OPENRULES®

Open-Source Business Rules and Decision Management System

Release 8.4.*

User Manual

Part1

(for Business Analysts)

OpenRules, Inc.

www.openrules.com

June-2021

Part 2

Table of Contents

INTRODUCTION

OpenRules® was developed in 2003 by OpenRules, Inc. as an open source Business Rules Management System (BRMS) and since then has become one of the most popular Business Rules products on the market. Over these years OpenRules® has been naturally transformed in a Business Rules and Decision Management System with proven records of delivering and maintaining reliable decision support software. OpenRules® is a winner of several software awards for innovation and is used worldwide by multi-billion corporations, major banks, insurers, health care providers, government agencies, online stores, universities, and many other institutions.

OpenRules® Components

OpenRules® offers the following decision management components:

- OpenRules Repository for management of enterprise-level decision rules
- OpenRules Engine for execution of decisions and different business rules
- OpenRules Dialog for building rules-based Web questionnaires
- OpenRules Analyzer for online decision model analysis and execution
- Rule Solver for solving constraint satisfaction and optimization problems.

OpenRules® also uses the following components in its consulting practice:

- OpenRules Learner for rules discovery and predictive analytics
- Finite State Machines for event processing and "connecting the dots"
- Additional components are described here.

Integration of these components with executable decisions provides OpenRules® customer with a general-purpose Business Rules and Decision Management System, oriented to "decision-centric" application development with externalized business logic. The newest OpenRules Decision Manager is available here.

OpenRules, Inc. is a professional open source company that provides software, product documentation and technical support and other services that are highly praised by our customers.

OpenRules® Manuals

You may start learning about product with the document "Getting Started" which describes how to install and use OpenRules® using a simple decision model. Then you may look at a more complex example in the tutorial "Calculating Tax Return". The User Manual consists of two parts:

- User Manual for Business Analysts (this document) covers the core OpenRules® concepts in greater depth but it is oriented to business analysts and covers only major decision modeling concepts.
- User Manual for Developers covers more advanced decision management topics including and integration with IT.

Many other helpful documents and tutorials can be found online at the Documentation page of www.openrules.com.

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.`

Basic Decision Model

OpenRules® is a general-purpose Business Rules Decision Management Systems that allows customers to develop their own operational business decision models. In this section we will introduce OpenRules® decision modeling approach and supporting decisioning concepts using a simple example.

Problem Definition

We will create a decision model "Vacation Days" that should eb able to calculate vacation days for any employee in accordance with the following rules:

> The number of vacation days depends on age and years of service.
>
> Every employee receives at least 22 days.
> Additional days are provided according to the following criteria:
>
> 1) Only employees younger than 18 or at least 60 years, or employees with at least 30 years of service will receive 5 extra days.
>
> 2) Employees with at least 30 years of service and also employees of age 60 or more, receive 3 extra days, on top of possible additional days already given.
>
> 3) 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. These 2 extra days can not be combined with the 5 extra days.

Goals and Subgoals

What is the main goal (objective) of this decision model? The goal is to define an employee's vacation days based of age and years of service. To do this we would also need to define an employee's eligibility to extra 5, 3, and 2 days (subgoals).

Glossary

We will start with creation of the business glossary that is a special table that contains all goals, subgoals, and other variables participating in making decisions. We may put our goal "Vacation Days" and 3 subgoals "Eligible for

Extra 5 Days", "Eligible for Extra 3 Days", "Eligible for Extra 2 Days" in our initial glossary that will look as below:

Glossary glossary	
Variable Name	**Business Concept**
Vacation Days	Employee
Eligible for Extra 5 Days	
Eligible for Extra 3 Days	
Eligible for Extra 2 Days	

Defining Business Logic in Decision Tables

Now we may define the business logic of our top-level goal "Vacation Days" using the following decision table:

DecisionTableMultiHit CalculateVacationDays				
If	If	If	Conclusion	
Eligible for Extra 5 Days	**Eligible for Extra 3 Days**	**Eligible for Extra 2 Days**	**Vacation Days**	
			=	22
TRUE			+=	5
	TRUE		+=	3
FALSE		TRUE	+=	2

The first ("black") row specifies the name of the table as "CalculateVacationDays" (spaces not allowed in the table's name) and its type "DecisionTableMultiHit". This decision table has 3 conditions (the keyword "If" in the second row) and one conclusion. The first rule will unconditionally assign 22 days to the variable "Vacation Days". Then the second rule will add (the operator "+=") 5 more days only if the first condition "Eligible for Extra 5 Days" is TRUE. Then the third rule will add 3 more days only if the second condition "Eligible for Extra 3 Days" is TRUE. And finally, the fourth rule will add 2 more days only if the condition "Eligible for Extra 2 Days" is TRUE and the first condition "Eligible for Extra 5 Days" is FALSE. This table considers all rules because this is a multi-hit decision table (opposite to the default single-hit decision table that stops its execution after at least one rule is satisfied).

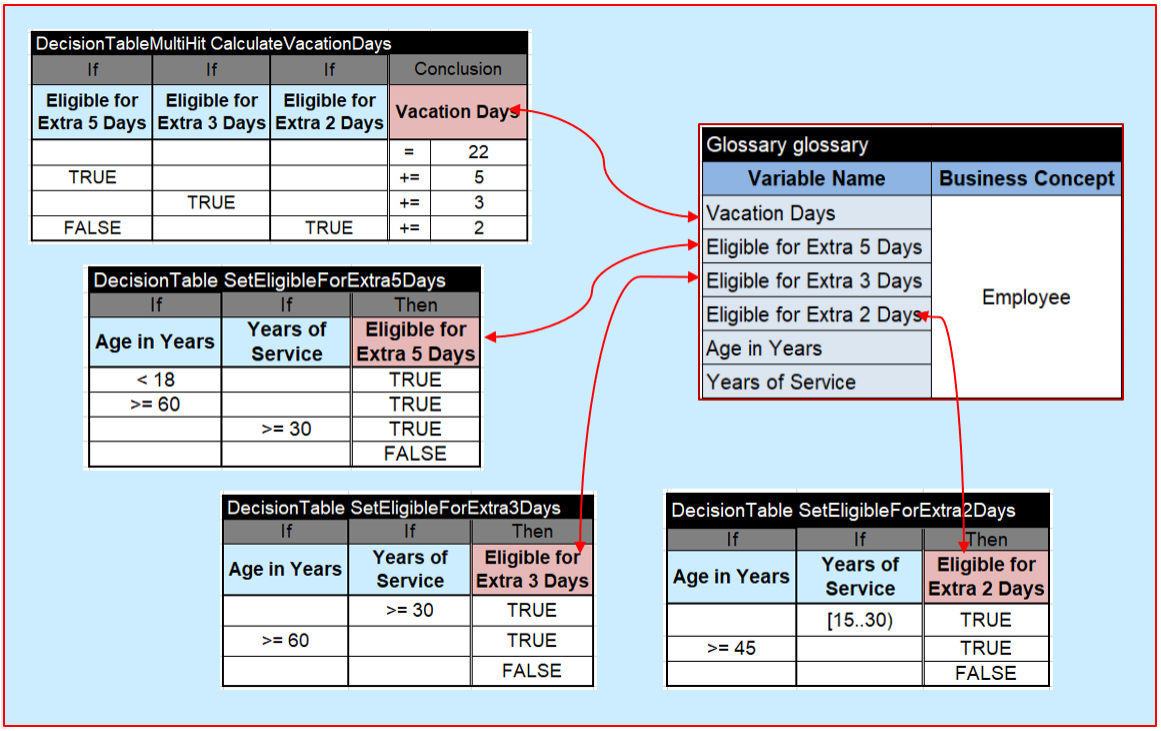
The following (single-hit) decision tables specify business logic for the subgoal for extra days eligibility:

DecisionTable SetEligibleForExtra5Days		
If	If	Then
Age in Years	**Years of Service**	**Eligible for Extra 5 Days**
< 18		TRUE
>= 60		TRUE
	>= 30	TRUE
		FALSE

DecisionTable SetEligibleForExtra3Days		
If	If	Then
Age in Years	**Years of Service**	**Eligible for Extra 3 Days**
	>= 30	TRUE
>= 60		TRUE
		FALSE

DecisionTable SetEligibleForExtra2Days		
If	If	Then
Age in Years	**Years of Service**	**Eligible for Extra 2 Days**
	[15..30)	TRUE
>= 45		TRUE
		FALSE

Thus, our decision table is defined by the glossary and 4 decision tables:



Testing Decision Model

To test this decision model, we will create test cases in Excel. First, we need to add the third column with technical names to the glossary:

Glossary glossary		
Variable Name	**Business Concept**	**Attribute**
Vacation Days	Employee	vacationDays
Eligible for Extra 5 Days		eligibleForExtra5Days
Eligible for Extra 3 Days		eligibleForExtra3Days
Eligible for Extra 2 Days		eligibleForExtra2Days
Age in Years		age
Years of Service		service

These names will be by test cases and real data passing to our model. Then we will define the Datatype “Employee”:

Datatype Employee	
String	id
int	age
int	service
boolean	eligibleForExtra5Days
boolean	eligibleForExtra3Days
boolean	eligibleForExtra2Days
int	vacationDays

Then we create a Data table with several test-employees:

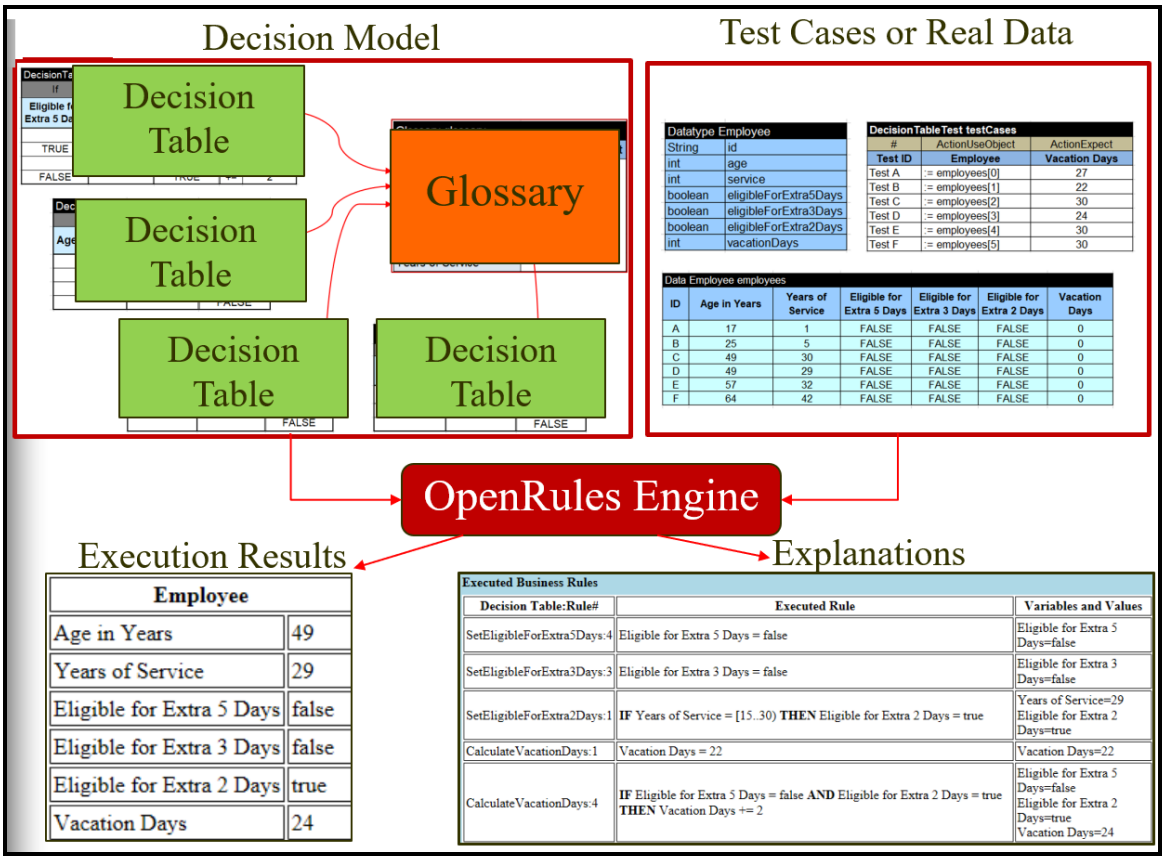
Data Employee employees						
ID	**Age in Years**	**Years of Service**	**Eligible for Extra 5 Days**	**Eligible for Extra 3 Days**	**Eligible for Extra 2 Days**	**Vacation Days**
A	17	1	FALSE	FALSE	FALSE	0
B	25	5	FALSE	FALSE	FALSE	0
C	49	30	FALSE	FALSE	FALSE	0
D	49	29	FALSE	FALSE	FALSE	0
E	57	32	FALSE	FALSE	FALSE	0
F	64	42	FALSE	FALSE	FALSE	0

And finally, we will define test cases with expected results:

DecisionTableTest testCases		
#	ActionUseObject	ActionExpect
Test ID	**Employee**	**Vacation Days**
Test A	:= employees[0]	27
Test B	:= employees[1]	22
Test C	:= employees[2]	30
Test D	:= employees[3]	24
Test E	:= employees[4]	30
Test F	:= employees[5]	30

Executing Decision Model

We may execute the decision model against test cases or against real data using OpenRules Engine as shown at the following schema:



Datatype Employee	
String	id
int	age
int	service
boolean	eligibleForExtra5Days
boolean	eligibleForExtra3Days
boolean	eligibleForExtra2Days
int	vacationDays

DecisionTable Test testCases		
#	ActionUseObject	ActionExpect
Test ID	Employee	Vacation Days
Test A	:= employees[0]	27
Test B	:= employees[1]	22
Test C	:= employees[2]	30
Test D	:= employees[3]	24
Test E	:= employees[4]	30
Test F	:= employees[5]	30

Data Employee employees

ID	Age in Years	Years of Service	Eligible for Extra 5 Days	Eligible for Extra 3 Days	Eligible for Extra 2 Days	Vacation Days
A	17	1	FALSE	FALSE	FALSE	0
B	25	5	FALSE	FALSE	FALSE	0
C	49	30	FALSE	FALSE	FALSE	0
D	49	29	FALSE	FALSE	FALSE	0
E	57	32	FALSE	FALSE	FALSE	0
F	64	42	FALSE	FALSE	FALSE	0

Employee	
Age in Years	49
Years of Service	29
Eligible for Extra 5 Days	false
Eligible for Extra 3 Days	false
Eligible for Extra 2 Days	true
Vacation Days	24

Executed Business Rules

Decision Table:Rule#	Executed Rule	Variables and Values
SetEligibleForExtra5Days:4	Eligible for Extra 5 Days = false	Eligible for Extra 5 Days=false
SetEligibleForExtra3Days:3	Eligible for Extra 3 Days = false	Eligible for Extra 3 Days=false
SetEligibleForExtra2Days:1	**IF** Years of Service = [15..30) **THEN** Eligible for Extra 2 Days = true	Years of Service=29 Eligible for Extra 2 Days=true
CalculateVacationDays:1	Vacation Days = 22	Vacation Days=22
CalculateVacationDays:4	**IF** Eligible for Extra 5 Days = false **AND** Eligible for Extra 2 Days = true **THEN** Vacation Days += 2	Eligible for Extra 5 Days=false Eligible for Extra 2 Days=true Vacation Days=24

OpenRules Engine does the following:

- Finds the execution path for the different goals
- Executes the decision model showing the execution results
- Explains why certain decision were made producing the explanations in the html-reports and saving them in Java objects.

Core Concepts

OpenRules® utilizes the well-established spreadsheet concepts of workbooks, worksheets, and tables to build enterprise-level rule repositories. Each OpenRules® workbook is comprised of one or more worksheets that can be used to separate information by types or categories.

OpenRules® supports different types of table that are defined by their keywords. Here is the list of OpenRules® tables along with brief description of each:

Table Type (Keyword)	Comment
Glossary	The central point of any decision model. The Glossary contains all decision variable (input and output) distributed between different business concepts. It also contains related implementation attributes and (optionally) their possible domains
Decision Table or **DT** or **DecisionTableSingleHit** or **RuleFamily**	This is a single-hit decision table that uses multiple conditions defined on different variables to reach conclusions about the decision variables. It stops working after the first rule is satisfied
DecisionTableMultiHit or **DecisionTable1** or **DT1**	A multi-hit decision table that executes multiple rules in the top-down order and allows rule overrides
DecisionTable2 or **DT2** or **DecisionTableSequence**	A multi-hit decision table that like DecisionTable2 executes all rules in top-down order but results of the execution of previous rules may affect the conditions of rules that follow
Datatype	Defines a new data type directly in Excel that can be used for testing
Data	Creates an array of test objects
Variable	Creates one test object
DecisionTableTest	Defines test cases with expected results
Environment	This table defines the structure of a rules repository by listing all included workbooks, XML files, and Java packages
Decision	Defines a decision that may consist of multiple sub-decisions associated with different decision tables. Can be automatically generated starting since release 7.0.0
DecisionObject	Associates business concepts specified in the glossary with concrete objects defined outside the decision (i.e. as Java objects or Excel Data tables)
DecisionTableIterate	A special decision table that iterates over a list (or array) of business objects applying other decision table to each element of the list
DecisionTableSort	A special decision table that sorts a list (or array) of business objects by applying sorting rules to each element of the list

DecisionTableAssign	A compact decision table that assigns values to variables using only two columns
Method	Defines expressions using snippets of Java code and known decision variables and objects
Rules	Defines a decision table that includes Java snippets that specify custom logic for conditions and actions. Read more. Some Rules tables may refer to templates that hide those Java snippets.
Layout	A special table type used by OpenRules® Forms and OpenRules® Dialog

Spreadsheet Organization and Management

OpenRules® uses Excel spreadsheets to represent and maintain business rules, web forms, and other information that can be organized using a tabular format. Excel is the best tool to handle different tables and is a popular and widely used tool among business analysts.

To create and edit rules and other tables presented in Excel-files you can use any standard spreadsheet editor such as:

- MS Excel®
- Google Sheets®
- OpenOffice®

Google Sheets™ is especially useful for collaborative rules management.

Workbooks, Worksheets, and Tables

OpenRules® utilizes commonly used concepts of workbooks and worksheets maintained in multiple Excel files. Each OpenRules® workbook is comprised of one or more worksheets that can be used to separate information by categories. Each worksheet, in turn, is comprised of one or more tables. Decision tables are the most typical OpenRules® tables and are used to represent business rules. Workbooks can include tables of different types, each of which can support a different underlying logic.

How OpenRules® Tables Are Recognized

OpenRules® recognizes the tables inside Excel files using the following parsing algorithm.

1. The OpenRules® parser splits spreadsheets into “parsed tables”. Each logical table should be separated by at least one empty row or column at the start of the table. Table parsing is performed from left to right and from top to bottom. The first non-empty cell (i.e. cell with some text in it) that does not belong to a previously parsed table becomes the top-left corner of a new parsed table.
2. The parser determines the width/height of the table using non-empty cells as its clues. Merged cells are important and are considered as one cell. If the top-left cell of a table starts with a predefined keyword (see the table below), then such a table is parsed into an OpenRules® table.
3. All other "tables," i.e. those that do not begin with a keyword are ignored and may contain any information.

The list of all keywords was described above. OpenRules® can be extended with more table types, each with their own keyword.

While not reflected in the table recognition algorithm, it is good practice to use a black background with a white foreground for the very first row. All cells in this row should be ***merged***, so that the first row explicitly specifies the table width. We call this row the "***table signature***". The text inside this row (consisting of one or more merged cells) is the table signature that starts with a keyword. The information after the keyword usually contains a unique table name and additional information that depends on the table type.

If you want to put a table title before the signature row, use an empty row between the title and the first row of the actual table. Do not forget to put an empty row after the last table row. Here are examples of some typical tables recognized by OpenRules®.

OpenRules® table with 3 columns and 2 rows:

Keyword <some text>		
Something	Something	Something
Something	Something	Something

OpenRules® table with 3 columns and still 2 rows:

Keyword	Something	Something
Something	Something	Something
Something	Something	Something

OpenRules® table with 3 columns and 3 rows (empty initial cells are acceptable):

Keyword <some text>		
Something	Something	
	Something	Something
		Something

OpenRules® table with 3 columns and 2 rows (the empty 3rd row ends the table):

Keyword <some text>		
Something	Something	Something
Something	Something	Something
Something	Something	Something

OpenRules® table with 2 columns and 2 rows (the empty cell in the 3rd column of the title row results in the 4th columns being ignored. This also points out the importance of merging cells in the title row):

Keyword	Something		Something
Something	Something	Something	Something
Something	Something	Something	Something

OpenRules® will not recognize this table (there is no empty row before the signature row):

Table Title		
Keyword <some text>		
Something	Something	
	Something	Something
		Something

Fonts and coloring schema are a matter of the table designer's taste. The designer has at his/her disposal the entire presentation power of Excel (including comments) to make the OpenRules® tables more self-explanatory. We recommend you follow the coloring and placement recommendations provided by the OMG DMN standard.

Implementation Restriction. You should not merge rows at the very first column of any decision table. To avoid this restriction, you may add the first column of the type "C#" or "#"and use numbers to mark each table's row.

DECISION MODELING AND EXECUTION

OpenRules® uses a goal-oriented approach allows business analysts to develop their executable decision models with glossaries and related decision tables without (or only with a limited) help from software developers. The above decision model "Vacation Days" shows examples of the Glossary and multi-hit and single-hit decision tables. You may find more examples in the document "Getting Started" and several associated tutorials. You may also watch these videos:

- Building and Executing Decision Models with OpenRules
- Goal-Oriented Business Decision Modeling with OpenRules

Starting with Defining Decision Goal and Subgoals

From the OpenRules® perspective a decision model consists of:

- Decision variables that can take specific values from domains of values
- Decision rules (frequently expressed as decision tables) that specify relationships between decision variables.

Some decision variables are known (decision input) and some of them are unknown (decision output) that may represent goals/subgoals. To execute a decision model means to assign values to unknown decision variables in such a way that satisfies all related decision rules.

OpenRules® applies a top-down approach to decision modeling. This means that you usually start with the definition of a Decision Goal and not with rules or data. You put the top-level goal into a glossary and define its business logic using a decision table that specifies subgoals, this goal depends on. You continue this process for all subgoals until all subgoals are defined. For example, OpenRules installation comes with the decision model "PatientTherapy" in the workspace "openrules.models". Its top-level goal "Patient Therapy" depends on 3 subgoals: "Recommended Medication", "Recommended Dose", and "Drug Interaction Warning". The goal "Recommended Dose" depends on "Patient Creatinine Clearance". There is the glossary for this decision model:

Glossary glossary		
Decision Variable	**Business Concept**	**Attribute**
Encounter Diagnosis	DoctorVisit	encounterDiagnosis
Recommended Medication		recommendedMedication
Recommended Dose		recommendedDose
Drug Interaction Warning		warning
Patient Therapy		patientTherapy
Patient Age	Patient	age
Patient Weight		weight
Patient Allergies		allergies
Patient Creatinine Level		creatinineLevel
Patient Creatinine Clearance		creatinineClearance
Patient Active Medication		activeMedication

All goals/subgoals are highlighted as they use the hyperlinks to the proper decision tables. All other variables represent input data. When OpenRules analyses this decision model it automatically builds an execution path, which can be saved (using build.bat) in the file "Goal.xls" as the following table of the type "Decision":

Decision DecisionPatientTherapy
ActionExecute
Decision Tables
DefineMedication
CalculateCreatinineClearance
DefineDosing
WarnAboutDrugInteraction
DeterminePatientTherapy

When OpenRules executes the goal "Patient Therapy" it will actually execute the table "DecisionPatientTherapy" (or its internal representation). Of course, you may create this table manually, and/or even expand it making its "smarter". For example, before the release 7.0.0 OpenRules customers needed to create the tables of the type "Decision" manually for every goal and subgoal. Here is an example of such manually created table:

Decision DecisionPatientTherapy	
Decisions	**Execute Decision Tables**
Define Medication	DefineMedication
Define Creatinine Clearance	CalculateCreatinineClearance
Define Dosing	DefineDosing
Check Drug Interaction	WarnAboutDrugInteraction
Determine Patient Therapy	DeterminePatientTherapy

The table "Decision" by default has two columns "Decisions" and "Execute Decision Tables". The first column contains the display names of all sub-decisions. The second column contains exact names of decision tables that implement these sub-decisions. The decision table names cannot contain spaces or special characters (except for "underscore").

Some decisions may have a more complex structure. When you create them manually, you can even use conditions inside decision tables. For example, consider a situation when the first sub-decision validates your data and a second sub-decision executes complex calculations but only if the preceding validation was successful. Here is an example of such a decision from the tax calculation tutorial:

Decision Apply1040EZ			
Condition		ActionPrint	ActionExecute
1040EZ Eligible		**Decisions**	**Execute**
		Validate	ValidateTaxReturn
Is	TRUE	Calculate	DetermineTaxReturn
Is	FALSE	Do Not Calculate	

Since this table "Decision Apply1040EZ" uses an optional column "Condition", we must add a second row. The keywords "Condition", "ActionPrint", and "ActionExecute" are defined in the standard OpenRules® template "DecisionTemplate" – see the configuration file "DecisionTemplates.xls" in the folder "openrules.config". This table uses a decision variable "1040EZ Eligible" that is defined by the first (unconditional) sub-decision "Validate". We assume that the decision "ValidateTaxReturn" should set this decision variable to TRUE or FALSE. Then the second sub-decision "Calculate" will be executed only when "1040EZ Eligible" is TRUE. When it is FALSE, this decision, "Apply1040EZ", will simply print "Do Not Calculate". In our example the reason will be printed by the decision table "ValidateTaxReturn".

Note. You may use many conditions of the type "Condition" defined on different decision variables. Similarly, you may use an optional condition "ConditionAny" which instead of decision variables can use any formulas defined on any known objects. It is also possible to add custom actions using an optional action "ActionAny" – see "DecisionTemplates.xls" in the folder "openrules.config".

Defining Decision Tables

OpenRules® decision modeling approach utilizes the classical decision tables that were in the heart of OpenRules® from its introduction in 2003. OpenRules® uses the keyword "**DecisionTable**" (or "**DT**") for the most frequently used single-hit decision tables. For example, let's consider a very simple decision table:

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

Its first row contains a keyword "DecisionTable" and a unique name (no spaces allowed). The second row uses keywords "Condition" and "Conclusion" to specify

the types of the decision table columns. The third row contains decision variables expressed in plain English (spaces are allowed but the variable names should be unique).

The columns of a decision table define conditions and conclusions using different operators and operands appropriate to the decision variable specified in the column headings. The rows of a decision table specify multiple rules. For instance, in the above decision table "DefineGreeting" the second rule can be read as:

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

Similarly, we may define the second decision table "DefineSalutation" that determines a salutation word (it uses the keyword "DT" that is a synonym for "DecisionTable"):

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

If some cells in the rule conditions are empty, it is assumed that this condition is satisfied. A decision table may have no conditions, but it always should contain at least one conclusion.

Decision Table Execution Logic

By default, OpenRules® executes all rules within DecisionTable in a *top-down* order. When all conditions inside one rule (row) are satisfied the proper conclusion(s) from the same row will be executed, and all other rules will be ignored.

AND/OR Conditions

The conditions in a decision table are always connected by a logical operator "AND". When you need to use "OR", you may add another rule (row) that is an alternative to the previous rule(s). However, some conditions may have a decision variable defined as an array, and within such array-conditions "ORs" are allowed. Consider for example the following, more complex decision table:

RuleFamily 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
						Are	None	Sorry

Here the decision variables "Customer Profile", "Customer Product", and "Offered Products" are arrays of strings. In this case, the second rule can be read as:

```
IF Customer Profile Is One Of New or Bronze or Silver
AND Customer Products Include Checking Account and
Overdraft Protection
AND Customer Products Do Not Include CD with 25 basis point
increase, Money Market Mutual Fund, and Credit Card
THEN Offered Products ARE CD with 25 basis point increase,
Money Market Mutual Fund, and Credit Card
```

Decision Table Operators

OpenRules® supports multiple ways to define operators within decision table conditions and conclusions. When you use a text form of operators you can freely use upper and lower cases and spaces. The following operators can be used inside decision table conditions:

Operator	Synonyms	Comment
Is	=, ==	When you use "=" or "==" inside Excel write"'=" or"'==" with apostrophe to avoid confusion with Excel's own formulas
Is Not	!=, isnot, Is Not Equal To, Not, Not Equal., Not Equal To	Defines an inequality operator
>	Is More, More, Is More Than, Is Greater, Greater, Is Greater Than	For integers and real numbers, and Dates
>=	Is More Or Equal. Is More Or Equal To, Is More Than Or Equal To, Is Greater Or Equal To, Is Greater Than Or Equal To	For integers and real numbers, and Dates
<=	Is Less Or Equal, Is Less Than Or Equal To, Is Less Than Or Equal To, Is Smaller Or Equal To, Is Smaller Than Or Equal To, Is Smaller Than Or Equal To,	For integers and real numbers, and Dates
<	Is Less, Less, Is Less Than, Is Smaller, Smaller, Is Smaller Than	For integers and real numbers, and Dates
Is True		For booleans
Is False		For booleans
Is Empty		A string is considered "empty" if it is either "null" or contains only zero or more whitespaces
Contains	Contain	For strings only, e.g. "House" contains "use". The comparison is not case-sensitive
Does Not Contain	DoesNotContain	For strings only, e.g. "House" does not contain "user". The comparison is not case-sensitive
Starts With	Start with, Start	For strings only, e.g. "House" starts with "ho". The comparison is not case-sensitive
Match	Matches, Is Like, Like	Compares if the string matches a regular expression
No Match	NotMatch, Does Not Match, Not Like, Is Not Like, Different, Different From	Compares if a string does not match a regular expression
Within	Inside, Inside Interval, Interval	For integers and real numbers. The interval can be defined as: [0;9], (1;20], 5–10, between 5 and 10, more than 5 and less or equals 10 – see more
Outside	Outside Interval	Opposite to "Within": checks if an integer or real value is outside of the provided interval

Is One Of	Is One, Is One of Many, Is Among, Among	For integer and real numbers, and for strings. Checks if a value is among elements of the domain of values listed through comma
Is Not One Of	Is not among, Not among	For integer and real numbers, and for strings. Checks if a value is NOT among elements of the domain of values listed through comma
Include	Include All	To compare two arrays. Returns true when the first array (decision variable) include all elements of the second array (value within decision table cell)
Exclude	Exclude One Of, Do Not Include, Does Not Include, Include Not All	To compare two arrays. Returns true when the first array (decision variable) does not include all elements of the second array (value within decision table cell). This operator is opposite to the operator "Include"
Exclude All	Do Not Include All, Does Not Include All	To compare two arrays. Returns true when the first array (decision variable) does not include any element of the second array (value within decision table cell)
Intersect	Intersect With, Intersects	To compare an array with an array

If the decision variables do not have an expected type for the specified operator, the proper syntax error will be diagnosed.

Note that the operators **Is One Of, Is Not One Of, Include, Exclude**, and **Does Not Include** work with arrays or lists of values separated by commas. Sometimes a comma could be a part of the value and you may want to use a different separator. In this case you may simply add your separator character at the end of the operator. For example, if you want to check that your variable "Address" is if one of "*San Jose, CA*" or "*Fort Lauderdale, FL*", you may use the operator "**Is One Of #**" with an array of possible addresses described as "*San*

Jose, CA#Fort Lauterdale, FL". Instead of the character '#' you may use any other character-separator.

The following operators can be used inside decision table conclusions:

Operator	Synonyms	Comment
Is	=, ==	Assigns one value to the conclusion decision variable. When you use "=" or "==" inside Excel write"'=" or"'==" to avoid confusion with Excel's own formulas.
Are		Assigns one or more values listed through commas to the conclusion variable that is expected to be an array
Add		Adds one or more values listed through commas to the conclusion variable that is expected to be an array
Assign Plus	+=	Takes the conclusion decision variable, adds to it a value from the rule cell, and saves the result in the same decision variable.
Assign Minus	-=	Takes the conclusion decision variable, subtracts from it a value from the rule cell, and saves the result in the same decision variable.
Assign Multiply	*=	Takes the conclusion decision variable, multiplies it by a value from the rule cell, and saves the result in the same decision variable.
Assign Divide	/=	Takes the conclusion decision variable, divides it by a value from the rule cell, and saves the result in the same decision variable.

Note that for the operators **Add** and **Are** that work with arrays of values separated by commas, you may add your own character-separator at the end of the operator to use it in cases when values inside array contain commas.

Conditions and Conclusions without Operators

Sometimes the creation of special columns for operators seems unnecessary, especially for the operators "Is" and "Within". OpenRules® allows you to use a simpler format as in this decision table:

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

As you can see, instead of keywords “Condition” and “Conclusion” we use the keywords “If” and “Then” respectively. While this decision table looks much simpler in comparison with the functionally identical decision table defined above, we need to make an implicit assumption that the lower and upper bounds for the intervals “0-11”, “11-17”, etc. are included.

Executing Decision Tables from Decision Tables

You can use an action of the type "ActionExecute" to execute a decision table from a cell of another decision table. It is like the ActionExecute used in the tables of the type "Decision". Here is a simple example:

DecisionTable DeterminePayment		
Condition		ActionExecute
Payment Type		**Execute**
Is	MA	CalculatePaymentMA
Is	MC	CalculatePaymentMC

The cells inside the column ActionExecute contain the names of decision tables you want to be invoked (executed) from the current decision table.

Expressions Inside Decision Tables

OpenRules® allows you to use expressions (formulas) in the decision table cells.

There are two types of expressions:

1) OpenRules Java Snippet Expressions with Macros
2) DMN FEEL Expressions

OpenRules from the very beginning allows its users to write any arithmetic and logical expressions directly in the decision table cells. Such should be written as syntactically correct Java snippets even if a user does not now Java. To simplify the references to decision variables inside such expressions we introduced special macros. This is a preferred, the most powerful efficient way to write expressions. However, the DMN standard introduced a special Friendly Enough Expression Language (FEEL) that looks very close OpenRules Java snippets. However, DMN FEEL formulas essentially simpler and do not require starting expressions with special characters like :=. They even allow spaces in the decision variable names. Since the release 6.4.0 OpenRules supports both Java Snippets and DMN FEEL.

Using Java Snippets

You can use the following DMN FELL expression to define the variable "Result" inside OpenRules decision table of the type "DecisionTableAssign":

DecisionTableAssign DefineResult	
Variable	**Value**
Result	Greeting + ", " + Salutation + Name + "!"

to define the decision variable "Result" by concatenating the values of variables "Greeting", "Salutation" and "Name". However, you also can use the following Java snippet to do the same:

DecisionTable DefineResult	
Conclusion	
Result	
Is	::= ${Greeting} + ", " + ${Salutation} + customer(decision).name + "!"

In this expression ${Greeting} and ${Salutation} are so called "macros" that refer to the already calculated values of the decision variables "Greeting" and "Salutation"; the method customer(decision) defined in Excel returns the object Customer whose name will be added to the Result.

In general, Java snippets can be presented in one of the following formats:

```
::= expression or := expression
```

where "expression" can be written using standard Java expressions and macros for decision variables. Typically you use expressions with preceding "::=" in conditions that expect a text expression (the standard Java type String). When it is necessary to return other types (such a int, double, or Date) the expression may be preceded by ":=" (with only 1 semicolon).

Note. Actually ::=expression simply surrounds :=expression with brackets and concatenates it with an empty String as below:

```
:= "" + (expression)
```

Using Macros in OpenRules Expressions

Below is the list of currently available macros that can be used inside Java snippets within Excel tables of the type "Decision", "DecisionTable", and "Method":

Macro Format	Variable Type	Expanded As
${variable name}	**S**tring	decision.getString("variable name")
$I{variable name}	**I**nteger	decision.getInt("variable name")
$R{variable name}	**R**eal	decision.getReal("variable name")
$L{variable name}	**Long**	decision.getLong("variable name")
$D{variable name}	**D**ate	decision.getDate("variable name")
$B{variable name}	**B**oolean	decision.getBool("variable name")
$G{variable name}	Big**D**ecimal	decision.getBigDecimal("variable name")
$G{number}	Big**D**ecimal	new BigDecimal(number)
$V{variable name}	**V**ar	getVar(decision,"variable name") - used by Rule Solver
$O{ObjectName}	**Object**	((ObjectName)decision.getBusinessObject("ObjectName"))

The character after $ indicates the variable type following the variable name in curly brackets. The name like ${variable name} is used for String variables. The following example uses macros for real and integer decision variables:

DecisionTable CalculateTotalPayments	
Conclusion	
Total Payments	
Is	::= $R{Tax Withheld} + $R{Earned Income Credit} + $I{Extra}

Inside the expressions you may use any operator "+", "-", "*", "/", "%" and any other valid Java operator. You may freely use parentheses to define a desired execution order. You also may use any standard or 3rd party Java methods and functions, e.g. ::= Math.min($R{Line A}, $R{Line B}).

If the content of the decision table cell contains only a value of the text variable, say "Customer Location", then along with ::= ${Customer Location} you may also write $Customer Location even without ::= and {..}. In the following table

DT DefineWhomToCharge					
Condition		Condition		Conclusion	
Vendor		**Provider**		**Charged Entity**	
Is Empty	FALSE			Is	$Vendor
		Is Empty	TRUE	Is	UNKNOWN
		Is Not One Of	ABC, KLM, XYZ	Is	$Provider

the conclusion-column contains references $Vendor and $Provider to the values of decision variables Vendor and Provider. You may also use similar references inside arrays. For example, to express a condition that a Vendor should not be among providers, you may use the operator “Is Not One Of” with an array “ABC, $Vendor, XYZ”.

The macro **$O{ObjectName}** is used when you need to refer to an object "ObjectName" which corresponds to as a business concept inside the glossary.
If you use this macro, make sure that "ObjectName" is specified exactly as the proper Java class or Excel Datatype - no spaces allowed. Here is an example from the sample project "DecisionHello":

DecisionTable DefineResult	
Conclusion	
Result	
Is	::= ${Greeting} + ", " + ${Salutation} + $O{Customer}.name + "!"

Dealing with String Variables

You can use names of String variables inside decision table cells. If the content of the cell is a valid name of the string variable, it will be replaced by its value. For example, in the sample-project "DecisionHello" to assigns a complete customer's greeting to the decision variable "Result" you also can use string concatenation operation as in this decision table:

DecisionTableAssign DefineResult	
Variable	**Value**
Result	Greeting + ", " + Salutation + Name + "!"

You may use "" (double quotes) in the action cells to assign an empty string to a String variable.

Dealing with Date Variables

OpenRules naturally supports date comparison like in the following example:

DecisionTable DefineChild		
Condition		Action
Date of Birth		**Is Child**
<	January 1, 2010	FALSE
>=	January 1, 2010	TRUE

OpenRules assumes that Date variables are presented in the standard format specific for a particular country (locale). For example, the standard US date formats are: "MM/dd/yyyy" and "EEE MMM dd HH:mm:ss zzz yyyy". Additionally to the local formats, OpenRules always tries to recognize the following formats:

- MM/dd/yy HH:mm
- yyyy-MM-dd

To compare two Date variables, you may use macros as below:

DecisionTable CompareDates		
Condition		Message
Date 1		**Message**
<	$Date 2	Date 1 < Date 2
>=	$Date 2	Date 1 >= Date 2

If FEEL is On, you may simply write the name of the Date variable without preceding '$':

DecisionTable CompareDates		
Condition		Message
Date 1		**Message**
<	Date 2	Date 1 < Date 2
>=	Date 2	Date 1 >= Date 2

You may see more examples of how to use of new Date operators by analyzing the sample-project "HelloWithDates" in the workspace "openrules.models".

By default, OpenRules compares dates ignoring time. If you want to use time components of the Date variables, instead of the operators such as "<" you should to use the operator "< time", as in the table below:

DecisionTable ComparePassengerFlights							
Condition		Condition		Condition		Action	Action
Flight 1 Is Suitable		**Flight 2 Is Suitable**		**Flight 1 Arrival**		**Flight 1 Score**	**Flight 2 Score**
Is	TRUE	Is	FALSE			1	0
Is	FALSE	Is	TRUE			0	1
Is	TRUE	Is	TRUE	< time	Flight 2 Arrival	1	0
Is	TRUE	Is	TRUE	> time	Flight 2 Arrival	0	1
Is	TRUE	Is	TRUE	= time	Flight 2 Arrival	1	1

This table is a part of the decision model "FlightRebooking" included in the workspace "openrules.models".

When you need to apply arithmetic operations with date variables such as calculating a number of years, months, or days between dates, you still need to use OpenRules Java snippets. For these purposes you may use static methods

of the class "Dates" included in the standard OpenRules library "com.openrule.tools". For example, you may use the following Java snippet inside a condition cell of your decision table:

:= Dates.years($D{Date1}, $D{Date2}) >= 2

It checks that a number of years passed between the variables "Date1" and "Date2" is at least 2 years. You may calculate the age of the person from its birthday as follows:

Similarly use the following methods:

Dates.months(Date d1, Date2 d2)

Dates.monthsToday(Date date)

Dates.days(Date d1, Date d2)

Dates.daysToday(Date d).

The standard library "com.openrule.tools" also includes methods that produce new dates:

addHours(date, hours)

addDays(date,days)

addMonths(date,months)

addYears(date,years)

setYear(date,year)

setMonth(date,month)

setDay(date,day)

today()

newDate(year,month,day)

newDate("yyyy-mm-dd")

You also may get integer values of year, month, and day by calling Dates methods getYear(date), getMonth(date), and getDay(date).

All these methods can be used for dates arithmetic like in this example:

DecisionTableAssign DefineDates	
Variable	Value
Age	::= Dates.yearsToday($D{Date of Birth})
Date of Birth plus 6 Years	::= Dates.addYears($D{Date of Birth}, 6)
Today	::= Dates.today()

You just need to remember to add an "import.java" statement that points to "com.openrules.tools.Dates" to your Environment table.

Using Big Decimals

OpenRules® supports Java type "**BigDecimal**" inside OpenRules formulas, macros, and expressions. It allows a user to deal with calculations that require a high degree of precision (much higher than regular real numbers), such as when dealing with currency conversion, taxes or even high accuracy mathematical calculation.

The variables of the type BigDecimal may correspond (in the Glossary) to Java object attributes of the standard Java type *java.math.BigDecimal*. As described in the above table, you may use the macro **$G{variable name}** to refer to the value of the "variable name" of the type BigDecimal. Here the

letter "G" stands for "Giant" or "BIG" (note that $B is used for Booleans). For example, you may define decision variables Cost and Rate with expected type BigDecimal in your glossary, and then use them in formulas inside decision table cells like in the following example:

::= $G{Cost}.divide($G{Rate})

Here the operator "divide" is the standard Java BigDecimal method. If you want to specify a precision, you even may write

::= $G{Cost}.divide($G{Rate}, MathContext.DECIMAL128)

You may use BigDecimal constants inside BigDecimal expressions by writing **$G{number}.** For example, to present a real number 15.75 as a BigDecimal you may simply write **$G{15.75}**. You use such BigDecimal constants inside BigDecimal operations like below:

::= $G{Total Tax}.divide($G{15.75})

Internally OpenRules® will replace this expression with the following valid Java expression:

::= decision.getBigDecimal("Total Tax").divide(new BigDecimal(15.75),
MathContext.DECIMAL128)

Note. You may also use negative BigDecimals like $G{-100.25}. However, instead of $G{+100} you should simply write $G{100}.

Using Regular Expressions in Decision Table Conditions

OpenRules® allows you to use standard regular expressions. Operators "Match" and "No Match" (and their synonyms from the above table) allow you to match the content of your text decision variables with custom patterns such as phone number or Social Security Number (SSN). Here is an example of a decision table that validates SSN:

DecisionTable testSSN		
Condition		Message
SSN		**Message**
No Match	\d{3}-\d{2}-\d{4}	Invalid SSN
Match	\d{3}-\d{2}-\d{4}	Valid SSN

The use of this decision table is described in the sample project "DecisionHelloJava".

Using Variable Names with Spaces

DMN FEEL allows a user to freely use names with spaces inside its expression. Of course, users may use "underscores" instead of spaces between words like Patient_Creatinine_Level or PatientCreatinineLevel. However, we did not want to diminish FEEL "friendliness" and allowed our user to use variable names with spaces inside FEEL expressions. So, OpenRules allows you to write expressions similar the one from our Patient Therapy example:

DecisionTable CalculateCreatinineClearance
Action
Patient Creatinine Clearance
(140 - Patient Age) * Patient Weight / (Patient Creatinine Level * 72)

If a user prefers to explicitly specify that a combination of words separated by spaces is a variable name, she may surround such names with apostrophes like in the example below:

DecisionTable CalculateCreatinineClearance
Action
Patient Creatinine Clearance
(140 - 'Patient Age') * 'Patient Weight' / ('Patient's Creatinine Level' * 72)

Note that variable name may include apostrophes like 'Patient's Creatinine Level' above and OpenRules is capable to handle such names inside FEEL formulas without any additional indicators. However, if the name contains special characters like valid operators - or *, a user needs to surround the variable name with apostrophes to avoid a possible confusion.

Instead of apostrophes a user may define other symbols, e.g. $, & or #, by calling the following code just before executing the proper decision:

decision.put("LONG_NAME_INDICATOR", "$");

Note. In Excel the very first apostrophe in the cell is used as an indicator that the next character doesn't start an Excel's own formula. So, if your FEEL's formula starts with *'long variable name'*, you need to double the first apostrophe: *''long variable name'.*

Turning FEEL Processing On/Off

Processing of FEEL expressions is done during the decision execution and may be less efficient to compare with the standard OpenRules expressions. To avoid any overhead for the existing customers who do not want to use FEEL, by default FEEL processing if OFF. To turn it on, before executing your decision you need to write:

decision.put("FEEL", "On");

Allowed FEEL Operators and Predefined Functions

The current FEEL implementation allows you to use the standard arithmetic and logical operators and functions for integer and real numbers – see examples in the table below. It works only with numerical decision variables. It utilizes an open source package "expr" initially developed by Darius Bacon but essentially modified and now supported by OpenRules.

Feature	Syntax	Examples
Numbers	regular integer or real numbers	10, 465.25, -25, 3.14
Add	x + y	3+2
Subtract	x - y	3 - 2
Multiply	x * y	3 * 2
Divide	x / y	3/2
Power: x^y	x**y or x^y	5**2
Negate	-x	-3
Comparison	x < y x <= y	2 <> 3 [produces 1]

	x = y x <> y or x != y x >= y x > y	2 <> 2 [produces 0]
Logical "and"	x and y	1 and 1 [produces 1] 1 and 0 [produces 0] 0 and 0 [produces 0]
Logical "or"	x or y	1 and 1 [produces 1] 1 and 0 [produces 1] 0 and 0 [produces 0]
Absolute value	abs(x)	abs(-5) [produces 5] abs(5) [produces 5]
Maximum between two numbers	max(x,y)	max(5,6) [produces 6]
Minimum between two numbers	min(x,y)	min(5,6) [produces 5]
Floor	floor(x)	floor(3.5) [produces 3] floor(-3.5) [produces -3]
Ceiling	ceil(x) or ceiling(x)	ceil(3.5) [produces 4] ceil(-3.5) [produces -3]
Additional functions		
The mathematical constant "π"	pi	
e^x	exp(x)	exp(1) = 2.7182818284590451
Rounding	round(x)	round(3.5) [produces 4] round(-3.5) [produces -4]
Conditional	if(x,y,z)	if(1,50, 100) [produces 50] if(0,50,100) [produces 100]
Square root	sqrt(x)	sqrt(9) [produces 3]
Trigonometric functions	sin(x), cos(x), tan(x), asin(x), acos(x), atan(x)	sin(pi/2) [produces 1]

Using '-' as a Not Applicable Symbol Inside Decision Tables

OpenRules historically leaves decision table cells empty when the proper conditions and/or actions are not applicable. However, DMN requires using the symbol '-' in these cases. So, now we allow a user to use both possibilities interchangeably.

Defining Business Glossary

While defining decision tables, we freely introduced different decision variables assuming that they are somehow defined. The business glossary is a special OpenRules® table that defines all decision variables. The Glossary table has the following structure:

Glossary glossary			
Variable	**Business Concept**	**Attribute**	**Domain**

The first column simply lists all of the decision variables using exactly the same names that were used inside the decision tables. The second column associates different decision variables with the business concepts to which they belong. Usually you want to keep decision variables that belong to the same business concept together and merge all rows in the column "Business Concept" that share the same concept. Here is an example of a glossary from the standard OpenRules® example "LoanPreQualification":

Glossary glossary			
Variable	**Business Concept**	**Attribute**	**Domain**
Monthly Income	Customer	monthlyIncome	0-5000000
Monthly Debt		monthlyDebt	0-100000
Mortgage Holder		mortgageHolder	Yes,No
Outside Credit Score		outsideCreditScore	0-999
Loan Holder		loanHolder	Yes,No
Credit Card Balance		creditCardBalance	-1000000 - 100000000
Education Loan Balance		educationLoanBalance	-1000000 - 100000000
Internal Credit Rating		internalCreditRating	A,B,C,D,F
Internal Analyst Opinion		internalAnalystOpinion	High,Mid,Low
Amount	LoanRequest	amount	100-10000000
Term		term	36,48,60,72
Purpose		purpose	
Total Income		totalIncome	0-500000
Total Debt		totalDebt	0-500000
Income Validation Result		incomeValidationResult	SUFFICIENT,UNSUFFICIENT,?
Debt Research Result		debtResearchResult	High,Mid,Low,?
Loan Qualification Result		loanQualificationResult	QUALIFIED, NOT QUALIFIED, ?

All rows for the concepts such as “Customer” and “Request” are merged.

The third column “Attribute” contains “technical” names of the decision variables – these names will be used to connect our decision variables with attributes of objects used by the actual applications, for which a decision has been defined.

The application objects could be defined in Java, in Excel tables, in XML, etc. The decision does not have to know about it: the only requirement is that the attribute names should follow the usual naming convention for identifiers in languages like Java: it basically means no spaces allowed.

The last column, “Domain”, is optional, but it can be useful to specify which values are allowed to be used for different decision variables. Decision variable domains can be specified using the naming convention for the intervals and domains described below. The above glossary provides a few intuitive examples of such domains. These domains can be used during the validation of a decision.

It is helpful to use columns on the right of the glossary (not included in it) for comments as explain the meaning of different decision variables. Similarly you may use Excel comments.

Defining Test Data

OpenRules® provides a convenient way to define test data for decisions directly in Excel without the necessity of writing any Java code. A non-technical user can define all business concepts in the Glossary table using Datatype tables. For example, here is a Datatype table for the business concept “Customer” defined above:

Datatype Customer	
String	fullName
String	SSN
int	monthlyIncome
int	monthlyDebt
String	mortgageHolder
int	outsideCreditScore
String	loanHolder
int	creditCardBalance
int	educationLoanBalance
String	internalCreditRating
String	internalAnalystOpinion

The first column defines the type of the attribute using standard Java types such as “int”, “double”, “Boolean”, “String”, or “Date”. The second column contains the same attribute names that were defined in the Glossary. To create an array of objects of the type “Customer” we may use a special “Data” table like the one below:

Data Customer customers										
fullName	SSN	monthlyIncome	monthlyDebt	mortgageHolder	outsideCreditScore	loanHolder	creditCardBalance	educationLoanBalance	internalCreditRating	internalAnalystOpinion
Borrower Full Name	**Borrower SSN**	**Monthly Income**	**Monthly Debt**	**Mortgage Holder**	**Outside Credit Score**	**Loan Holder**	**Credit Card Balance**	**Education Loan Balance**	**Internal Credit Rating**	**Internal Analyst Opinion**
Peter N. Johnson	157-82-5344	5000	2300	Yes	720	No	2500	0	A	Low
Mary K. Brown	056-45-8233	4300	2800	No	620	No	5654	23800	B	Low
Robert Cooper Jr.	241-56-9082	6400	2800	Yes	735	Yes	1200	0	C	Mid

This table is too wide (and difficult to read), so we could transpose it to a more convenient but equivalent format:

Data Customer customers				
fullName	**Borrower Full Name**	Peter N. Johnson	Mary K. Brown	Robert Cooper Jr.
SSN	**Borrower SSN**	157-82-5344	056-45-8233	241-56-9082
monthlyIncome	**Monthly Income**	5000	4300	6400
monthlyDebt	**Monthly Debt**	2300	2800	2800
mortgageHolder	**Mortgage Holder**	Yes	No	Yes
outsideCreditScore	**Outside Credit Score**	720	620	735
loanHolder	**Loan Holder**	No	No	Yes
creditCardBalance	**Credit Card Balance**	2500	5654	1200
educationLoanBalance	**Education Loan Balance**	0	23800	0
internalCreditRating	**Internal Credit Rating**	A	B	C
internalAnalystOpinion	**Internal Analyst Opinion**	Low	Low	Mid

Now, whenever we need to reference the first customer we can refer to him as customers[0]. Similarly, if you want to define a doubled monthly income for the second custromer, "Mary K. Brown", you may simply write

```
::= (customers[1].monthlyIncome * 2)
```

You can find many additional details about data modeling in this section.

Connecting Decisions with Business Objects

To tell OpenRules® that you want to associate the object customers[0] with the business concept "Customer" defined in the Glossary, you need to use a special table "DecisionObject" that may look as follows:

DecisionObject decisionObjects	
Business Concept	**Business Object**
Customer	:= customers[0]
Request	:= loanRequests[0]
Internal	:= internal

Here we also associate other business concepts namely Request and Internal with the proper business objects – see how they are defined in the standard example "LoanPreQualification".

The above table connects a decision with test data defined by business users directly in Excel. This allows the decision to be tested. However, after the decision is tested, it will be integrated into a real application that may use objects defined in Java, in XML, or in a database, etc. For example, if there are instances of Java classes Customer and LoanRequest, they may be put in the object "decision" that is used to execute the decision. In this case, the proper table "decisionObjects" may look like:

DecisionObject decisionObjects	
Business Concept	**Business Object**
Customer	:= decision.get("customer")
Request	:= decision.get("loanRequest")
Internal	:= internal

It is important that the decision model does not "know" about a particular object implementation: the only requirement is that the attribute inside these objects should have the same names as in the glossary.

Decision Execution

OpenRules® provides a template for Java launchers that may be used to execute different decisions. When you want to build and to test the standard decision models in the workspace "openrules.models" you simple double-click (from Windows Explorer) on files "build.bat" and "run.bat".

You also may launch your decision model against test data using a Java launcher that utilizes OpenRules® Java API. Here is an example of a Java launcher for the sample project "LoanPreQualification":

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

public class Main {
    public static void main(String[] args) {
        String fileName = "file:rules/Test.xls";
        Decision decision = new Decision("DecisionLoanQualificationResult",fileName);
        decision.put("report", "On");
        decision.put("FEEL", "On");
        decision.test("testCases");
    }
}
```

It creates an instance of the class Decision. It has only two parameters:

1) a path to the main Excel file "Test.xls"
2) a name of the main decision "DecisionLoanQualificationResult" inside this Excel file "Goals.xls".

When you execute this Java launcher using the provided batch file "run.bat" or execute it from your Eclipse IDE, it will produce output that may look like as follows:

```
RUN TEST: Test 1 2018-08-13 15:47:42.752
  Assign: Total Debt = 165600.0
  Assign: Total Income = 360000.0
  Assign: Income Validation Result = SUFFICIENT
  Assign: Debt Research Result = High
  Assign: Debt Research Result = Low
  Assign: Loan Qualification Result = NOT QUALIFIED
ADDITIONAL DEBT RESEARCH IS NEEDED [produced by DetermineLoanPreQualificationResult]
Validating results for the test <Test 1>
Test 1 was successful
Executed test Test 1 in 88 ms

RUN TEST: Test 2 2018-08-13 15:47:42.84
  Assign: Total Debt = 100800.0
  Assign: Total Income = 154800.0
  Assign: Income Validation Result = SUFFICIENT
  Assign: Debt Research Result = Low
  Assign: Debt Research Result = Low
  Assign: Loan Qualification Result = NOT QUALIFIED
ADDITIONAL DEBT RESEARCH IS NEEDED [produced by DetermineLoanPreQualificationResult]
Validating results for the test <Test 2>
Test 2 was successful
Executed test Test 2 in 83 ms

RUN TEST: Test 3 2018-08-13 15:47:42.923
  Assign: Total Debt = 67200.0
  Assign: Total Income = 153600.0
  Assign: Income Validation Result = SUFFICIENT
  Assign: Debt Research Result = High
  Assign: Debt Research Result = Mid
  Assign: Loan Qualification Result = QUALIFIED
The borrower has been successfully pre-qualified for the requested loan [produced by
Validating results for the test <Test 3>
Test 3 was successful
Executed test Test 3 in 20 ms
All 3 tests succeeded!
Executed all tests cases in 194 ms - 2018-08-13 15:47:42.944
```

This output shows all sub-decisions and conclusion results for the corresponding decision tables. Because our launcher requested the generation of the html-reports using decision.put("report","On"), the proper reports will be generated and will be useful to explain why certain decisions were made. Here is an execution report for the Test 1:

Decision Table : Rule#	Executed Rule	Variables and Values
DetermineIncomeValidationResult:1	**IF** Total Income >Total Debt * 1.4 **THEN** Income Validation Result = SUFFICIENT	Total Income=360000.0 Total Debt=165600.0 Income Validation Result=SUFFICIENT
DetermineDebtResearchResult:1	**IF** Mortgage Holder = Yes **THEN** Debt Research Result = High	Mortgage Holder=true Debt Research Result=High
DetermineDebtResearchResult:13	**IF** Internal Analyst Opinion = Low **THEN** Debt Research Result = Low	Internal Analyst Opinion=Low Debt Research Result=Low
DetermineLoanPreQualificationResult:2	**IF** Income Validation Result = SUFFICIENT **AND** Debt Research Result = Low **THEN** Loan Qualification Result = NOT QUALIFIED **AND** Message ADDITIONAL DEBT RESEARCH IS NEEDED	Income Validation Result=SUFFICIENT Debt Research Result=Low Loan Qualification Result=NOT QUALIFIED

MORE COMPLEX DECISION TABLES

In real-world projects you may need much more complex representations of rule sets and the relationships between them than those allowed by the default decision tables. OpenRules® allows you to use advanced decision tables and to define your own rule sets with your own logic.

Multi-Hit Decision Tables

By default, decision tables of the type "DecisionTable" are single-hit meaning that after the execution of the first satisfied rule the table ends its work. You even can use a keyword **DecisionTableSingleHit** along with **DecisionTable**. However, sometimes this behavior is not sufficient.

OpenRules® provide two additional types of decision tables:

- **DecisionTable1** or **DecisionTableMultiHit** (or DT1)
- **DecisionTable2** or **DecisionTableSequence** (or DT2).

DecisionTableMultiHit

Contrary to the standard single-hit DecisionTable, decision tables of type "DecisionTable1" or "DecisionTableMultiHit" are implemented as multi-hit decision tables. "DecisionTable1" supports the following rules execution logic:

1. All rules are evaluated and if their conditions are satisfied, they will be marked as "to be executed"
2. All action-columns (such as "Conclusion", "Then", "Action", "ActionAny", or "Message") will be executed in the top-down order for the list of "to be executed" rules.

Thus, you should make two important observations about the behavior of the multi-hit decision tables:

- **Rule actions cannot affect the conditions of any other rules** in the decision table – there will be no re-evaluation of any conditions
- **Rule overrides are permitted**. The action of any executed rule may override the action of any previously executed rule.

Let's consider an example of a rule that states: "A person of age 17 or older is eligible to drive. However, in Florida 16-year-olds can also drive". If we try to present this rule using the default single-hit decision table, it may look as follows:

DecisionTable ValidateDrivingEligibility					
Condition		Condition		Conclusion	
Driver's Age		**US State**		**Driving Eligibility**	
>=	17			Is	Eligible
Is	16	Is Not	Florida	Is	Not Eligible
Is	16	Is	Florida	Is	Eligible
<	16			Is	Not Eligible

Using a multi-hit decision table, we may present the same rule as:

DecisionTable1 ValidateDrivingEligibility					
Condition		Condition		Conclusion	
Driver's Age		US State		Driving Eligibility	
				Is	Eligible
<	17			Is	Not Eligible
>=	16	Is	Florida	Is	Eligible

In the DecisionTable1 the first unconditional rule will set "Driving Eligibility" to "Eligible". The second rule will reset it to "Not Eligible" for all people younger than 17. But for 16-year-olds living in Florida, the third rule will override the variable again to "Eligible".

It is also very convenient to use multi-hit decision tables to accumulate some data. For example, the following decision table accumulates "Applicant Credit Score" based on 4 different conditions:

DecisionTableMultiHit ApplicantCreditScoreDecisionTable					
If	If	If	If	Conclusion	
Applicant Number of Default Payments in Last 12 Months	Applicant had declared Bankrupcy	Applicant Years with Current Account Bank	Applicant Amount of Available Credit Used Percentage	Applicant Credit Score	
[1 - 3]					100
[4-6]					50
>6					0
0					250
	TRUE				0
	FALSE				250
		< 1		+=	50
		[1-3]			150
		>3			250
			[0-24]		200
			[25-49]		249
			[50-74]		150
			[75-100]		100
			>100		0

Note that the operator "+=" increments the score using a value provided by the proper row (rule) of the last column.

DecisionTableSequence

There is one more type of decision table, "**DecisionTable2**" or "**DecisionTableSequence**" that is similar to "DecisionTable1", but allows the actions of already executed rules to affect the conditions of rules specified below them. "DecisionTable2" supports the following rules execution logic:

1. Rules are evaluated in top-down order and if a rule condition is satisfied, then the rule actions are immediately executed.
2. Rule overrides are permitted. The action of any executed rule may override the action of any previously executed rule.

Thus, you may make two important observations about the behavior of the "DecisionTable2":

- Rule actions can affect the conditions of other rules
- There could be rule overrides when rules defined below already executed rules could override already executed actions.

Let's consider the following example:

DecisionTable2 CalculateTaxableIncome			
Condition		Conclusion	
Taxable Income		**Taxable Income**	
		Is	::= ${Adjusted Gross Income} - ${Dependent Amount}
Is Less	0	Is	0

Here the decision variable "Taxable Income" is present in both the condition and the action. The first (unconditional) rule will calculate and set its value using the proper formula. The second rule will check if the calculated value is less than 0. If it is true, this rule will reset this decision variable to 0.

Tables of the type "Method"

Sometimes you may prefer instead of a graphical decision table to use text formulas. OpenRules offers you a special table of the type "Method" (or a synonymous type "Code") to write a piece of logic in Java. For example, the following decision table

DecisionTable CalculateCreatinineClearance
Action
Patient Creatinine Clearance
(140 - Patient Age) * Patient Weight / (Patient Creatinine Level * 72)

can be also presented as the Method:

Method void CalculateCreatinineClearance(Decision decision)
double pcc = (140 - $I{Patient Age}) * $R{Patient Weight} / ($R{Patient Creatinine Level} * 72); decision.setReal("Patient Creatinine Clearance",round(pcc,0.01));

This method has the word "void" before its name and you must the parameter (Decision decision) after its name. The body of this method is a valid Java snippet. The first statement creates a double variable "pcc" using a formula with macros like $R{Patient Weight} to access decision variables. The second statement uses the method decision.setReal(<variable-name>,<variable-value>) to set "Patient Creatinine Clearance" to the value of pcc rounded to 2 digits after the decimal point.

You may use any valid Java statements (including if-then-else and for-loop) inside the Method's body.

Using Natural Language Inside Decision Tables

OpenRules® allows a rules designer to use "almost" natural language expressions inside rules tables to represent intervals of numbers, strings, dates, etc. Instead of the standard columns of the types "**Condition**" with a special sub-column for an operator, you may specify columns of the types "**If**" that does not require sub-

columns. They allow you to use operators or even natural language expressions together with values to represent different intervals and domains of values.

Integer and Real Intervals

You may use plain English expressions to define different intervals for integer and real decision variables inside IF-columns. You may define FROM-TO intervals in practically unlimited English using such phrases as: "500-1000", "between 500 and 1000", "Less than 16", "More or equals to 17", "17 and older", "< 50", ">= 10,000", "70+", "from 9 to 17", "[12;14)", etc.

You also may use many other ways to represent an interval of integers by specifying their two bounds or sometimes only one bound. Here are some examples of valid integer intervals:

Cell Expression	Comment
5	equals to 5
[5,10]	contains 5, 6, 7, 8, 9, and 10
5;10	contains 5, 6, 7, 8, 9, and 10
[5,10)	contains 5, 6,7,8, and 9 (but not 10)
[5..10)	The same as [5,10)
5 - 10	contains 5 and 10
5-10	contains 5 and 10
5- 10	contains 5 and 10
-5 - 20	contains -5 and 20
-5 - -20	error: left bound is greater than the right one
-5 - -2	contains -5 , -4, -3, -2
from 5 to 20	contains 5 and 20
less 5	does not contain 5
less than 5	does not contain 5
less or equals 5	contains 5
less or equal 5	contains 5
less or equals to 5	contains 5
smaller than 5	does not contain 5

more 10	does not contain 10
more than 10	does not contain 10
10+	more than 10
>10	does not contain 10
>=10	contains 10
between 5 and 10	contains 5 and 10
no less than 10	contains 10
no more than 5	contains 5
equals to 5	equals to 5
greater or equal than 5 and less than 10	contains 5 but not 10
more than 5 less or equal than 10	does not contain 5 and contains 10
more than 5,111,111 and less or equal than 10,222,222	does not contain 5,111,111 and contains 10,222,222
[5'000;10'000'000)	contains 5,000 but not 10,000,000
[5,000;10,000,000)	contains 5,000 but not 10,000,000
(5;100,000,000]	contains 5,000 and 10,000,000

You may use many other ways to represent integer intervals as you usually do in plain English. The only limitation is the following: *min* should always go before *max*!

Along with integer intervals, you may similarly represent intervals of real numbers. The bounds of double intervals could be integer or real numbers such as [2.7; 3.14).

Comparing Integer and Real Numbers

You may use regular comparison operators in front of integer or real numbers to compare them with a decision variable defined inside an IF-column. Examples of acceptable operators:

Cell Expression	Comment
<= 5	less or equals to 5
< 5	strictly less than 5

> 5.25	strictly more than 5.25
>= 5	more or equals to 5
!=	not equal to 5
5	equals to 5. Note that absence of a comparison operator means equality. You cannot use an explicit operator "=" (not to be confused with Excel's formulas).

Comparing Boolean Values

If a parameter type is defined as "boolean", you can use the following values inside rule cells:

- True, TRUE, Yes, YES
- False, FALSE, No, NO

You also may use formulas that produce a Boolean, .e.g.

```
{ loan.additionalIncomeValidationNeeded; }
```

Sometimes, you want to indicate that a condition is satisfied, or an action should be executed. You may use any character like X or * without checking its actual value – the fact that the cell is not empty indicates that the condition is true.

ConditionAny and ConclusionAny

Sometimes your conditions or actions are not related to one decision variable and can be calculated using formulas. For example, a condition can be defined based on combination of several decision variables, and you would not want to artificially add an intermediate decision variable to your glossary in order to accommodate each needed combination of existing decision variables. In such a case, you may use special columns "**ConditionAny**" and "**ConclusionAny**". The titles of these columns do not represent any decision variable and may contain any text. You may use any formulas inside the cells of these columns that execute some custom actions.

ConditionVarOperValue

When your decision table contains too many columns it may become too large and unmanageable. In practice large decision tables have many empty cells because not all decision variables participate in all rule conditions even if the proper columns are reserved or all rules. To make your decision table more compact, OpenRules® allows you to move a variable name from the column title to the rule cells. To do that, instead of the standard column's structure with two sub-columns

Condition	
Variable Name	
Oper	Value

you may use another column representation with 3 sub-columns:

ConditionVarOperValue		
Attribute		
Variable Name	Oper	Value

This way you may replace a wide table with many condition columns like the one below:

DecisionTable classifcationRules										
Condition		Condition			Condition		Conclusion		Conclusion	
C_OTH_EXPNS_AMT		A_ESTATE_TAX_AMT		...	Attribute		ClassifiedAs		HitRate	
>=	398	>=	10054		Oper	Value	Is	High	=	66
							Is	Low	=	63
>=	53						Is	Low	=	49
							Is	Low	=	86
							Is	Low	=	78
							Is	Other	=	56

to a much more compact table that may look as follows:

DecisionTable classifcationRules												
ConditionVarOperValue			ConditionVarOperValue			...	ConditionVarOperValue			Conclusion		Conclusion
Attribute			Attribute			...	Attribute			ClassifiedAs		HitRate
C_OTH_EXPNS_AMT	>=	398	A_ESTATE_TAX_A	>=	10054		Attribute	Oper	Value	Is	High	= 66
E_PRTSCRP_TOT_LO	<=	-6955	AGI_TPI_RATIO	<=	0.95993					Is	Low	= 63
C_OTH_EXPNS_AMT	>=	53	ORD_DIVIDENDS_	<=	6617					Is	Low	= 49
TAXABLE_INC_TPI_R	<=	0.810447	TENT_TAX_AMT	>=	301630					Is	Low	= 86
DIVIDENDS_AND_INT	<=	12348	EXTNSN_PYMNT_	<=	30000					Is	Low	= 78
										Is	Other	= 56

You simply replace a column of the type "Condition" to the one that has the standard type "ConditionVarOperValue". Similarly, instead of a column of the type "Conclusion" you may use a column of the type "ConclusionVarOperValue" with 3 sub-columns that represent a variable name, an operator, and a value.

Assigning Values to Decision Variables

You can use regular conclusions/actions to assign a value to a variable. However, when you need to make multiple assignments, it is more convenient to use a special decision table of the type "**DecisionTableAssign**". Here is an example:

DecisionTableAssign CalculateInternalVariables	
Variable	**Value**
Total Debt	Monthly Debt * Term
Total Income	Monthly Income * Term

The first column contains decision variable, to which the values from the second column will be assigned. The values may be calculated using DMN FEEL formulas (like in this example) or OpenRules Java snippets. This table is equivalent to the regular decision table:

DecisionTable CalculateInternalVariables	
Action	Action
Total Debt	**Total Income**
Monthly Debt * Term	Monthly Income * Term

You can see the entire decision model that uses this decision table in the standard project "LoanPreQualification".

Dealing with Collections of Business Objects

In practice business rules deal not only with separate decision variables but also with arrays of lists of variables. OpenRules provides necessary constructs to add values to such arrays, to check if the arrays contain certain elements, to iterate over arrays making some changes in their elements or making additional calculation. And finally, we need to be able to sort the arrays in accordance with some comparison rules.

Adding Values to Arrays and Lists

You can use the standard conclusions inside decision tables with the operator "Add" to added values to decision variables defined as arrays or lists. The following table shows how to add values only to arrays of the types String[], int[], double[], and Date[]:

DecisionTable AddArrays							
Conclusion		Conclusion		Conclusion		Conclusion	
Regions		Terms		Amounts		Dates	
add	NORTHEAST, MIDWEST, MOUNTAIN, PACIFIC-COAST	add	36,72,120	add	3.14,78.5	add	10/19/1979, 3/21/2014, Thu Dec 31 15:56:48 EST 2015

Instead of arrays of constants such as {36,73,120} you may add an array (or a value) located in the decision variable "MyTerms" (or "MyTerm") by writing $MyTerms (or $MyTerm) in the proper cell, e.g .

DecisionTable AddArrays							
Conclusion		Conclusion		Conclusion		Conclusion	
Regions		Terms		Amounts		Dates	
Add	NORTHEAST, MIDWEST, MOUNTAIN, PACIFIC-COAST	Add	$MyTerms	Add	$External Amounts	Add	10/19/1979, 3/21/2014, Thu Dec 31 15:56:48 EST 2015

If you add only one value (not an array), you may even omit '$', e.g. instead of "$MyTerm" you may simply write "MyTerm".

Similarly the operator "Add" works with arrays of the types long[] and BigDecimal[].

Along with arrays you can use lists of objects. Let's say your class Department includes an attribute “employee” of the type List<Employee> that refers to all employees working in this department. Let's say you want to use rules to define lists of all women and all men among these employees. In your Java class Department these lists can be defined as:

```
List<Employee> women;

List<Employee> men;
```

We may navigate through all Employees using the following table:

DecisionTableIterate EvaluateAllEmployees	
Array of Objects	**Rules**
Employees	EvaluateOneEmployee

To fill out the lists Women and Men we may use this simple decision table:

DecisionTable EvaluateOneEmployee					
Condition		Conclusion		Conclusion	
Gender		**Women**		**Men**	
Is	Female	Add	Employee		
Is	Male			Add	Employee

The proper glossary should contain the variables "Women" and "Men" associated with the concept "Department" with attributes women and men.

Now let's assume we want to define an array of rich employees (whose salary is larger than a certain amount). We may add List<Employees> richEmployees to the class Department.java and associate a variable "Rich Employees" with the concept "Department" in our glossary. Then we may simply expand the rules "EvaluateAllEmployees" as below:

DecisionTableMultiHit EvaluateOneEmployee									
Condition		Condition		Conclusion		Conclusion		Conclusion	
Salary		**Gender**		**Women**		**Men**		**Rich Employees**	
		Is	Female	Add	Employee				
		Is	Male			Add	Employee		
>=	85000							ADD	Employee

You don't have to change anything in these rules if instead of lists you use arrays of objects inside your Java objects. However, for big arrays performance may suffer to compare with lists. For example, in the class Department.java we may replace

```
List<Employees> richEmployees;
```

with

```
Object[] richEmployees;
```

Our decision tables will continue to work (probably slightly slower). Please note that in this case you have to use "Object[]" instead of "Employee[]" - this is an (unfortunate) implementation restriction.

You can see all implementation details in the sample-project "AggregatedValuesWithLists".

Special Operators for Numeric Arrays

There are several operators that simplify manipulations with numeric arrays such as int[], double[], long[], and BigDecimal[]. Here they are:

Min, Max, Average, Count, Sum, Product, One Element, All Elements

The following example explains how to use these operators:

DecisionTableMultiHit CheckAmounts		
ConditionRealArray		Message
Amounts		**message**
All Elements	> 5000	All amounts > 5000
Average	> 8000	Average amount > 8000
One Element	> 15000	There are amounts > 15K
Sum	< 1000000	Sum of all amount < 1M
Count	> 3	Array Amount has more than 3 elements

This decision table uses the condition type "**ConditionRealArray**". The name of the array "Amount" should be placed to the column title. The first sub-column contains one of the above operators, and the second sub-column contains any valid FEEL expressions for these types of numbers. For arrays of integers and big decimals, you may similarly use conditions of the new types "**ConditionIntArray**", "**ConditionLongArray**", and "**ConditionBigDecimalArray**".

The first 6 operators also can be used in the standard columns of the type "Conclusion" along with array names to calculate their Min, Max, Average, Count, Sum or Product. For example, this table

DecisionTable CalculateArrayFeatures			
Conclusion		Conclusion	
Average Amount		**Minimal Term**	
Average	Amounts	Min	Terms

calculates Average Amount for the integer array "Amounts" and "Minimal Term" for the real array "Terms". You may try these new operators by running the sample-project "DecisionNumericArrays" from the workspace "openrules.dmn".

Iterating Through Collections of Business Objects

You can always use Java snippets with regular Java loops to navigate through collections (arrays) of business objects. However, OpenRules® provides more business-friendly capabilities to deal with array and list of objects. They allow a user to define which decision tables to execute against a collection of objects and to calculate values defined on the entire collection.

Let's consider an example "**DecisionManyCustomers**" in the standard workspace "openrules.dmn". There is a standard Java bean Customer with

different customer attributes such as name, age, gender, salary, etc. There is also a Java class "CollectionOfCustomers":

```
public class CollectionOfCustomers {
   Customer[] customers;
   int minSalary;
   int maxSalary;
   int numberOfRichCustomers;
   int totalSalary;
   ...
}
```

We want to pass this collection to a decision that will process all customers from this collection in one run and will calculate such attributes as "minSalary", "totalSalary", "numberOfRichCustomers", and similar attributes, which are specified for the entire collection. Each customer within this collection can be processed by the following rules:

DecisionTable1 EvaluateOneCustomer											
Condition		Conclusion		Conclusion		Conclusion		Conclusion		Conclusion	
Salary		**Wealth**		**Total Salary**		**Number of Rich Customers**		**Maximal Salary**		**Minimal Salary**	
				+=	::= $I{Salary}			Max	::= $I{Salary}	Min	::= $I{Salary}
>	100000	Is	Rich			+=	1				

Pay attention that we use here a multi-hit table (DecisionTable1), so both rules will be executed. The first one unconditionally calculates the Total Salary, Maximal and Minimal Salaries. The second rule defines Number of Rich Customers inside the collection. To accumulate the proper values, we use the existing operator "+=" and newly introduced operators "Min" and "Max".

To execute the above decision table for all customers, we will utilize a new action "ActionRulesOnArray" within the following decision table:

DecisionTable CalculateCustomerTotals								
Conclusion		Conclusion		Conclusion		ActionRulesOnArray		
Total Salary		**Maximal Salary**		**Minimal Salary**		**Array of Objects**	**Object Type**	**Rules**
Is	0	Is	0	Is	1000000	Customers	Customer	EvaluateOneCustomer

Here the first 3 actions (conclusions) simply define initial values of collection attributes. The last action has 3 sub-columns:

- The name of the array of objects as it is defined in the glossary ("Customers")
- The type of those objects ("Customer")
- The name of the decision table ("EvaluateOneCustomer") that will be used to process each object from this collection.

Instead of the action "ActionRulesOnArrays" you use the action of the "**ActionIterate**" that allows you to omit the sub-column "Object Type" like in the following example:

DecisionTableMultiHit DefineAllDriverPremiums	
ActionIterate	
Array of Objects	**Rules**
Drivers	DefineDriverPremium

Thus, a combination of the two regular decisions tables (similar to the above ones) provides business users with a quite intuitive way to apply rules over collections of business objects without necessity to deal with programming constructions.

DecisionTableIterate for Iteration Over Arrays and Lists

You can use a special decision table of the type "**DecisionTableIterate**" to iterate over arrays or lists of business objects. Here is an example:

DecisionTableIterate IterateOverDriversAndCars	
Array of Objects	**Rules**
Drivers	DefineDriverEligibilityRating
Drivers	DefineDriverEligibilityScore
Cars	DefineAutoEligibilityRating
Cars	DefineAutoEligibilityScore

This table iterates over arrays of objects, defined in the first column, applying the rules, defined in the second column, to every element of these arrays. This

example is borrowed from the sample project "DecisionUserv", which glossary consists of 3 business concepts: Driver, Car, and Client. Client defines decision variables Drivers and Cars that are arrays of objects which have types Driver and Car correspondingly.

Along with arrays of different objects, you also can use lists of objects. You may see how it's done in the sample project DecisionSortPassengersList, in which instead of Passenger[] array we use List<Passengers>.

DecisionTableSort for Array Sorting

You can use a special decision table of the type "**DecisionTableSort**" to sort arrays of objects using regular decision tables that compare any two elements of such arrays. Consider the following example provided in the standard project "**DecisionSortPassengers**":

DecisionTableSort SortPassengers
Array of Objects
Passengers

This table sorts the arrays "Passengers", which elements have type "Passenger". During the sorting process, every two elements of this array are compared using the following rules:

DecisionTable ComparePassengers							
Condition		Condition		Condition		Action	Action
Passenger 1 Status		**Passenger 2 Status**		**Passenger 1 Miles**		**Passenger 1 Score**	**Passenger 2 Score**
Is	GOLD	Is One Of	SILVER, BRONZE			1	0
Is		Is	GOLD	>	Passenger 2 Miles	1	0
Is		Is		<	Passenger 2 Miles	0	1
Is	SILVER	Is	GOLD			0	1
Is		Is	BRONZE			1	0
Is		Is	SILVER	>	Passenger 2 Miles	1	0
Is		Is		<	Passenger 2 Miles	0	1
Is	BRONZE	Is One Of	GOLD,SILVER			0	1
Is		Is	BRONZE	>	Passenger 2 Miles	1	0
Is		Is		<	Passenger 2 Miles	0	1

This table assigns scores to each pair of passengers (Passenger 1 and Passenger 2) - a passenger with higher credentials is supposed to receive a higher score. The name of this table "ComparePassengers" is composed by the word "Compare" and the name of the array "Passengers" (with omitting spaces if any). If you prefer to explicitly define the comparison rules, you can do it using the following decision table format:

DecisionTable SortUsingRules SortPassengers	
Array of Objects	**Comparison Rules**
Passengers	ComparePassengers

It's assumed that the elements of the array "Passengers" are instances of a Java class "Passenger" that extends the standard OpenRules class ComparableDecisionVariable.

The proper Glossary is defined as follows:

Glossary glossary		
Variable	**Business Concept**	**Attribute**
Passengers	Problem	passengers
Passenger 1 Status	Passenger1	status
Passenger 1 Score		score
Passenger 1 Miles		miles
Passenger 2 Status	Passenger2	status
Passenger 2 Score		score
Passenger 2 Miles		miles

Thus, the array itself is a decision variable defined by the business concept "Problem". The glossary must include two business concepts "Passenger1" and "Passenger2", which names are formed by adding "1" and "2" to the type of the array elements. These business concepts should at a minimum include all decision variables that are being used by the comparison rules "ComparePassengers". You may find more complex example in the sample project "**FlightRebooking**" from the workspace “openrules.models”.

The sample project "**SortProducts**" from the same workspace demonstrates how to sort arrays of objects not inherited from ComparableDecisionVariable.

Along with arrays of different objects, you also can use lists of objects. You may see how it's done in the sample project SortPassengersList, in which instead of Passenger[] array we use List<Passengers>.

Defining and Using Rule Identification

You may associate a unique identificator (ID) with any rule inside a decision table and later on refer to this ID in the conclusion columns. To do that, you may use the column of the type "#" as the very first column of your decision table. If you put any text ID in front of the rule inside this column, then this ID will be assigned to this rule but only when it will be executed. Then you may to your rule IDs in the action columns like in the following example:

DecisionTable Swap			
#	If	Then	Message
Rule Id	**X**	**X**	**Message**
Rule 1	1	2	Executed rule <$RULE_ID>
Rule 2	2	1	

DECISION ANALYSIS

Decision Syntax Validation

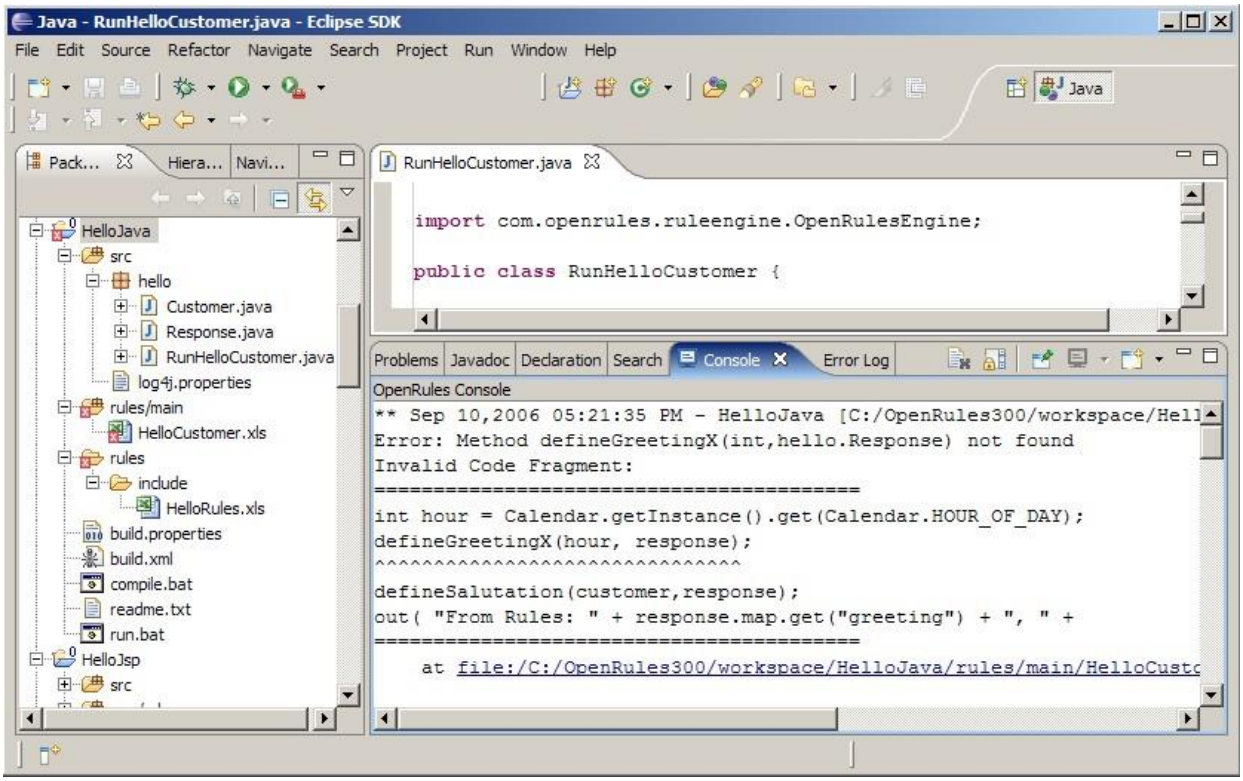
OpenRules® automatically validates your decision by checking that:

- there are no syntax error in the organization of all decision tables
- values inside decision variable cells correspond to the associated domains defined in the glossary (for Rule Solver only).

OpenRules® also provides a special plugin for Eclipse IDE, a de-facto standard project management tools for software developers within a Java-based development environment. Eclipse is used for code editing, debugging, and testing of rule projects within a single commonly known integrated development environment. OpenRules® has been designed to catch as many errors as possible in design-rime vs. run-time when it is too late.

Eclipse Plugin diagnoses any errors in Excel-files before you even deploy or run your OpenRules-based application. To make sure that Eclipse controls your OpenRules® project, you have first to right-click to your project folder and "Add OpenRules Nature". You always can similarly "Remove OpenRules Nature".

To be validated, your main xls-files should be placed into an Eclipse source folder while all included files should be kept in regular (non-source) folders. OpenRules® Plugin displays a diagnostic log with possible errors inside the Eclipse Console view. The error messages include hyperlinks that will open the proper Excel file with a cursor located in a cell where the error occurred. The following picture shows how OpenRules® Plugin automatically diagnoses errors in the Excel-files and displays the proper error messages:



Decision Testing

OpenRules® provides an ability to create a Test Harness comprised of an executable set of different test cases. It is important that the same people who

design rules (usually business analysts) can (and should!) design test cases for them. Usually they create test cases directly in Excel by specifying their own data/object types and creating instances of test objects of these types.

Test data can be defined in Excel using tables Datatype and Data as described above in the section Defining Test Data. Additionally, OpenRules® provides an **automatic comparison of expected and actual results** of the decision execution. You may define the expected execution results using special decision tables of the type "**DecisionTableTest**". Such tables may define your test cases containing:

1) Test objects for different business concepts defined in the Glossary
2) The expected results for several decision variables.

For example, the standard project "PatientTherapy" includes test Data objects "visits" and "patients":

Data DoctorVisit visits			
date	encounterDiagnosis	recommendedMedication	recommendedDose
Date	**Encounter Diagnosis**	**Recommended Medication**	**Recommended Dose**
2/15/2011	Acute Sinusitis	?	?
2/25/2011	Acute Sinusitis	?	?

Data Patient patients				
name	age	allergies	creatinineLevel	creatinineClearance
Name	**Age**	**Allergies**	**Creatinine Level**	**Creatinine Clearance**
John Smith	58	Penicillin Streptomycin	2.00	44.42
Mary Smith	65		1.80	48.03

To specify expected results for each of these two tests, we may add a decision table of the type **DecisionTableTest** that we will call "testCases":

DecisionTableTest testCases				
#	ActionUseObject	ActionUseObject	ActionExpect	ActionExpect
Test ID	**Visit**	**Patient**	**Patient Creatinine Clearance**	**Recommended Medication**
Test 1	:= visits[0]	:= patients[0]	44.42	Levofloxacin
Test 2	:= visits[1]	:= patients[1]	48.04	Amoxicillin

Here the first column specifies test IDs, next two columns specify test-objects defined in the above Data tables, and the last two columns specify the expected results.

In our Java launcher instead of putting test objects into the decision object and then calling decision.execute(), we may simply call

decision.test("testCases");

The Decision's method "test" will execute all tests from the Excel table "testCases" using business objects defined in the columns of the type "ActionUseObject". After the execution of each test-case, the produced results will be automatically compared with the expected results, which are defined in the columns of the type "ActionExpect". All mismatches will be reported in the execution log. For instance, the above example will produce the following records:

```
Validating results for the test <Test 1>
Test 1 was successful
Validating results for the test <Test 2>
MISMATCH: variable <Patient Creatinine Clearance> has value <48.03>
while <48.04> was expected
Test 2 was unsuccessful
1 test(s) out of 2 failed!
```

You may add any number of business objects and decision variables with expected results to your test cases. An example of the test harness can be found in the project "DecisionPatientTherapyTest" in the standard installationworkspace "openrules.dmn".

Decision Execution Reports

OpenRules® provides an ability to generate decision execution reports in the HTML-format. To generate an execution report, you should add the following setting to the decision's Java launcher:

```
decision.put("report", "On");
```

before calling `decision.execute()` or `decision.test("testCases")`. By default, execution reports are not generated as they are needed mainly for decision testing and analysis. Reports are regenerated for every decision's run.

During decision execution, OpenRules® automatically creates a sub-directory "report" in the main project directory and generates a report inside this sub-directory. For `decision.execute()` OpenRules® generates an html-file with the name `<DecisionTableName>.html`. For example, for the sample project "DecisionVacationDays" OpenRules® will generate this report:

Decision Table : Rule#	Executed Rule	Variables and Values
SetEligibleForExtra5Days:2	**IF** Age in Years >= 60 **THEN** Eligible for Extra 5 Days = true	Age in Years=64 Eligible for Extra 5 Days=true
SetEligibleForExtra3Days:1	**IF** Years of Service >= 30 **THEN** Eligible for Extra 3 Days = true	Years of Service=42 Eligible for Extra 3 Days=true
SetEligibleForExtra2Days:2	**IF** Age in Years >= 45 **THEN** Eligible for Extra 2 Days = true	Age in Years=64 Eligible for Extra 2 Days=true
CalculateVacationDays:1	Vacation Days = 22	Vacation Days=22
CalculateVacationDays:2	**IF** Eligible for Extra 5 Days = true **THEN** Vacation Days += 5	Eligible for Extra 5 Days=true Vacation Days=27
CalculateVacationDays:3	**IF** Eligible for Extra 3 Days = true **THEN** Vacation Days += 3	Eligible for Extra 3 Days=true Vacation Days=30

The report contains 3 columns:

- decision table name following the number of the executed rule
- the content of the executed rule (in a compact format that skips empty columns)
- current values of variables involved in this rule that helps to explain why this rule was executed.

Note. Execution reports are intended to explain the behavior of certain decision tables and are used mainly for analysis and not for production. If you turn on report generation mode in a multi-threaded environment that shares the same

instance of OpenRulesEngine, the reports will be produced only for the first thread.

Consistency Checking

OpenRules® provides a special component Rule Solver™ that along with powerful optimization features allow a user to check consistency of the decision models and find possible conflicts within decision tables and across multiple decision tables. The detail description of the product can be found at http://openrules.com/pdf/RulesSolver.UserManual.pdf.

Graphical Decision Model Analyzers

OpenRules® provides two special web-based analyzers for decision models created in accordance with the DMN standard:

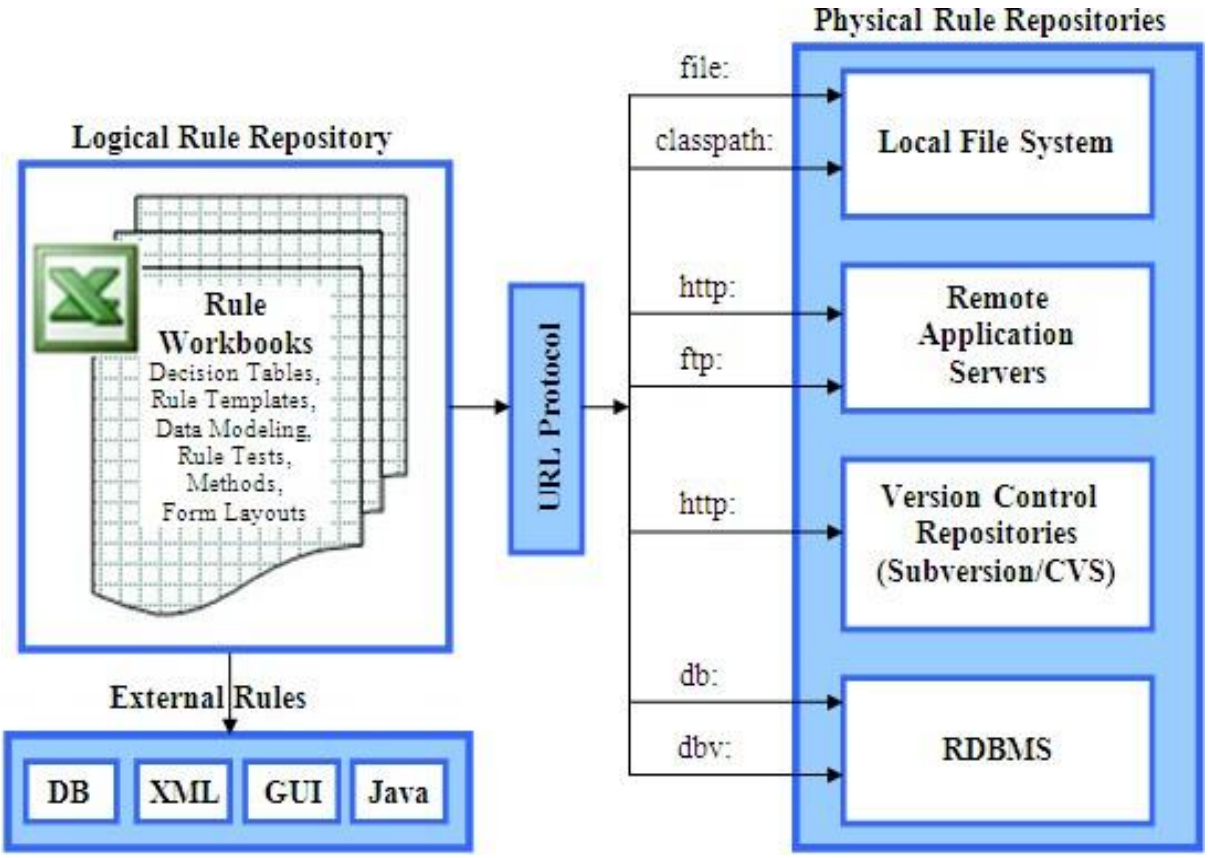
1) Decision Model Analyzer: provides a web interface to allow a business user to analyze and execute predefined and custom decision models. Along with the produced decisions, it shows all rules that were actually executed with values of the involved decision variable on the moment of execution.
2) WHAT-IF Analyzer: provides a web interface to allow a business user to activate/deactivate various rules. It immediately shows the changes in the domains of decision variables, finds one solution and navigates through multiple solutions, finds an optimal solution.

OpenRules® Knowledge Repository

OpenRules® knowledge repository usually consists of business rules and other source of business knowledge placed in regular Excel files. OpenRules® utilizes a popular spreadsheet mechanism and allows users to build enterprise-level repositories as hierarchies of inter-related xls-files. The OpenRules® Engine may access these rules files directly whether they are located in the local file system, on a remote server, in a standard version control system or in a relational database.

Logical and Physical Repositories

The following picture shows the logical organization of an OpenRules® repository and its possible physical implementations:



Logically, OpenRules® Repository may be considered as a hierarchy of rule workbooks. Each rule workbook is comprised of one or more worksheets that can be used to separate information by types or categories. Decision tables are the most typical OpenRules® tables and are used to represent business rules. Along with decision tables, OpenRules® supports tables of other types such as: Form Layouts, Data and Datatypes, Methods, and Environment tables.

Physically, all workbooks are saved in well-established formats, namely as standard xls- or xml-files. The proper Excel files may reside in the local file system, on remote application servers, in a version control system such as Subversion, or inside a standard database management system.

OpenRules® uses an URL pseudo-protocol notation with prefixes such as **"file:", "classpath:", "http://", "ftp://", "db:"**, etc.

Decision Templates

OpenRules® provides a set of standard templates implemented in xls-files located in the configuration folder "**openrules.config**" included in every standard workspace. These templates should be available for any custom project that uses OpenRules-based decision models.

Decision templates are based on the very powerful templatization mechanism provided by OpenRules for compactly organizing enterprise-level business rules repositories. You can find more information about the templates in the User Manual Advanced.

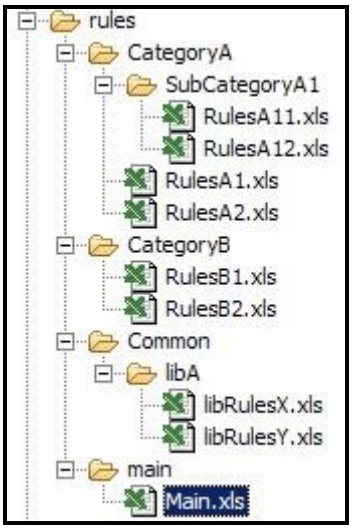
Hierarchies of Excel Workbooks

An OpenRules® repository usually consists of multiple Excel workbooks distributed between different subdirectories. Each workbook may include references to other workbooks thus comprising complex hierarchies of inter-related workbooks and decision tables.

Included Workbooks

Workbooks refer to other workbooks using so called "includes" inside the OpenRules® "**Environment**" tables. To let OpenRules® know about such include-relationships, you must place references to all included xls-files into the table "Environment".

Here is an example of an OpenRules® repository that comes with the standard sample project "RuleRepository" from "openrules.dmn":



The main xls-file "Main.xls" is located in the local directory "rules/main". To invoke any rules associated with this file, the proper Java program creates an OpenRulesEngine using a string "file:rules/main/Main.xls" as a parameter. There are many other xls-files related to the Main.xls and located in different subdirectories of "rules". Here is a fragment of the Main.xls "Environment" table:

include	../CategoryA/RulesA1.xls
	../CategoryA/RulesA2.xls
	../CategoryB/RulesB1.xls
	../CategoryB/RulesB2.xls
	../Common/libA/libRulesX.xls
	../Common/libA/libRulesY.xls

As you can guess, in this instance all included files are defined relative to the directory "rules/main" in which “Main.xls” resides. You may notice that files “RulesA11.xls” and “RulesA12.xls” are not included. The reason for this is that only “RulesA1.xls” really "cares" about these files. Naturally its own table "Environment" contains the proper "include":

Environment	
import.java	myjava.packA1.*
include	SubCategoryA1/RulesA11.xls
	SubCategoryA1/RulesA12.xls

Here, both "includes" are defined relative to the directory "CategoryA" of their "parent" file “RulesA1.xls”. As an alternative, you may define your included files relative to a so called "include.path" - see sample in the next section.

Include Path and Common Libraries of Workbooks

Includes provide a convenient mechanism to create libraries of frequently used xls-files and refer to them from different rule repositories. You can keep these libraries in a file system with a fixed "include.path". You may even decide to move such libraries with common xls-files from your local file system to a remote server. For instance, in our example above you could move a subdirectory "libA" with all xls-files to a new location with an http address http://localhost:8080/my.common.lib. In this case, you should first define

a so-called "include.path" and then refer to the xls-files relative to this include.path using angle brackets as shown below:

include.path	http://localhost:8080/my.common.lib/
include	<libA/libRulesX.xls>
	<libA/libRulesX.xls>

Here we want to summarize the following important points:

- The structure of your rule repository can be presented naturally inside xls-files themselves using "includes"
- The rule repository can include files from different physical locations
- Complex branches on the rules tree can encapsulate knowledge about their own organization.

Using Regular Expressions in the Names of Included Files

Large rule repositories may contain many files (workbooks) and it is not convenient to list all of them by name. In this case you may use regular expression inside included file names within the Environment table. For example, consider in the following Environment table:

Environment	
include	../category1/*.xls
include	../category2/XYZ*.xls
include	../category3/A?.xls

The first line will include all files with an extension “xls” from the folder “category1”. The second line will include all files with an extension “xls” and which names start with “XYZ” from the folder “category2”. The third line will include all files with an extension “xls” that start with a letter “A” following exactly one character from the folder “category1”.

Actually along with wildcard characters “*” or “?” you may use any standard regular expressions to define the entire path to different workbooks.

Parameterized Rule Repositories

An OpenRules® repository may be parameterized in such a way that different rule workbooks may be invoked from the same repository under different circumstances. For example, let's assume that we want to define rules that offer different travel packages for different years and seasons. We may specify a concrete year and a season by using environment variables YEAR and SEASON. Our rules repository may have the following structure:

```
rules/main/Main.xls

rules/common/CommonRules.xls

rules/2007/SummerRules.xls

rules/2007/WinterRules.xls

rules/2008/SummerRules.xls

rules/2008/WinterRules.xls
```

To make the OpenRulesEngine automatically select the correct rules from such a repository, we may use the following parameterized include-statements inside the Environment table of the main xls-file rules/main/Main.xls:

Environment	
import.java	season.offers.*
include	../common/SalutationRules.xls
include	../${YEAR}/${SEASON}Rules.xls

Thus, the same rules repository will handle both WinterRules and SummerRules for different years. A detailed example is provided in the standard project SeasonRules.

Decision Import and Loosely Coupled Decision Models

In practice “big” decision models can be decomposes into smaller, loosely coupled decision models, which can be implemented and executed independently. Then a “big” decision model can simply “import” these “small” decision model using a

special OpenRules table of the type "DecisionImport". Here is an example of such a table:

DecisionImport decisionImports	
Imported Decision File	**Internal Decision Name**
file:repository/Decisions/ApplicationRiskScore/Decision.xls	DetermineApplicationRiskScore
file:repository/Decisions/PreBureauRiskCategory/Decision.xls	DeterminePreBureauRiskCategory
file:repository/Decisions/Affordability/Decision.xls	DetermineAffordability
file:repository/Decisions/BureauStrategy/Decision.xls	DetermineBureauStrategy
file:repository/Decisions/PostBureauRiskCategory/Decision.xls	DeterminePostBureauRiskCategory
file:repository/Decisions/Routing/Decision.xls	DetermineRouting
file:repository/Decisions/LoanOriginationResult/Decision.xls	DetermineLoanOriginationResult

The first column refers to the full path of the main file for every decision model we want to import into the big decision model. The second column contains to the names of the imported decision models, under which they are known to the "big" decision.

Each imported decision model will use a separate instance of OpenRules Engine, so all imported models are independent and may even use the same names for decision tables implemented differently.

The detailed description of decomposition of big decision models into small loosely coupled decision models and their consecutive integration (import) can be found here. The proper working examples:

- DecisionLoanOrigination in the workspace "openrules.dmn"
- LoanOrigination in the workspace "openrules.models", see "LoanOriginationResult".

RULES AUTHORING AND MAINTENANCE TOOLS

OpenRules® relies on commonly used tools to create and manage its knowledge repository.

Rules Editors

To create and edit rules and other tables presented in Excel-files you may use any standard spreadsheet editors such as:

- MS Excel™
- OpenOffice™
- Google Sheets™

Google Sheets are especially useful for sharing spreadsheet editing - see section Collaborative Rules Management with Google Docs.

Rules Version Control

For rules version control you can choose any standard version control system that works within your traditional software development environment. We would recommend using an open source products such as “Git” or "Subversion". One obvious advantage of the suggested approach is the fact that both business rules and related Java/XML files will be handled by the same version control system.

You may even keep your Excel files with rules, data and other OpenRules® tables directly in your version control system. If your include-statements use http-addresses that point to a concrete repository then the OpenRulesEngine will dynamically access these repositories without the necessity to move Excel files back into a file system.

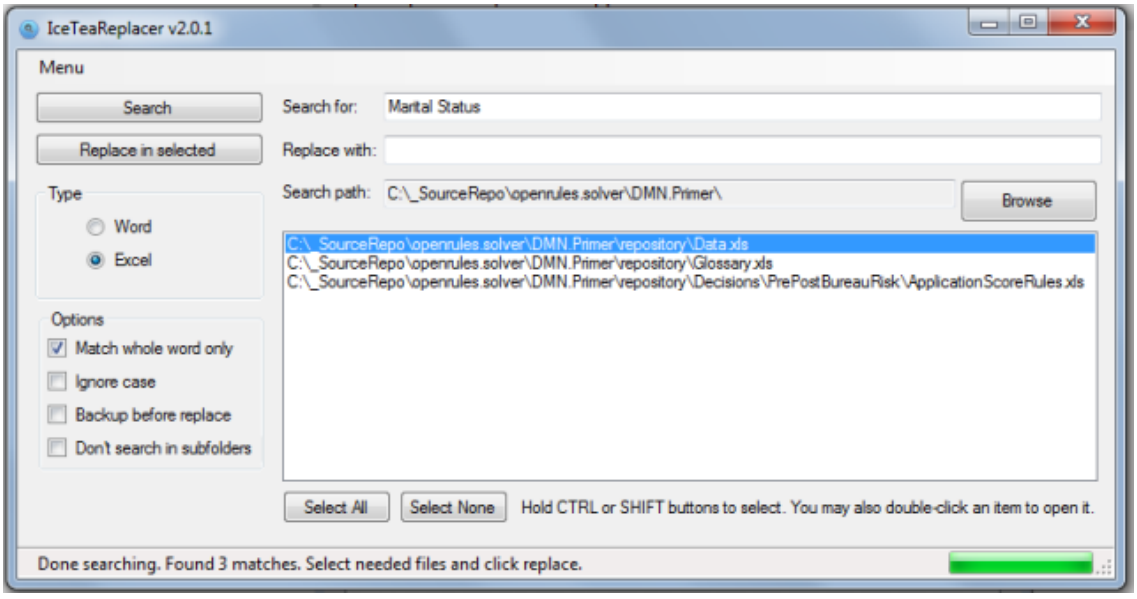
Another possible way to use version control is to place your rule workbooks in a database and use DBV-protocol to access different versions of the rules in run-time - read more.

OpenRules® relies on standard commonly used tools (mainly from Open Source) to organize and manage a Business Rules Repository.

Rules Search

To analyze rules within one Excel files you may effectively use familiar Search and Replace features provided by Excel or Google Docs.

When you want to search across multiple Excel files and folders, you may use a free while powerful tool called "IceTeaReplacer" that can be very useful for doing search & replace in OpenRules repositories. Here is an example of its graphical interface:



The following options are available:

- Perform search before replacing
- Match whole word only
- Ignore word case
- Do backup before replace
- Deselect files on which you don't want to perform replace.

OPENRULES® PROJECTS

Pre-Requisites

OpenRules® requires the following software:

- Java Standard Edition JDK 1.6 or higher
- Apache Ant
- MS Excel or OpenOffice or Google Sheets (for rules and forms editing only)
- Eclipse SDK (optional, for complex project management and IT integration)

Sample Projects

The complete OpenRules® installation includes the following workspaces:

openrules.models - decision models using OpenRules-7

openrules.dmn - decision models using OpenRules-6

openrules.decisions – old but still working decision projects

openrules.rules - various rules projects

openrules.dialog – rules-based web questionnaires

openrules.web - rules-based web applications & web services

openrules.solver - constraint-based decisions with Rule Solver

openrules.analyzers – web-based analysers

openrules.cloud - cloud-based applications.

Each project has its own subdirectory, e.g. "Hello". OpenRules® libraries and related templates are in the main configuration project, “openrules.config”, included in each workspace.

Main Configuration Project

OpenRules® provides a set of libraries (jar-files) and Excel-based templates in the folder "`openrules.config`" to support different projects.

Supporting Libraries

All OpenRules® jar-files are included in the folder, "`openrules.config/lib`". For the decision management projects you need at least the following jars:

- `commons-logging-1.1.jar`
- `log4j-1.2.15.jar`
- `commons-lang-2.3.jar`
- `poi-3.10-FINAL-20140208.jar`
- `poi-ooxml-3.10-FINAL-20140208.jar`
- `poi-ooxml-schemas-3.10-FINAL-20140208.jar`
- `dom4j-1.6.1.jar`
- `xmlbeans-2.3.0.jar`

There is a supporting library

- `com.openrules.tools.jar`

contains the following optional facilities:

- operators described in the Java class `Operator` that can be used inside your own Rules tables and templates
- convenience methods like "`out(String text)`" described in the Java class `Methods`
- simple JDBC interfaces `DbUtil, Database, DatabaseIterator`
- text validation methods like "`isCreditCardValid(String text)`" described in the Java class `Validator`
- XML reader.

If you use the JSR-94 interface you will also need

- `com.openrules.jsr94.jar`

If you use external rules from a database you will also need

- `openrules.db.jar`
- `openrules.dbv.jar`
- `derby.jar`
- `commons-cli-1.1.jar.`

Different workspaces like "`openrules.models`", "`openrules.dmn`", etc. include the proper versions of the folder "`openrules.config`".

Predefined Types and Templates

The Excel-based templates that support Decisions and Decision Tables included in the folder, "openrules.config":

- `DecisionTemplates.xls`
- `DecisionTableExecuteTemplates.xls`

There is one more file "`DecisionTemplatesDeprecated.xls`" that was used in old OpenRules releases. If you still use the deprecated methods defined in this file (not recommended), you may simply remove an empty line in the Environment table inside the file "`DecisionTemplates.xls`"

Sample decision projects include Excel tables of the type "Environment" that usually refer to "`../../../openrules.config/DecisionTemplates.xls`". You may move all templates to another location and simply modify this reference making it relative to your main xls-file.

TECHNICAL SUPPORT

Direct all your technical questions to support@openrules.com or to this Discussion Group. Read more at http://openrules.com/services.htm.