration of these components with executable decisions has effectively converted OpenRules® Release 6 from a BRMS to a BDMS, Business Decision Management System, oriented to "decision-centric" application development.

This document helps a user to get started with OpenRules®. It describes how to install OpenRules® and use it starting with simple examples. This document is aimed at business analysts and software developers who may assist them in development of decision making applications. More information is available in the User Manual.

Document Conventions

The regular Century Schoolbook font is used for descriptive information.

The italic Century Schoolbook font is used for notes and fragments clarifying the text.

```
The Courier New font is used for code examples.
```

OPENRULES DECISIONS - INTRODUCTORY EXAMPLE

This section will demonstrate how to build simple executable decision projects with underlying decision tables using only OpenRules® and Excel. We will take a very simple example (similar to a traditional "Hello World!") and will explain how to implement it as an executable decision project.

Decision "DetermineCustomerGreeting"

The following example demonstrates how to develop a very simple application that should decide how to greet a customer during different times of the day. The proper decision might be a part of an interactive voice response (IVR) system. For example, if a customer Robinson is a married woman and local time is 14:25, we want our decision to produce a greeting like "*Good Afternoon, Mrs. Robinson!*"

We will use Excel to represent decisions, related decision tables, and several test cases. Then we will demonstrate how to connect them with data coming from two different sources:

- From Excel Data tables
- From Java objects.

Starting with Decision

We will call our decision "DetermineCustomerGreeting" because it should "Determine" (a decision word) the variable "Customer Greeting". To make this decision we should make two sub-decisions:

- Define Greeting Word (e.g. "Good Afternoon")
- Define Salutation Word (e.g. "Mrs.")

We will need to create rules for each of these sub-decisions. OpenRules® provides special tables for representing decisions where each sub-decision is implemented as a Decision Table. So, first we will create an Excel Decision table "DetermineCustomerGreeting":

Decision DetermineCustomerGreeting	
Decisions	**Execute Rules**
Define Greeting Word	:= DefineGreeting()
Define Salutation Word	:= DefineSalutation()

The first column of this table defines the decision name and the second column defines

the decision table that implements the related (sub) decision.

Note. The second column uses ":=" in front of decision table names and "()" after them because this table actually executes the proper decision tables as Java methods in the top-down order.

Defining Decision Tables

Let's first define a decision table that determines a greeting word:

DecisionTable DefineGreeting	
If	Then
Current Hour	**Greeting**
0-11	Good Morning
11-17	Good Afternoon
17-22	Good Evening
22-24	Good Night

Here the first row defines the name "DefineGreeting" of the decision table following the keyword "DecisionTable" (without space). The second row specifies types of columns in this decision table using "If" for conditions and "Then" for actions. The third row describes variables for the proper columns. The decision table "DefineGreeting" uses the variable "Current Hour" as a condition and the variable "Greeting" for the corresponding action.

OpenRules® allows a user to define decision tables in different ways. For example, the methodological approach described in the book "The Decision Model" uses a special type of decision tables (called "rule families"), for which:

- all conditions and actions should consist of two sub-columns for operators and values
- only one conclusion (action) is allowed.

To satisfy these requirements, we may use keywords "Condition" and "Conclusion" instead of "If" and "Then". In the next example we define a decision table functionally equivalent to the previous one:

DecisionTable DefineGreeting					
Condition		Condition		Conclusion	
Current Hour		**Current Hour**		**Greeting**	
>=	0	<=	11	Is	Good Morning
>=	11	<=	17	Is	Good Afternoon
>=	17	<=	22	Is	Good Evening
>=	22	<=	24	Is	Good Night

In our example all rules contain operators and operands appropriate to the variable in the column headings. For instance, the second rule can be read as:

> *"IF Current Hour is more or equal to 11 AND Current Hour is less or equal to 17 THEN Greeting is Good Afternoon".*

Similarly, we may define a decision table "DefineSalutation" that determines a salutation word:

DecisionTable DefineSalutation					
Condition		Condition		Conclusion	
Gender		**Marital Status**		**Salutation**	
Is	Male			Is	Mr.
Is	Female	Is	Married	Is	Mrs.
Is	Female	Is	Single	Is	Ms.

Defining Business Glossary

Previous rules dealt with the following variables:

- Current Hour
- Gender
- Marital Status
- Greeting
- Salutation

We need to associate these variables with business concepts and their attributes. We will assume that Gender and Marital Status are related to the business object Customer and Current Hour, Greeting, Salutation are related to an object Response that is used as an output for our decision "DetermineCustomerGreeting". We will assume that an instance of Customer is defined by an Excel method "customer()" and an instance of Response is defined by the method "response()". Now we may define the proper business glossary that basically connects all (!) variables with concrete data:

Glossary glossary		
Variable Name	**Business Concept**	**Attribute**
Gender	Customer	gender
Marital Status		maritalStatus

Greeting	Response	greeting
Salutation		salutation
Current Hour		hour

Defining Test Data in Excel

Now, we want to test the developed decision. Initially we will define test data using OpenRules® Datatype and Data tables. First, we define two Datatypes "Customer" and "Response":

Datatype Customer	
String	**name**
String	**maritalStatus**
String	**gender**
int	**age**

Datatype Response	
String	**greeting**
String	**salutation**
int	**hour**

Then we create several test-customers:

Data Customer customers			
name	maritalStatus	gender	age
Customer Name	**Marital Status**	**Gender**	**Age**
Robinson	Married	Female	24
Smith	Single	Male	19

and one variable for a response:

Variable Response response		
greeting	salutation	hour
Greeting	**Salutation**	**Current Hour**
?	?	14

All variables inside the response should be defined decision tables.

Finally, we will use a table of the predefined type "DecisionObject" to map business concepts Customer and Response (defined in the glossary above) with business objects defined in the test data:

DecisionObject decisionObjects	
Business Concept	**Business Object**
Customer	:= customers[0]
Response	:= response

Executing Decision

Now we are ready to execute the defined decision against the above test data. You may run a standard OpenRules launcher for our decision "DetermineCustomerGreeting" by double-click on the batch file "run.bat" in the standard OpenRules project "DecisionHello". It will produce the following results:

```
Define Current Time
Conclusion: Current Hour is 14
Define Greeting Word
Conclusion: Greeting is Good Afternoon
Define Salutation Word
Conclusion: Salutation is Mrs.
```

Adding Rules That Define Current Hour

Instead of using a fixed Current Hour specified in the test data as 14, we may define a simple decision table that sets Current Hour based on the actual time of the day. The following decision table "DefineCurrentHour" consists of one unconditional action:

DecisionTable DefineCurrentHour	
Conclusion	
Current Hour	
Is	::= Calendar.getInstance().get(Calendar.HOUR_OF_DAY)

As you can see, it uses a Java's snippet with the standard Java Calendar class to set the Current Hour. Of course, you need to know Java to write such a formula, and a software developer can always help you to create the proper Java snippet. We also may get the current hour from a calling application by putting the following Java snippet inside this cell

```
::= decision().get("hour")
```

Here the characters "`::=`" serve as an indicator that a snippet of Java code is used. Showing this code, we simply want to demonstrate that you still may use the entire power of the Java library already built into OpenRules®.

To make this new decision table a part of our decision we have to add another row in our decision table "DetermineCustomerGreeting":

Decision DetermineCustomerGreeting	
Decisions	**Execute Rules**
Define Current Time	:= DefineCurrentHour()
Define Greeting Word	:= DefineGreeting()
Define Salutation Word	:= DefineSalutation()

If we run this decision say at 20:45 PM local time, it will produce the following results:

```
Define Current Time
Conclusion: Current Hour is 20
Define Greeting Word
Conclusion: Greeting is Good Evening
Define Salutation Word
Conclusion: Salutation is Mrs.
```

We may add one more sub-decision that combines the defined greeting and salutation in one final greeting for this particular customer:

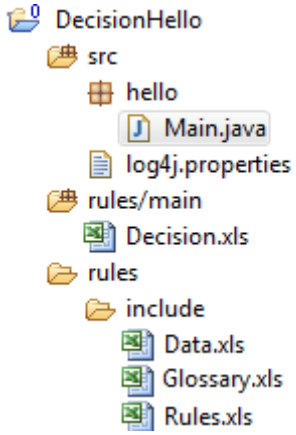
Decision DetermineCustomerGreeting	
Decisions	**Execute Rules**
Define Current Time	:= DefineCurrentHour()
Define Greeting Word	:= DefineGreeting()
Define Salutation Word	:= DefineSalutation()
Define Customer Greeting	:= "Decision: " + response.greeting + ", " + response.salutation + customers[0].name + "!")

This decision will produce the following results:

```
Define Current Time
Conclusion: Current Hour is 20
Define Greeting Word
Conclusion: Greeting is Good Evening
Define Salutation Word
Conclusion: Salutation is Mrs.
Decision: Good Evening, Mrs.Robinson!
```

Decision Project

The described decision is available in the project "DecisionHello" included in the OpenRules® installation. This project has the following structure:



The folder "rules" is a rules repository. Its sub-folder "rules/main" contains the main methods for different decisions. For this example, it is one file "Decision.xls" with the table "Decision DetermineCustomerGreeting". The sub-folder "rules/include" contains Excel files for data, glossary, and decision tables described above. The file "src/hello/Main.java" contains the standard Java launcher for executing Main.xls. Here is its code:

```
public class Main {
  public static void main(String[] args) {
    String fileName = "file:rules/main/Decision.xls";
    Decision decision = new Decision("DetermineCustomerGreeting",fileName);
    decision.execute();
  }
}
```

A user may distribute Excel tables between xls-files placed in different directories and subdirectories. To define a desired structure for the rules repository, a user should include a special OpenRules® table "Environment" in the main file "Decision.xls". In this example the "Environment" table looks as follows:

Environment	
include	../include/Rules.xls
	../include/Data.xls
	../include/Glossary.xls
	../../../openrules.config/DecisionTemplates.xls

The "../include/" in front of files Rules.xls, Data.xls, and Glossary.xls point to the location of these files relative to the file rules/main/Decision.xls. All decision tables utilize the standard file "DecisionTemplates.xls" that is located in the OpenRules® configuration project "openrules.config" (3 levels above rules/main).

Accepting Data from Java

In the real-world, decisions are incorporated in business applications that supply the decisions with data and expect to receive back data-specific decisions. Above we have

shown how to define data types and data instances in Excel tables. Now we will explain how the same decision deals with data defined in Java objects. The standard OpenRules® installation includes the proper decision project "DecisionHelloJava".

This project has a Java package "hello" with two Java classes "Customer" and "Response" that are simple Java beans with the following organization:

```
public class Customer {
      String   name;
      String   maritalStatus;
      String   gender;
      int      age;
      double   salary;
      Date     dob;
      boolean  retired;
      // getters and setters
}

public class Response {
      int    hour;
      String greeting;
      String salutation;
      String result;
     // getters and setters
}
```

The main Java class Main.java now additionally creates instances of the classes Customer and Response, puts them to the instance of Decision, and executes OpenRulesEngine for this decision:

```
public static void main(String[] args) {
    String fileName = "file:rules/main/Decision.xls";
    Decision decision = new Decision("DetermineCustomerGreeting",fileName);
    Customer customer = new Customer();
    customer.setName("Robinson");
    customer.setGender("Female");
    customer.setMaritalStatus("Married");
    customer.setAge(20);
    decision.put("customer", customer);
    Response response = new Response();
    decision.put("response", response);
    decision.execute();
}
```

This code uses a predefined OpenRules® class "Decision" that extends HashMap and allows a user to put and get any object to the decision using a keyword like

```
decision.put("customer", customer);
```

To use custom Java objects of types Customer and Response within our Excel-based decision, we have to change the table "DecisionObject" that maps business concepts Customer and Response (defined in the glossary) with business objects defined in Java:

DecisionObject decisionObjects	
Business Concept	**Business Object**
Customer	:= decision().get("customer")
Response	:= decision().get("response")

We do not utilize the file "Data.xls" anymore, so we should not refer to it in the Environment table. However, we need to use a new line with "import.java" to mention custom Java packages such as "hello.*" that are used by our application:

Environment	
import.java	hello.*
include	../include/Rules.xls
	../include/Glossary.xls
	../../../openrules.config/DecisionTemplates.xls

SUMMARY OF INTRODUCED CONCEPTS

In the above introductory example we demonstrated the use of several predefined OpenRules® tables:

- **Decision** – to specify a decision structure as a combination of intermediate decisions and related decision tables
- **DecisionTable** – to specify business logic behind different decisions
- **Glossary** – to specify all used decision variables and related business concepts and their attributes. The glossary remains independent of any particular implementation of business objects that could be defined in Excel data tables, in Java classes, or in XML.
- **DecisionObject** – to map business concepts defined in a glossary with concrete business objects defined in Java or Excel.
- **Datatype** – to specify data types for test data
- **Data** – to specify concrete test instances
- **Method** – to specify access to different objects (like Excel Data or Java-based instances) or using Java snippets to express more complicated calculations (like local time)
- **Environment** – to specify a structure of the decision projects.

This example demonstrated how business analysts can create and test decisions without coding. At the same time it demonstrated how software developers can help to integrate a tested decision into an existing Java application.

The following simple decision projects will demonstrate a different use of already introduced concepts.

DECISION "DETERMINEVACATIONDAYS"

Now we will show how to build an executable decision with calculation formulas. Our decision should specify how to calculate vacation days for different categories of employees. This example was proposed by Prof. J. Vanthienen. The number of vacation days depends on age and years of service following these rules:

- Every employee receives at least 22 days.
- Additional days are provided according to the following criteria:
  - Only employees younger than 18 or at least 60 years, or employees with at least 30 years of service will receive 5 extra days.
  - Employees with at least 30 years of service and also employees of age 60 or more, receive 3 extra days, on top of any days already given.
  - If an employee has at least 15 but less than 30 years of service, 2 extra days are given. These 2 days are also provided for employees of age 45 or more. The 2 extra days cannot be combined with the 5 extra days.

This time we implement the decision using a single Excel file DecisionVacationDays.xls – see the standard OpenRules project "DecisionVacationDays". Our decision will consist of two steps as described in the following decision table:

Decision DetermineVacationDays	
Decisions	**Execute Rules**
Define Vacation Days	:= DefineVacationDays()
Show Results	:= System.out.println(employee())

The first step will execute the decision table "DefineVacationDays" that should calculate a number of vacation days for an employee and the second step will simply print out this employee.

Let's start with defining the rules. There are many different ways to do it but here is an example of the proper decision table:

DecisionTable DefineVacationDays		
If	If	Then

Age in Years	Years of Service	Vacation Days
<18		::= (22 + 5)
[18;45)	<15	::= 22
[18;45)	[15;30)	::= (22 + 2)
[18;45)	>=30	::= (22 + 5 + 3)
[45;60)	<15	::= (22 + 2)
[45;60)	[15,30)	::= (22 + 2)
[45;60)	>=30	::= (22 + 5 + 3)
60+		::= (22 + 5 +3)

We do not use operators and instead we define different intervals of age and years of service directly with their values. That's why instead of the word "Condition" (that assumes the presence of an operator column) we use the word "If" to specify conditions. Inside our action column of the type "Then" we use formulas that start with "::=". Each cell defines the basic 22 days plus additional days according to the specified conditions. This rule table uses 3 variables that we should specify in the problem glossary:

Glossary glossary		
Variable Name	**Business Concept**	**Attribute**
Age in Years	Employee	age
Years of Service		service
Vacation Days		days

Now we will define a new type Employee using the following datatype table

Datatype Employee	
String	id
int	age
int	service
int	days

and create several test-employees of this type:

Data Employee employees			
id	age	service	days
ID	**Age in Years**	**Years of Service**	**Vacation Days**
A	17	1	0
B	25	5	0
C	45	30	0

D	64	42	0

To refer to a selected employee we are using the method `employee()` defined as

Method Employee employee()
return employees[0];

To map a business concept "Employee" from the glossary to a concrete employee object, we define the following table:

DecisionObject decisionObjects	
Business Concept	**Business Object**
Employee	:= employee()

We do not use any additional files or Java classes, so our Environment table only refers to the standard decision templates:

Environment	
include	../../openrules.config/DecisionTemplates.xls

Now we are ready to run this decision that will produce the following results:

```
Define Vacation Days
Show Results
Employee(id=0){
  id=A
  age=17
  days=27
  service=1
}
```

If we change the method `employee()` to point to `employees[2]`, the decision will produce the following results:

```
Define Vacation Days
Show Results
Employee(id=0){
  id=C
  age=45
  days=30
  service=30
}
```

DECISION DETERMINEUPSELLOFFERS"

Next we will show how to build an executable decision that demonstrates how to define complex up-sell rules based on existing and available financial products. The lists of products and services can be easily adjusted to any industry. Based on a customer profile (that also should be defined by rules) and products that s/he already has, our

decision should offer a set of up-sell products. In this example the decision deals not with single objects but with arrays of objects.

We will implement our decision using a single Excel file DecisionUpSell.xls – see the standard OpenRules project "DecisionUpSell". Our decision will consist of three steps as described in the following decision table:

Decision DetermineUpSellOffers	
Decisions	**Execute Rules**
Define Customer Profile	:= DefineCustomerProfile()
Define Up-Sell Products	:= DefineUpSellProducts()
Show Results	:= System.out.println(response())

The first step will execute the decision table "DefineCustomerProfile"; the second one will execute the decision table "DefineUpSellProducts", and the third step will simply show the results by printing out a response from the decision. We will use a Java class Response defined in the package "upsell" that contains an array `String[] products`. The object "response" will be passed to our decision through the decision from the following launcher in the file Main.java:

```
import com.openrules.ruleengine.Decision;

public class Main {

   public static void main(String[] args) {
        String fileName = "file:rules/DecisionUpSell.xls";
        Decision decision = new Decision("DetermineUpSellOffers",fileName);
        Response response = new Response();
        decision.put("response",response);
        decision.execute();
        System.out.println(response);
   }
}
```

We will access this object using the Excel method `response()` as described as:

Method Response response()
return decision().get("response");

A customer, to whom we want to up-sell products will be defined directly in Excel using the method `customer()`:

Method Customer customer()
return customers[0];

We will specify a type Customer later when we know what Customer's attributes we actually need to support our up-selling rules. Let's start with defining the customer profiling rules:

DecisionTable DefineCustomerProfile			
Condition		Conclusion	
Combined Balance		**Customer Profile**	
Within	0-500	Is	New
Within	500-2000	Is	Bronze
Within	2000-5000	Is	Silver
Within	5000-15000	Is	Gold
Within	15000-10000000	Is	Platinum

The first condition uses the operator "Within" for the variable "Combined Balance". For example, the second rule states that if a customer's Combined Balance is within an interval 500-2000 (inclusively) then their Customer Profile is Bronze.

The up-selling rules will offer new products to a customer based on his/her profile and the products that this customer already has. We will use operators that deal with multiple instances separated by commas. For example, the operator "Is One Of" will be used to check if Customer Profile is one on the following: New, Bronze, Silver. Similarly, we will use the operator "Include" to check if Customer Products (those that she already has) include such products as: Checking Account, Overdraft Protection. We also use the operator "Do Not Include" to check if Customer Products (those that she already has) do not include such products as Checking Account, CD, Credit Card. Here is the proper decision table:

DecisionTable DefineUpSellProducts								
Condition		Condition		Condition		Conclusion		Message
Customer Profile		**Customer Products**		**Customer Products**		**Offered Products**		**Set Comment**
Is One Of	New,Bronze,Silver	Include	Checking Account	Do Not Include	Saving Account	Are	Saving Account, Debit/ATM Card, Web Banking	
Is One Of	New,Bronze,Silver	Include	Checking Account, Overdraft Protection	Do Not Include	CD with 25 basis point increase, Money Market Mutual Fund, Credit Card	Are	CD with 25 basis point increase, Money Market Mutual Fund, Credit Card	
Is One Of	New,Bronze,Silver	Include	Checking Account, Saving Account	Do Not Include	CD with 25 basis point increase, Money Market Mutual Fund, Credit Card	Are	CD with 50 basis point increase, Money Market Mutual Fund, Credit Card, Debit/ATM Card, Web Banking	
Is One Of	Gold	Include	Checking Account	Do Not Include	CD with 25 basis point increase, Money Market Mutual Fund, Web Banking	Are	CD with 50 basis point increase, Money Market Mutual Fund, Credit Card, Debit/ATM Card, Web Banking, Brokerage Account	Gold Package
Is One Of	Platinum	Include	Checking Account, Saving Account	Do Not Include	CD with 25 basis point increase, Money Market Mutual Fund, Web Banking	Are	CD with 50 basis point increase, Money Market Mutual Fund, Credit Card with no annual fee, Debit/ATM Card, Web Banking with no charge, Brokerage Account	Platinum Package

The conclusion "Offered Products" uses the operator "Are" (instead of the regular "Is") to define a set of offered products separated by commas.

Now we are ready to define a glossary that lists all variables used by our two decision tables:

Glossary glossary		
Variable Name	**Business Concept**	**Attribute**
Customer Profile	Customer	profile
Combined Balance		combinedBalance
Customer Products		products
Offered Products	Response	products

Now we may define a type Customer using the following datatype table

Datatype Customer	
String	firstName
String	middleInitial
String	lastName
int	age
Date	customerFrom
String	state
String[]	**products**
int	combinedBalance
String	profile

and create several test-customers using this vertical data table:

Data Customer customers			
firstName	**First Name**	John	Mary
middleInitial	**Middle Initial**	D.	K.
lastName	**Last Name**	Smith	Smith
age	**Age**	28	25
customerFrom	**Customer From Date**	10/15/1998	3/10/2002
state	**State**	NJ	NJ
products	**Products**	Checking Account	Checking Account
		Saving Account	
combinedBalance	**Combined Balance**	$12,000.00	$10,000.00
profile	**Profile**	?	?

Please observe that each customer can have an array of products that are defined in different sub-rows of the table "customers". To map business concepts "Customer" and "response" from the glossary to concrete objects, we define the following table:

DecisionObject decisionObjects	
Business Concept	**Business Object**
Customer	:= customer()
Response	:= response()

This decision uses an external Java class "Response" from the package "upsell", so our Environment table should refer to this package:

Environment	
include	../../openrules.config/DecisionTemplates.xls
import.java	upsell.*

Now we are ready to run this decision. The above Java launcher will produce the following results:

```
Define Customer Profile
Conclusion: Customer Profile Is Gold
Define Up-Sell Products
Conclusion: Offered Products Are CD with 50 basis point increase, Money
Market Mutual Fund, Credit Card, Debit/ATM Card,  Web Banking, Brokerage
Account
Gold Package from DefineUpSellProducts
Show Results
Offered Products:
      CD with 50 basis point increase
      Money Market Mutual Fund
      Credit Card
      Debit/ATM Card
```

```
Web Banking
Brokerage Account
```

INSTALLATION

Download and Run

OpenRules® can be downloaded from http://openrules.com/download.htm. You will download one file "`openrules.decisions.zip`" that should be unzipped at your hard-drive (just avoid spaces in folder names). After unzip, the folder "`openrules.decisions`" will contain executable decision projects described in this document and several additional sample projects. OpenRules libraries and templates necessary for the execution of these projects are located in the configuration project "`openrules.config`". You may execute any project by double-click on the file "`run.bat`" from a regular Window Explorer. If you use Linux, you should rename "`run.bat`" to "`run.sh`" (and make the proper changes).

Pre-Requisites

OpenRules requires:

- Java 2 Standard Edition: J2SE Development Kit release 1.5 or higher
- Apache Ant 1.6 or higher*

OpenRules works in any operating environment with standard Java. If you use Linux/Unix you are probably already well aware how to deal with Java and Ant. If you use Windows and are not very familiar with Java/Ant configuration issues, below are straight-forward recommendations:

1. Install Windows Platform - J2SE Development Kit 6u1 (or later) from http://java.sun.com/javase/downloads/index.jsp. Accept all defaults, and your JDK will be installed at c:\Program Files\Java\jdk1.6.0_xx, where xx - is an actual number of the release you downloaded
2. Install Current Release of Ant (1.6.5 or later) from http://ant.apache.org/bindownload.cgi. By default, you will install it at c:/apache-ant-1.x.y After the installation rename c:/apache-ant-1.x.y to c:/ant.
3. Now you want to set up your environment variables in a way similar to the one below:

   set ANT_HOME=c:\ant
   set JAVA_HOME=c:\Program Files\Java\jdk1.6.0_xx
   set PATH=%PATH%;%ANT_HOME%\bin;%JAVA_HOME%\bin

   To set up environment variables, left-click on the Windows Start button, then right-click on My Computer, and chose Properties. Select the tab "Advanced" and then push

"Environment Variables". Add or correct User variables JAVA_HOME, ANT_HOME, and PATH accordingly.

4. Click on Start + Run and enter "cmd" to open a DOS window. Type

   >java -version
   >ant -version

   to make sure that you use the correct versions of Java and Ant.

TECHNICAL SUPPORT

Direct all your technical questions to support@openrules.com.