OPENRULES®

Open Source Business Decision Management System

Release 6.2.1

Database Integration

OpenRules, Inc.

www.openrules.com

June-2012

TABLE OF CONTENTS

INTRODUCTION

OpenRules® is as an open source Business Decision Management System (BDMS) with proven records of delivering and maintaining reliable decision support software. The detailed description of OpenRules® can be found in the User Manual.

This manual is devoted to an integration of OpenRules® with relational databases. There are two aspects of such integration:

1. Accessing data located in a database
2. Saving and maintaining rules in a database as Blob objects
3. Using External Rules to access custom rule repositories.

The standard OpenRules® installation comes with sample projects that explain how to use access database and use DB-protocols to create and maintain business rules.

ACCESSING DATA LOCATED IN DATABASE

You may use any 3rd party tools to access data from a database and use it to create Java objects that will be processed by OpenRulesEngine or Decisions. At the same time, OpenRules® provides simple JDBC-based interfaces for accessing relational databases.

Simple DB Access Library

The standard OpenRules® installation comes with the library "***com.openrules.tools.jar***" that contains different convenience Java classes. In particular, it include such classes as Database, DatabaseIterator, and DBUtil that utilize the standard JDBC interface. The sources of the proper Java classes are handily available as a project "com.openrules.tools".

OpenRules® provide a convenience Java class DatabaseIterator that allows you to iterate through any table of a relational database. You may use standard Data Source configuration facilities to define a data source with the name "dbName" that point to a concrete database created using MS Access, MS SQL, or Oracle. Assuming that this database includes a table with the name "tableName", you should be able to iterator through this table using the following pseudo-code:

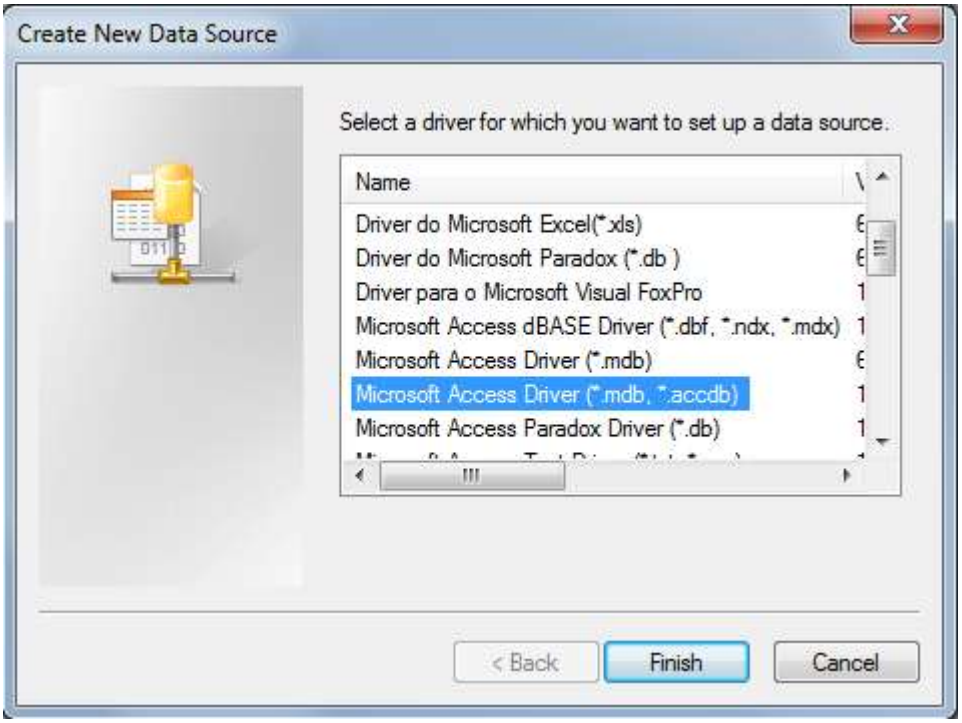
```
DatabaseIterator iter = new DatabaseIterator(dbName, tableName);
while (iter.hasNext()) {
     DynaBean bean = iter.next();
     // Create an object from a data record
     MyObject object = new MyObject();
     object.setAttr1( (String) bean.get("attr1 name") );
     Double attr2 = (Double) bean.get("attr2 name");
     object.setAttr2(attr2.doubleValue());
     …

     // execute OpenRulesEngine or Decision for this object
}
iter.close();
```

DatabaseIterator utilizes Apache Commons BeanUtils. We will demonstrate the use of DatabaseIterator in the example below.

Example "HelloCustomerDB" with DB-based Customers

This example included in the workspace "openrules.rules" demonstrates how to read customers from a database and process them with an instance of OpenRulesEngine. MS Access comes with a standard database "DBStudents.accdb" containing basic information about students. We consider students as our customers.

First, we will make this database known to our application by adding the proper data source. To do this in MS Windows we need to open a Control Panel, select Administrative Tools, and then "Set up data sources (ODBC)". It will open "ODBC Data Source Administrator". Select the tab "System DSN" and click on "Add". You will see the following dialog:



Select "Microsoft Access Driver (*mdb,*accdb)" and click on "Finish".

Note. If you have problems to find Microsoft Access Driver, read http://goo.gl/w4vNo or http://msdn.microsoft.com/en-us/library/windows/desktop/ms712362(v=vs.85).aspx.

In the next dialog enter Data Source Name "HelloStudents", then any description, and click on "Select" to select already created file "DBStudents.accdb". Then click "OK". Now we should be able to read the table "Students" it from a Java program using a JDBC interface.

We will reuse the same Java classes Customer and Response that were used in the basic OpenRules example "HelloJava". To map only necessary fields from a database to our class we will use dynamic beans:

```
public static void main(String[] args) {
	String fileName = "file:rules/main/HelloCustomer.xls";
	String methodName = "helloCustomer";

	OpenRulesEngine engine = new OpenRulesEngine(fileName);
	String dbName = "HelloStudents";
	String tableName = "Students";
```

```
        DatabaseIterator iter = new DatabaseIterator(dbName, tableName);
        while (iter.hasNext()) {
               DynaBean bean = iter.next();
               // Create a customer from a data record
               Customer customer = new Customer();
               String first = (String)bean.get("first name");
               String last = (String)bean.get("last name");
               String gender = (String)bean.get("gender");
               String maritalStatus = (String)bean.get("marital status");
               customer.setName(first + " " + last);
               customer.setGender(gender);
               customer.setMaritalStatus(maritalStatus);

               Response response = new Response();
               Object[] objects = new Object[] { customer, response };
               engine.run(methodName,objects);
               System.out.println("From Java : " +
                              response.getMap().get("greeting") + ", " +
                              response.getMap().get("salutation") +
                              customer.getName() + "!"
               );
        }
        System.out.print("\nRead and processed total " + n + " records");
        iter.close();
  }
```

This program will read every student from the table “Students”, convert it to an instance of Customer, and run it through the proper greeting rules. Note that the engine is created only once and processes all customers.

Example "DecisionPatientTherapyDB" with DB-based Patients

This example included in the workspace “openrules.decision” demonstrates how to read customers from a database and process them with an instance of the class Decision. It utilizes the same database this time using students as patients. We will define the class PatientIterator as a subclass of the class DatabaseIterator that reads and converts students to patients. Instead of OpenRulesEngien this example uses decisions. The xdetailed description of this project can be found at http://openrules.com/pdf/Tutorial.DecisionPatientTherapyDB.pdf.

CREATING AND MAINTAINING RULES IN DATABASE

The key component of OpenRules® is an enterprise-level Rule Repository that usually utilizes a popular spreadsheet mechanism to represent business rules placed in regular Excel files. Along with rules repositories organized as a hierarchy of Excel files, OpenRules® allows you to use standard relational databases to keep and maintain your business rules. To do this, you may place your Excel files with OpenRules® tables into any relational database as Blob objects. OpenRules® provides a direct access to Excel files saved in a database without necessity to download them into a file system. OpenRules® supports several protocols that handle DB-based rules repositories:

- **Simple DB protocol "db:"** this protocol supports one table in which all Excel files are saved as Blob objects with unique keys that usually correspond to relative paths of these files in a file system. It supports a very simple protocol *"**db:<filepath>**"* such as "db:/hello/rules/include/HelloRules.xls". The provided example shows how to configure this protocol using one Excel-based Environment table and a file “db.properties”.
- **DB protocol "db:" without Excel** the same “db:” protocol can be configured without use of any external (not DB-based) configuration files.
- **Versioning DB protocol "dbv:"** this protocol additionally to the features provided by the simple DB protocol allows a user to maintain different rules versions and provides checkin/checkout facilities. It supports the protocol *"**dbv://<repository>:<version>/<filepath>**"* such as "db://rules:25/include/DiscountRules.xls".

- **Custom DB protocols** with any user-defined name. A user can customize the standard OpenRules® protocols to take into consideration specifics of their databases.

Simple DB-based Rule Repository

OpenRules® provides a simple protocol "**db:**<filepath>" that allows you to use standard databases to keep and maintain your business rules. To do this, you

may place your Excel files with tables into any relational database as Blob objects (binary large objects). OpenRules® provides a direct access to Excel files saved in a database without necessity to download them into a file system. This protocol supports one database table in which all Excel files are saved as Blob objects with unique keys that usually correspond to relative paths of these files in a file system.

Database Structure

The "db:<filepath>" protocol assumes that all Excel files are saved in one database table called "dbstorage" with the following structure:

- name - a primary key up to 255 characters
- content - a BLOB object

For example, if your Excel files were initially located in the folder "rules/include/" you may copy them to the database using keys such as "/rules/include/FileName.xls". After that, you do not have to change a structure of your OpenRules® Environment table - just use the property "include.path" with the value "db:/rules/include/". All included Excel files that were described in the property "include" as "<FileName.xls>" would be directly available to OpenRulesEngine.

Example "HelloJavaDB" with rules inside Apache Derby DB

This example demonstrates how to convert a basic rules project "HelloJava" to work with rules placed into the standard open source Java database known as Apache Derby. This project is included into the standard OpenRules® installation under the name "HelloJavaDB". It has exactly the same structure as HelloJava - even the Java code was not changed. We have only added a new folder "db" in which we created a Derby database as a placeholder for rules previously kept in the folder "rules". The folder "db" initially contains only two bat-files: *db.bat* and *loadall.bat* that are used to create and administer the database from a command-line interface. Here what we did with HelloJava

project and what you may do with your own rule projects to move their repositories to a database.

1. Eclipse project HelloJavaDB has one additional library in its Java Build Path: it refers to *derby.jar* that we included in the updated *openrules.config/lib*.

2. We created an instance of Derby database "dbstorage" inside the folder "*db*". To do this, we launched a console window (command prompt), navigated to the /db directory, and executed the following command:

   >db -i

3. We added a database configuration file “db.properties” into the folder "rules/main" that already contains the main file *HelloCustomer.xls*.

4. This particular project contains only 3 Excel files:

   1) ***rules/main/HelloCustomer.xls*** the main file for a rules engine to start with. It contains only one Environment table
   2) ***rules/include/HelloRules.xls***: greeting and salutation rules
   3) ***rules/include/HelloMethod.xls***: a table with a method that calls the rules from *HelloRules.xls*.

   The main xls-file *HelloCustomer.xls* continues to be used outside the database serving as an OpenRules® configuration file. To upload two other Excel files to the database we executed the following commands:

   >db -u /HelloRules.xls -f ../rules/include/HelloRules.xls
   >db -u /HelloMethods.xls -f ../rules/include/HelloMethods.xls

   Here "-u" stands for "upload", the string "/HelloRules.xls" is a database key for the rules that were uploaded from the local file "*../rules/include/HelloRules.xls*".

5. You always may check the content of your database by command:

   >db -l ""*

   In our case this command will display:

   List for ''*
   /HelloRules.xls
   /HelloMethods.xls

6. To inform OpenRulesEngine that now it should look rather to the database than to a local file system, we only have to change slightly the main file rules/main/HelloCustomer.xls. It used to contain the following Environment table:

Environment	
include.path	../include/
include	<HelloRules.xls>
	<HelloMethods.xls>
import.java	hello.*
import.static	com.openrules.tools.Methods

7. The modified Environment table will look like here:

Environment	
datasource	**classpath:db.properties**
include.path	**db:/**
include	<HelloMethods.xls>
	<HelloRules.xls>
import.java	hello.*
import.static	com.openrules.tools.Methods

8. Please, note that we did not change the structure of the project that potentially may involve a lot of changes. We only added a new data source that is our database configuration file "db.properties" and changed the property "include.path" that now is based on the protocol "db:". In this case we expect that the file "db.properties" is located in the classpath. However, we also could use any other standard protocol to locate this file, for example "file:rules/main/db.properties".

9. Now, we can execute the rules directly from the database using the same Java program RunHelloCustomer, for example by double-clicking on run.bat.

Database Configuration File "db.properties"

The configuration file "***db.properties***" is used to define a concrete database configuration. Its location is specified in the OpenRules® main xls-file inside an Environment table using the property "*datasource*". Here is an example of the db configuration file for the HelloJavaDB project:

```
=====================================================
# OpenRules Data Source Protocol Properties

openrules.protocol.name=db
openrules.protocol.class=com.openrules.db.DbDataSourceHandler

# DB Access Properties
db.user=embedded
db.password=none
db.url=jdbc:derby:C:/_openrules/openrules.examples/HelloJavaDB/db/dbstorage;create=true
db.driver=org.apache.derby.jdbc.EmbeddedDriver
db.selectSql=select content from dbstorage where name = ?

# Optional DB Administration Properties
db.statementfactory.class=com.openrules.jdbc.StatementFactoryDerbyEmbedded
db.createDDL=CREATE TABLE dbstorage (NAME VARCHAR(255) NOT NULL,CONTENT BLO
B,PRIMARY KEY(NAME))
db.insertSql=insert into dbstorage(name,content) values(?,?)
db.listSql=select name from dbstorage where name like ?
db.deleteSql=delete from dbstorage where name like ?
=====================================================
```

The property "*openrules.protocol.name*" defines the name of the simple protocol provided by OpenRules® with its implementation class described by the property "*openrules.protocol.class*". Both these properties could be customized.

To access the database we have to specify 5 properties:

1) db.user - a user name (here is "embedded" is the default name for Apache Derby).
2) db.password - a user password
3) db.url - defines a physical location of the database. You should make sure that this location corresponds to your file structure (!)
4) db.driver - defines a JDBC driver
5) db.selectSql - shows how the db protocol will access the database.

Other parameters are optional. However, if you use a database command-line administration interface provided by OpenRules® (see examples above), then these 5 properties should be present.

Simple Database Administration Interface

You may use any DB administration interface to create a table "dbstorage" inside your database and to upload Excel files into this table. However, for your convenience OpenRules® provide a simple command-line interface for the DB administration. Here is the list of the available commands and options:

Option "**Initialize**": -i or --init
Usage: db -i
This option initializes a data storage. Warning: existing data will be destroyed

Option "**Upload File**": -u or --up
Usage: db -u <key> -f <local-file-path>
This option uploads the content of <local-file-path> into the database using the <key>. If <local-file-path> is a folder all files and subfolders this folder will be uploaded by adding their relative paths to the <key>.
Example: *db -u /insurance/policy/DriverDiscountRules.xls -f ../rules/insurance/policy/DriverDiscountRules.xls*

Option "**Upload Directory**": -dir or ----from-dir
Usage: db -u <key> -dir <local-dir-path>
This option recursively uploads the entire content of the directory <local-dir-path>
into the database using the <key>.
Example: *db -u /hello/rules/include/ -f ../rules/include/*

Option "**Download**": -d or --down
Usage: db -u <key>
This option downloads data from database using this <key>
Example: *db -d /insurance/policy/DriverDiscountRules.xls -f c:/temp/DriverDiscountRules.xls*
This command will download file DriverDiscountRules.xls into the directory c:/temp.

Option "**Remove**": -rm or --remove

Usage: db -rm "<mask>"

This option removes data from database using this <mask>. The mask can include

wildcards '*' and '?'

Example: *db -rm "*"* will remove all uploaded files from the database

Option "**List**": -l or --list

Usage: db -l

This option lists all keys in the storage. You may use a mask to list only that satisfy the optional mask. The mask can include wildcards '*' and '?'.

Example: *db -l "*"* will display all uploaded files

Option "**File**": -f or --file

Usage: db -f <local-file-path> ...

This option is used with options -u and -d.

Option "**Help**": -h or --help

Usage: db -h

This option displays a list of all options.

Example "RulesRepositoryDB" with rules inside Java Derby DB

This example demonstrates how to convert a more complex rules hierarchy from a file system to a database. We will take the directory "rules" from a standard OpenRules® example "RulesRepository" as a basis and will convert it to the new project "RulesRepositoryDB". Here are the major conversion steps.

1. Eclipse project RulesRepositoryDB has one additional library in its Java Build Path: it refers to *derby.jar* that is included in the updated *openrules.config/lib*.

2. Create an instance of Derby database "dbstorage" inside the folder "db". To do this, we executed the following command from a system console (being inside

the db directory):

>db -i

3. Add a database configuration file db.properties into the folder "rules/main" that already contains the main file Main.xls.

4. This project contains many Excel files:
rules/main/Main.xls the main file for a rules engine to start with. It contains only the Environment table
rules/CategoryA/RulesA1.xls
rules/CategoryA/RulesA2.xls
rules/CategoryA1/RulesA1.xls
rules/CategoryA1/SubCategoryA1/RulesA11.xls
rules/CategoryA1/SubCategoryA1/RulesA12.xls
rules/CategoryB/RulesB1.xls
rules/CategoryB/RulesB2.xls
rules/Common/libA/libRulesX.xls
rules/Common/libA/libRulesY.xls

The main xls-file Main.xls will continue to be used outside the database serving as a OpenRules® configuration file. To upload all other Excel files to the database we may execute many command like this one:

>db -u /examples/RulesRepositoryDB/CategoryA/RulesA1.xls -f ../rules/CategoryA/RulesA1.xls

or alternatively we may execute only one command:

>db -u /examples/RulesRepositoryDB/rules -f ../rules

to copy all files and subfolder starting from ../rules folder into the database. All files from the directory "../rules" will be uploaded. The proper keys will be

created automatically. If now we enter the command:

>db -l ""*

it will display:

==========

List for ''*
/examples/RulesRepositoryDB/rules/CategoryA/RulesA1.xls
/examples/RulesRepositoryDB/rules/CategoryA/RulesA2.xls
/examples/RulesRepositoryDB/rules/CategoryA/SubCategoryA1/RulesA11.xls
/examples/RulesRepositoryDB/rules/CategoryA/SubCategoryA1/RulesA12.xls
/examples/RulesRepositoryDB/rules/CategoryB/RulesB1.xls
/examples/RulesRepositoryDB/rules/CategoryB/RulesB2.xls
/examples/RulesRepositoryDB/rules/Common/libA/libRulesX.xls
/examples/RulesRepositoryDB/rules/Common/libA/libRulesY.xls

==========

5. To inform OpenRulesEngine that now it should look rather to the database than to a local file system, we only have to change slightly the main file rules/main/Main.xls. It used to contain the following Environment table:

Environment	
include.path	../
include	<CategoryA/RulesA1.xls>
	<CategoryA/RulesA2.xls>
	<CategoryB/RulesB1.xls>
	<CategoryB/RulesB2.xls>
	<Common/libA/libRulesX.xls>
	<Common/libA/libRulesY.xls>
import.java	myjava.package1.*
import.static	com.openrules.tools.Methods

6. The modified Environment table will look like here:

Environment	
datasource	**classpath:db.properties**
include.path	**db:/examples/RulesRepositoryDB/rules/**
include	<CategoryA/RulesA1.xls>
	<CategoryA/RulesA2.xls>
	<CategoryB/RulesB1.xls>

	<CategoryB/RulesB2.xls>
	<Common/libA/libRulesX.xls>
	<Common/libA/libRulesY.xls>
import.java	myjava.package1.*
import.static	com.openrules.tools.Methods

7. The db configuration file *rules/main/db.properties* will be exactly the same as above with one exception - a physical location of the database file: *db.url=jdbc:derby:C:/_openrules/openrules.examples/RulesRepositoryDB/db/db storage;create=true*

   Make sure that you change this property to an absolute or relative path that corresponds to your directory structure.

8. Now, we can execute the rules directly from the database by double-clicking on compile.bat and run.bat.

Pure DB-based Rule Repository (no Excel)

In the previous example we still used two configuration files (Main.xls and db.properties) to define a database configuration. However, OpenRules® allows a user to exclude such configuration files and rely only on the database records. It could be important when a user wants to use OpenRules® within its db-centric environment, for example deploying rule services in OJVM (Java Virtual Machine in the Oracle Database) that requires that all of its resources be available on the Java class path.

Example "RulesRepositoryPureDB"

In the above example "RulesRepositoryDB" we used external (not DB-based) configuration files:

- The main xls-file "*file: rules/main/Main.xls*" to contains the Environment table with a reference to the "datasource" like "classpath:db.properties"
- The text file "db.properties" that describes a database connection properties.

In this example "RulesRepositoryPureDB", we demonstrate how to remove these files.

Main Java Launcher

The previous example "RulesRepositoryDB" used the following Java launcher:

```
public static void main(String[] args) {

        String fileName = "file:rules/main/Main.xls";
        OpenRulesEngine engine = new OpenRulesEngine(fileName);

        Appl appl = new Appl();
        Object[] objects = new Object[] { appl };
        engine.run("main",objects);
}
```

Main.xls defined a data source as the "db.properties" file, that included an connection URL as

```
db.url=jdbc:derby:/OR/openrules.examples/RulesRepositoryDB/db/dbstorage
```

The new Java launcher will look like below:

```
public static void main(String[] args) {

        String fileName =
             "db:/examples/RulesRepositoryDB/rules/main/Main.xls";
        Properties properties = OpenRulesEngine.getDbProperties();
        properties.setProperty(DaoOptions.CONNECT_URL,
 "jdbc:derby:/OR/openrules.examples/RulesRepositoryDB/db/dbstorage");
        OpenRulesEngine engine =
            new OpenRulesEngine(properties,fileName);

        Appl appl = new Appl();
        Object[] objects = new Object[] { appl };
        engine.run("main",objects);
}
```

Please note that instead of the protocol "**file:**"

```
String fileName = "file:rules/main/Main.xls";
```

now we use the protocol "**db:**"

```
String fileName =
   "db:/examples/RulesRepositoryDB/rules/main/Main.xls";
```

Defining Database Properties

OpenRules® (starting with the release 6.2.1) provides a Java API for setting database properties. The static method

```
Properties properties = OpenRulesEngine.getDbProperties();
```

allows a user to get the default properties tuned for the Derby database. Here is the list of the default properties:

Property Name	Default Value
USER	embedded
PASSWORD	none
CONNECT_URL	none
DRIVER	org.apache.derby.jdbc.EmbeddedDriver
SELECT_SQL	select content from dbstorage where name = ?
STATEMENT_FACTORY_CLASS	com.openrules.jdbc.StatementFactoryDerbyEmbedded
CREATE_DDL	create table dbstorage (name varchar(255) not null,content blob,primary key(name))
INSERT_SQL	insert into dbstorage(name,content) values(?,?)
LIST_SQL	select name from dbstorage where name like ?
DELETE_SQL	delete from dbstorage where name like ?

All names of properties are static members of the standard public interface DaoOptions. You may reuse (or redefine) these defaults but you always need to specify the property DaoOptions.*CONNECT_URL*as it was done above:

```
properties.setProperty(DaoOptions.CONNECT_URL,
   "jdbc:derby:/OR/openrules.examples/RulesRepositoryDB/db/dbstorage");
```

that point to the actual location of your database using a "`jdbc:`" protocol.

Special OpenRulesEngine Constructor

OpenRules® (starting with the release 6.2.1) provides a special constructor

```
public OpenRulesEngine(Properties dbProperties,String fileName);
```

that creates an instance of the class OpenRulesEngine based on the "dbProperties". The parameter "filename" as usual points to the main xls-files that in this case is expected to be located in a database as a Blob object, e.g.

```
String fileName =
          "db:/examples/RulesRepositoryDB/rules/main/Main.xls";
```

The filename should start with the protocol "db:". The proper "Main.xls" as usual contains an Environment table that describes the rule project structure with references to all other "include"-files and "import"-classes. However, it does not contain a "datasource" that used to point to "db.properties" – they are already defined.

If you want to use a decision project, you may create a rule engine as above, and then create a decision as in the examples below:

```
public static void main(String[] args) {

       String fileName =
            "db:/examples/RulesRepositoryDB/rules/main/Decision.xls";
       Properties properties = OpenRulesEngine.getDbProperties();
       properties.setProperty(DaoOptions.CONNECT_URL,
  "jdbc:derby:/OR/openrules.examples/RulesRepositoryDB/db/dbstorage");
       OpenRulesEngine engine =
           new OpenRulesEngine(properties,fileName);

       Decision decision = new Decision("MyDecision", engine);
       Appl appl = new Appl();
       Decision.put("appl",appl);
       decision.execute();
}
```

Changes in the Main.xls file

Previously the file Main.xls contained this Environment table:

Environment	
datasource	**classpath:db.properties**
include.path	**db:/examples/RulesRepositoryDB/rules/**
include	<CategoryA/RulesA1.xls>
	<CategoryA/RulesA2.xls>
	<CategoryB/RulesB1.xls>
	<CategoryB/RulesB2.xls>
	<Common/libA/libRulesX.xls>
	<Common/libA/libRulesY.xls>
import.java	myjava.package1.*
import.static	com.openrules.tools.Methods

The modified Environment table will look like here:

Environment	
include.path	../
include	<CategoryA/RulesA1.xls>
	<CategoryA/RulesA2.xls>
	<CategoryB/RulesB1.xls>
	<CategoryB/RulesB2.xls>
	<Common/libA/libRulesX.xls>
	<Common/libA/libRulesY.xls>
import.java	myjava.package1.*
import.static	com.openrules.tools.Methods

Please, note that we remove a pointer to not-needed anymore file "db.properties" and only changed "include.path". All additional rule repository tables (such as Method "main" in the Main.xls remain unchanged.

Database Administration Interface

If you keep your rule repository completely in a database (as Blob objects) you will probably rely on your own DB administration interface to upload Excel files in your database. However, you still may use a simple, command-line DB administration interface provided by OpenRules®. In the above example "RulesRepositoryPureDB", we used a sub-directory "db" copied from the example "RulesRepositoryDB". We still needed the file "db.properties" (saved in the

rules/main) to replace a slightly modified Main.xls file to the new database using commands:

>db -rm /examples/RulesRepositoryDB/main/Main.xls

>db -u /examples/RulesRepositoryDB/main/Main.xls -f ../rules/main/Main.xls

The only difference with a previous "db.properties" is that now the property "`db.url`" points to a new physical location of our database:

```
db.url=jdbc:derby:/_SourceRepo/openrules.rules/RulesRepositoryPureDB/db/dbstorage
```

You even do not have to change the path names for the all files saved in the "dbstorage".

DB-based Rules Repository with Version Control

OpenRules® provide a protocol "**dbv:**//<repository:version>/<filepath>" that allows you to use standard databases to keep and maintain your business rules. This protocol is similar to "**db:**<filepath>" but additionally provides version control capabilities. This protocol assumes that you place your Excel files with OpenRules® tables into any relational database as Blob objects (binary large objects). OpenRules® provides a direct access to Excel files saved in a database without necessity to download them into a file system. This protocol supports one table in which all Excel files are saved as Blob objects with unique keys that usually correspond to relative paths of these files in a file system. You may check-in or check-out your Excel files in a way similar to standard version control systems such as Subversion and tell OpenRules® engine which version of rules saved in the database you want to use.

Database Structure

The "***dbv://<repository:version>/<filepath>***" protocol assumes that all Excel files are saved in one database table called "dbstorage" with the following structure:

- revision - an integer, not null, a primary key
- name - a string up to 255 characters, not null
- author - a string up to 64 characters, not null
- comments - a CLOB object (large objects consisting of single-byte fixed-width character data)
- content - a BLOB object, a placeholder for an Excel file
- deleted - a single character, not null, default is 'N'
- timestamp - a timestamp, not null, default is the current timestamp.

You may initialize a repository with your own name, say "rules" and start checking in your Excel files into this repository.

Repository Versioning Concept

The "dbv:" protocol supports a simple check-in/check-out mechanism that in general is similar to the Subversion check-in/check-out. Upon checking in, all files in the database are uniquely identified by an automatically assigned revision number and their names. When you check in a file that is already checked-in the previous copy is not deleted and is still available under its own revision number. The latest revision number becomes a current revision number of the repository. It is always possible to get access to the different states of repository by their revision numbers. For example,

"dbv://rules:25/policy/Driver.xls"

provides an access to the file "/policy/Driver.xls" whose revision number is equal or less than 25. Your OpenRules® Environment table may define the property "include" with the value

"dbv://rules:/policy/Driver.xls"

In this case when a version number is missing, OpenRulesEngine will pick up the latest revision of the file with the name "/policy/Driver.xls".

Example "RulesRepositoryDBV" with rules inside Apache Derby

This example demonstrates how to convert a rules hierarchy from a file system to a database. We will take the directory "rules" from a standard OpenRules® example "RulesRepository" as a basis and will convert it to the new project "RulesRepositoryDBV" with ability to use different rules revisions. This project is included into the complete OpenRules® installation. Here are the conversion steps.

1. Initially the project "RulesRepositoryDBV" has the same structure as the project "RulesRepository". The Java code will not be changed. We will check-in all rules from the folder "rules" into the standard open source Java database known as Apache Derby. Add a new folder "db" as a placeholder for a Derby database.
2. Add to the folder "db" the following bat-file "db.bat"

```
=================================================================
@echo off
cd %~dp0
set db.properties=../rules/main/db.properties
set CONFIG=../../openrules.config/lib
set CLASSPATH=%CONFIG%/openrules.dbv.jar
set CLASSPATH=%CLASSPATH%;%CONFIG%/commons-cli-1.0.jar
set CLASSPATH=%CLASSPATH%;%CONFIG%/derby.jar
java -DDB_PROPERTIES="%db.properties%" -classpath "%CLASSPATH%"
com.openrules.dbv.admin.DB %*
=================================================================
```

This file will be used as a command-line interface to create and administer the database.

3. Add to the folder "rules/main/" the following file "db.properties". It will be used as the database configuration file:

```
==========================================================================
# OpenRules Data Source Protocol Properties
openrules.protocol.name=dbv
openrules.protocol.class=com.openrules.dbv.DbvDataSourceHandler

# DB Access Properties
db.user=embedded
db.password=none
db.url=jdbc:derby:/_openrules/openrules.examples/RulesRepositoryDBV/db/repository;create=true
db.driver=org.apache.derby.jdbc.EmbeddedDriver

db.statementfactory.class=com.openrules.jdbc.StatementFactoryDerbyEmbedded
db.createDDL=\
CREATE TABLE repo (\
REVISION INT NOT NULL,\
NAME VARCHAR(255) NOT NULL,\
AUTHOR VARCHAR(64) NOT NULL,\
COMMENTS CLOB,\
CONTENT BLOB,\
DELETED CHAR(1) NOT NULL DEFAULT 'N',\
TIMESTAMP TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,\
PRIMARY KEY(REVISION)\
)
db.selectSql=select content from dbstorage where name = ?

# DB Administration Properties
db.implementation.class=com.openrules.dbv.admin.impl.RepoDAOImpl

==========================================================================
```

The folder "../rules/main/" is a natural place for the file "db.properties" because this folder already contains the main file Main.xls used by an OpenRulesEngine.

4. The property "openrules.protocolname" defines the name "**dbv**" of this data source protocol. The property "openrules.protocol.class" defines its implementation class provided by OpenRules® using openrules.dbv.jar.

To access the database we have to specify 5 properties:

1) db.user - a user name (here is "embedded" is the default name for Apache Derby.
2) db.password - a user password
3) db.url - defines a physical location of the database. You should make sure that this location corresponds to your file structure (!)
4) db.driver - defines a JDBC driver
5) db.selectDDL - defines how the repository will be created.
6) db.selectSql - shows how the db protocol will access the database.

The remaining three properties define are used by a database command-line administration interface provided by OpenRules® (see below).

5. Switch to a command line interface and position yourself into the newly created directory "db". We will create a new repository called "rules" inside a Derby database. To do that, we execute the following command from a system console:

   >db -r rules -i

   First, a new instance of tne Derby database will be created in accordance with the property "db.util" from the db.properties file. In our case it is defined as:

*db.url=jdbc:derby:/_openrules/openrules.examples/RulesRepositoryDBV/db/**repository**;create=true*

Make sure that first you change this property to an absolute or relative path that corresponds to your directory structure. Thus, a subfolder "repository" will be created inside the folder "db".

6. This project contains many Excel files:

rules/main/Main.xls the main file for a rules engine to start with. It contains only the Environment table
rules/CategoryA/RulesA1.xls
rules/CategoryA/RulesA2.xls
rules/CategoryA1/RulesA1.xls
rules/CategoryA1/SubCategoryA1/RulesA11.xls
rules/CategoryA1/SubCategoryA1/RulesA12.xls
rules/CategoryB/RulesB1.xls
rules/CategoryB/RulesB2.xls
rules/Common/libA/libRulesX.xls
rules/Common/libA/libRulesY.xls

Let's add one more file ***rules/Common/Version.xls*** to display the current repository version using the following method:

Method void displayVersion()
out("This is a DEVELOPMENT revision of the rules repository");

7. The main xls-file Main.xls will continue to be used outside the database serving as a OpenRules® configuration file.
8. To check-in all Excel files (except of Main.xls) to the database we will execute the following commands:

>db -ci /Common/Version.xls -f ../rules/Common/Version.xls -u admin -m "initial check-in"

>db -ci /Common/libA/libRulesX.xls -f ../rules/Common/libA/libRulesX.xls -u admin -m "initial check-in"
>db -ci /Common/libA/libRulesY.xls -f ../rules/Common/libA/libRulesY.xls -u admin -m "initial check-in"
>db -ci /CategoryA/RulesA1.xls -f ../rules/CategoryA/RulesA1.xls -u admin -m "initial check-in"

>db -ci /CategoryA/RulesA2.xls -f ../rules/CategoryA/RulesA2.xls -u admin -m "initial check-in"
>db -ci /CategoryA/SubCategoryA1/RulesA11.xls -f ../rules/CategoryA/SubCategoryA1/RulesA11.xls -u admin -m "initial check-in"
>db -ci /CategoryA/SubCategoryA1/RulesA12.xls -f ../rules/CategoryA/SubCategoryA1/RulesA12.xls -u admin -m "initial check-in"
>db -ci /CategoryB/RulesB1.xls -f ../rules/CategoryB/RulesB1.xls -u admin -m "initial check-in"
>db -ci /CategoryB/RulesB2.xls -f ../rules/CategoryB/RulesB2.xls -u admin -m "initial check-in"

After every check-in the system will display the latest revision number like here:

>db -ci /CategoryB/RulesB2.xls -f ../rules/CategoryB/RulesB2.xls -u admin -m "initial check-in"

File ../rules/CategoryB/RulesB2.xls has been checked in
rules:9:/CategoryB/Rules
B2.xls
Repository at revision 9.

If now we enter the command:

>db -l ""*

it will display something like this:

==========

>db -l ""*
/Common/Version.xls r.1 2007-05-18 06:41:40
/Common/libA/libRulesX.xls r.2 2007-05-18 06:02:41

/Common/libA/libRulesY.xls r.3 2007-05-18 06:17:41

/CategoryA/RulesA1.xls r.4 2007-05-18 06:31:41

/CategoryA/RulesA2.xls r.5 2007-05-18 06:46:41

/CategoryA/SubCategoryA1/RulesA11.xls r.6 2007-05-18 06:01:42

/CategoryA/SubCategoryA1/RulesA12.xls r.7 2007-05-18 06:30:42

/CategoryB/RulesB1.xls r.8 2007-05-18 06:48:42

/CategoryB/RulesB2.xls r.9 2007-05-18 06:04:43

Repository at revision 9.

==========

9. To inform OpenRulesEngine that now it should look rather to the database than to a local file system, we only have to change slightly the main file rules/main/Main.xls. It used to contain the following Environment table:

Environment	
include.path	../
include	<CategoryA/RulesA1.xls>
	<CategoryA/RulesA2.xls>
	<CategoryB/RulesB1.xls>
	<CategoryB/RulesB2.xls>
	<Common/libA/libRulesX.xls>
	<Common/libA/libRulesY.xls>
import.java	myjava.package1.*
import.static	com.openrules.tools.Methods

10. The modified Environment table will look like here:

Environment	
datasource	**classpath:db.properties**
include.path	**dbv://rules/**
include	<CategoryA/RulesA1.xls>
	<CategoryA/RulesA2.xls>

	<CategoryB/RulesB1.xls>
	<CategoryB/RulesB2.xls>
	<Common/Version.xls>
	<Common/libA/libRulesX.xls>
	<Common/libA/libRulesY.xls>
import.java	myjava.package1.*
import.static	com.openrules.tools.Methods

11. Now, we can execute the rules directly from the database by double-clicking on compile.bat and run.bat. The results will be displayed as:

[java] ==========
[java] OpenRulesEngine: file:rules/main/Main.xls
[java] ==========
[java] This is a DEVELOPMENT revision of the rules repository
[java] Execute RulesA1
[java] Execute RulesA11
[java] Execute LibRulesX
[java] Sat May 19 09:57:38 BST 2007
[java] Execute RulesA2
[java] ==========

Maintaining Multiple Repository Versions

You may use revisions to maintain different version of your rules repositories. Let's consider the following scenario using a rules repository we created above as an example. The current state of this repository will always represent the latest rules development version. At certain point of development we may decide to label the current state of repository as a release with some arbitrary name. For example, let's label the current state of our rules repository as "Alpha Release". Remember the last revision number was 9. For the testing purposes let's modify the file /Common/Version.xls to display the words "Alpha Release" instead of development. We can check-out this file and make the following changes in it:

Method void displayVersion()
out("This is a ALPHA RELEASE of the rules repository - revision 10");

You may wonder why we use revision 10 instead of 9. Because when we check-in back the modified Version.xls it will make the repository revision equal to 9+1=10. Here are the results of this check-in:

>db -ci /Common/Version.xls -f ../rules/Common/Version.xls -u admin -m "ALPHA RELEASE"
File ../rules/Common/Version.xls has been checked in
rules:10:/Common/Version.xls
Repository at revision 10.

To run OpenRulesEngine against this particular Aplha Release (revision 10) of our rules repository, we have to modify the OpenRules® configuration file Main.xls as follows:

Environment	
datasource	classpath:db.properties
include.path	**dbv://rules:10/**
include	<CategoryA/RulesA1.xls>
	<CategoryA/RulesA2.xls>
	<CategoryB/RulesB1.xls>
	<CategoryB/RulesB2.xls>
	<Common/Version.xls>
	<Common/libA/libRulesX.xls>
	<Common/libA/libRulesY.xls>
import.java	myjava.package1.*
import.static	com.openrules.tools.Methods

Revision 10 is ALPHA RELEASE

Please pay attention that we only changed "dbv://rules/" to "dbv://rules:10/". We also added an optional comment under the Environment table.

If you double-click to the run.bat now, it will produce the following results:

[java] ==========
[java] OpenRulesEngine: file:rules/main/Main.xls
[java] ==========
[java] This is a APLHA RELEASE of the rules repository - revision 10
[java] Execute RulesA1
[java] Execute RulesA11
[java] Execute LibRulesX
[java] Sat May 19 09:57:38 BST 2007
[java] Execute RulesA2
[java] ==========

How to run OpenRulesEngine against different versions of the rules repository at the same time? It is a matter of where you want to place different versions of your Main.xls file. For example, let's place a new Main.xls (for the Alpha Release) in the folder "rules/main.alpha" while the folder "rules/main" will continue to keep our old Main.xls (without any revision number). Directing the OpenRulesEngine to one Main.xls or another will execute different versions. For example, let's change the main Java module of this particular project Appl.java to make it execute OpenRulesEngine first for the DEVELOPMENT and then for the ALPHA releases. Here is a modified code of Appl.java:

```
public static void main(String[] args) {
   runRuleEngine("file:rules/main/Main.xls");
   runRuleEngine("file:rules/main.alpha/Main.xls");
}

public static void runRuleEngine(String mainXlsFile) {
```

```
    OpenRulesEngine engine = new OpenRulesEngine(mainXlsFile);
    System.out.println(
    "\n===========================================\n" +
    "OpenRulesEngine: " + mainXlsFile +
    "\n===========================================\n");
    Appl appl = new Appl();
    Object[] objects = new Object[] { appl };
    String methodName = "main";
    engine.run(methodName,objects);
    System.out.println(
    "\n===========================================\n");
}
```

If you double-click to the run.bat now, it will produce the following results:

[java] ==
[java] OpenRulesEngine: ***file:rules/main/Main.xls***
[java] ==
[java] This is a DEVELOPMENT revision of the rules repository
[java] Execute RulesA1
[java] Execute RulesA11
[java] Execute LibRulesX
[java] Sat May 19 09:57:38 BST 2007
[java] Execute RulesA2
[java] ==

.....

[java] ==
[java] OpenRulesEngine: ***file:rules/main.alpha/Main.xls***
[java] ==
[java] This is a APLHA RELEASE of the rules repository - revision 10
[java] Execute RulesA1
[java] Execute RulesA11
[java] Execute LibRulesX

[java] Sat May 19 09:57:38 BST 2007
[java] Execute RulesA2
[java] ==========

Database Administration Interface

OpenRules® provide a command-line interface for the DB administration using the "dbv:" protocol. You may see concrete examples above. Here is the list of the available commands and options:

Option "**Initialize**": -i or --init
Usage: db -i or db -r <repo> -i
This option initializes a repository <repo>. By default the bat-file db.bat is using "-r rules", so the option -r <repo> may be omitted. Warning: Existing data will be destroyed
Example: db -i

Option "**Check-In**": -ci or --checkin
Usage: db -ci <repo-path> -f <local-file-path> [-u <user>] [-m <message>]
This option checks in (uploads) the content of <local-file-path> into the database using the <repo-path> as its name. The file will be given a new revision name that is equal to (the current repository revision + 1). The user name and message (if any) will be attached to this revision. If <user> is not specified, the option will use the name specified by the property "db.user" inside the file "db.properties".
Example: *db -ci /Common/libA/libRulesX.xls -f ../rules/Common/libA/libRulesX.xls -u admin -m "initial check-in"*

Option "**Check-Out**": -co or --checkout
Usage: db -co <repo-path> -f <local-file-path> [-rev <revision-number>]
This option checks out (downloads) data from the database repository "rules" using this <repo-path> and <revision-number> into the <localfile-path>. If revision is omitted the latest revision will be used.

Example: *db -co /Common/libA/libRulesX.xls -f ../rules/Common/libA/libRulesX.xls -rev 10*
This command will download file DriverDiscountRules.xls into the directory c:/temp.

Option "**Remove**": -rm or --remove
Usage: db -rm "<mask>"
This option removes data from the repository using this <mask>. The mask can include
wildcards '*' and '?'
Example: *db -rm "*"* will remove all files from the rules repository
Option "**List**": -l or --list
Usage: db -l <mask> [-rev <number>]
This option lists items checked-in to the rules repository. The mask is used to specify different items in the repository. The mask can include wildcards '*' and '?'.
Example: *db -l "*"* -rev 9 will display all checked-in files with a revision number 9 or under.

Option "**History**": -y or --history
Usage: db -y <repo-path>
This option display detailed information about all revisions for the selected <repo-path>.
Example: *db -y /Common/Version.xls* will display all revisions of the file /Common/Version.xls
Option "**Revision**": -rev or --revision
Usage: -rev <number>
This option is used with option -co.

Option "**File**": -f or --file
Usage: -f <local-file-path>
This option is used with options -ci and -co.

Option "**Help**": -h or --help

Usage: db -h

This option displays a list of all options.

Custom DB Protocols

To take into consideration specifics of their databases, advanced users may create custom DB protocols with their own names. To customize the standard OpenRules® protocols a user can make modifications in a configuration file "db.properties" are required. The users also may create and register their own db-protocol implementations using the OpenRules® source code as an example.

Using External Rules to Access Custom Rule Repositories

If you do not want to use your own organization of rules in any database (not the one supported by standard protocols such as “db:”) you may utilize the concept of External Rules. The detailed description of external rules coming from a custom database is provided at http://openrules.com/external_rules_from_db.htm. It comes with a detailed rule project “ExternalRulesFromDB”.

Supporting Jars and Projects

If you use access a database using DatabaseIterator you need to add only one jar-file “com.openrules.tools.jar” to your classpath. It is included in the standard configuration project “openrules.config/lib”. The standard projects that demonstrate the use of database iterators are:

- HelloCustomerDB
- DecisionPatientTherapyDB.

If you use a database protocol like “db:” or “dbv:” you need the following jar-files to be added to your classpath:

- `openrules.db.jar`
- `openrules.dbv.jar`

- `derby.jar`
- `commons-cli-1.1.jar.`

They are included in the standard configuration project “openrules.config/lib”.

The following standard projects may serve as prototypes for your own OpenRules® projects with database interfaces:

- HelloJavaDB
- RulesRepository
- RulesRepositoryDB
- RulesRepositoryPureDB
- RulesRepositoryDBV.

They can be found in the workspace “openrules.rules” available as a part of the complete product release.

TECHNICAL SUPPORT

Direct all your technical questions to support@openrules.com or to this Discussion Group.