Red Hat Decision Manager 7.3

Designing a decision service using spreadsheet decision tables

Guide

Red Hat Decision Manager 7.3 Designing a decision service using spreadsheet decision tables

Guide

Enter your first name here. Enter your surname here.
Enter your organisation's name here. Enter your organisational division here.
Enter your email address here.

法律上の通知

概要

This document describes how to design a decision service using spreadsheet decision tables in Red Hat Decision Manager 7.3.

目次

PREFACE

As a business analyst or business rules developer, you can define business rules in a tabular format in spreadsheet decision tables and then upload the spreadsheets to your project in Business Central. These rules are compiled into Drools Rule Language (DRL) and form the core of the decision service for your project.

Prerequisite

The team and project for the decision tables have been created in Business Central. Each asset is associated with a project assigned to a team. For details, see Getting started with decision services.

第1章 DECISION-AUTHORING ASSETS IN RED HAT DECISION MANAGER

Red Hat Decision Manager supports several assets that you can use to define business decisions for your decision service. Each decision-authoring asset has different advantages, and you might prefer to use one or a combination of multiple assets depending on your goals and needs.

The following table highlights the main decision-authoring assets supported in Red Hat Decision Manager projects to help you decide or confirm the best method for defining decisions in your decision service.

表1.1 Decision-authoring assets supported in Red Hat Decision Manager

Asset	Highlights	Authoring tools	Documentation
Decision Model and Notation (DMN) models	- Are decision models based on a notation standard defined by the Object Management Group (OMG) - Use graphical decision requirements diagrams (DRDs) with one or more decision requirements graphs (DRGs) to trace business decision flows - Use an XML schema that allows the DMN models to be shared between DMN-compliant platforms - Support Friendly Enough Expression Language (FEEL) to define decision logic in DMN decision tables and other DMN boxed expressions - Are optimal for creating comprehensive, illustrative, and stable decision flows	Business Central or other DMN-compliant editor	Designing a decision service using DMN models

Asset	Highlights	Authoring tools	Documentation
Guided decision tables	- Are tables of rules that you create in a UI-based table designer in Business Central - Are a wizard-led alternative to spreadsheet decision tables - Provide fields and options for acceptable input - Support template keys and values for creating rule templates - Support hit policies, real-time validation, and other additional features not supported in other assets - Are optimal for creating rules in a controlled tabular format to minimize compilation errors	Business Central	Designing a decision service using guided decision tables
Spreadsheet decision tables	- Are XLS or XLSX spreadsheet decision tables that you can upload into Business Central - Support template keys and values for creating rule templates - Are optimal for creating rules in decision tables already managed outside of Business Central - Have strict syntax requirements for rules to be compiled properly when uploaded	Spreadsheet editor	Designing a decision service using spreadsheet decision tables
Guided rules	- Are individual rules that you create in a UI-based rule designer in Business Central - Provide fields and options for acceptable input - Are optimal for creating single rules in a controlled format to minimize compilation errors	Business Central	Designing a decision service using guided rules

Asset	Highlights	Authoring tools	Documentation
Guided rule templates	- Are reusable rule structures that you create in a UI-based template designer in Business Central - Provide fields and options for acceptable input - Support template keys and values for creating rule templates (fundamental to the purpose of this asset) - Are optimal for creating many rules with the same rule structure but with different defined field values	Business Central	Designing a decision service using guided rule templates
DRL rules	- Are individual rules that you define directly in **.drl** text files - Provide the most flexibility for defining rules and other technicalities of rule behavior - Can be created in certain standalone environments and integrated with Red Hat Decision Manager - Are optimal for creating rules that require advanced DRL options - Have strict syntax requirements for rules to be compiled properly	Business Central or integrated development environment (IDE)	Designing a decision service using DRL rules

Asset	Highlights	Authoring tools	Documentation
Predictive Model Markup Language (PMML) models	• Are predictive data-analytic models based on a notation standard defined by the Data Mining Group (DMG) • Use an XML schema that allows the PMML models to be shared between PMML-compliant platforms • Support Regression, Scorecard, Tree, Mining, and other model types • Can be included with a standalone Red Hat Decision Manager project or imported into a project in Business Central • Are optimal for incorporating predictive data into decision services in Red Hat Decision Manager	PMML or XML editor	Designing a decision service using PMML models

第2章 SPREADSHEET DECISION TABLES

Spreadsheet decision tables are XLS or XLSX spreadsheets that contain business rules defined in a tabular format. You can include spreadsheet decision tables with standalone Red Hat Decision Manager projects or upload them to projects in Business Central. Each row in a decision table is a rule, and each column is a condition, an action, or another rule attribute. After you create and upload your spreadsheet decision tables, the rules you defined are compiled into Drools Rule Language (DRL) rules as with all other rule assets.

All data objects related to a spreadsheet decision table must be in the same project package as the spreadsheet decision table. Assets in the same package are imported by default. Existing assets in other packages can be imported with the decision table.

第3章 DATA OBJECTS

Data objects are the building blocks for the rule assets that you create. Data objects are custom data types implemented as Java objects in specified packages of your project. For example, you might create a **Person** object with data fields **Name**, **Address**, and **DateOfBirth** to specify personal details for loan application rules. These custom data types determine what data your assets and your decision services are based on.

3.1. CREATING DATA OBJECTS

The following procedure is a generic overview of creating data objects. It is not specific to a particular business asset.

Procedure

1. In Business Central, go to **Menu → Design → Projects** and click the project name.

2. Click **Add Asset → Data Object**.

3. Enter a unique **Data Object** name and select the **Package** where you want the data object to be available for other rule assets. Data objects with the same name cannot exist in the same package. In the specified DRL file, you can import a data object from any package.
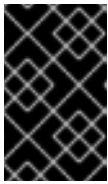
   IMPORTING DATA OBJECTS FROM OTHER PACKAGES

   You can import an existing data object from another package directly into the asset designer. Select the relevant rule asset within the project and in the asset designer, go to **Data Objects → New item** to select the object to be imported.

4. To make your data object persistable, select the **Persistable** checkbox. Persistable data objects are able to be stored in a database according to the JPA specification. The default JPA is Hibernate.

5. Click **Ok**.

6. In the data object designer, click **add field** to add a field to the object with the attributes **Id**, **Label**, and **Type**. Required attributes are marked with an asterisk (*).

   - **Id:** Enter the unique ID of the field.

   - **Label:** (Optional) Enter a label for the field.

   - **Type:** Enter the data type of the field.

   - **List:** Select this check box to enable the field to hold multiple items for the specified type.
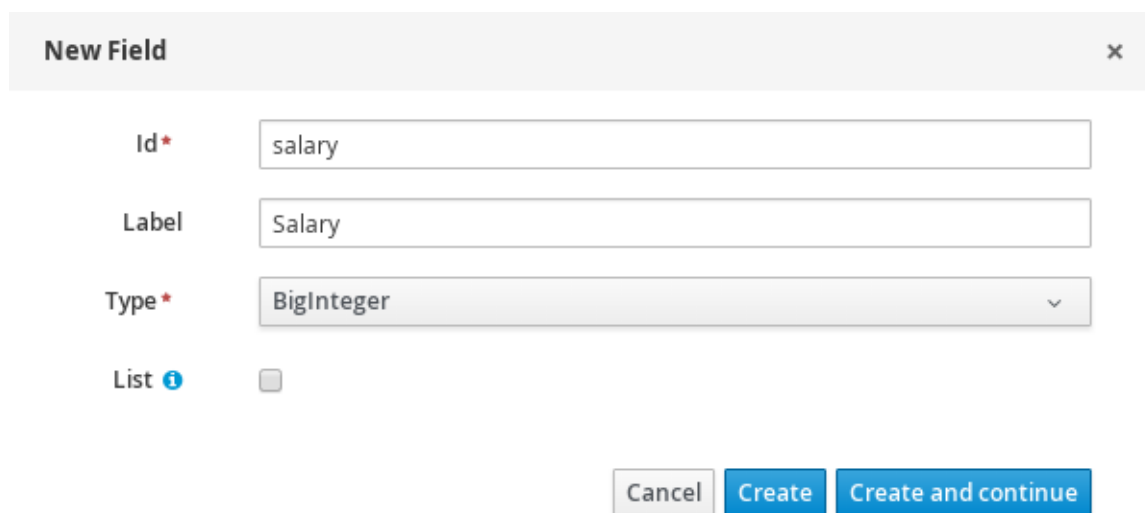
図3.1 Add data fields to a data object

New Field   ✕

Id*   salary

Label   Salary

Type*   BigInteger   ⌄

List ⓘ   ☐

                                    Cancel   Create   Create and continue

7. Click **Create** to add the new field, or click **Create and continue** to add the new field and continue adding other fields.
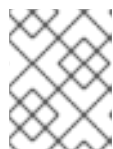
> **注記**
>
> To edit a field, select the field row and use the **general properties** on the right side of the screen.

第4章 DECISION TABLE USE CASE

An online shopping site lists the shipping charges for ordered items. The site provides free shipping under the following conditions:

- The number of items ordered is 4 or more and the checkout total is $300 or more.

- Standard shipping is selected (4 or 5 business days from the date of purchase).

The following are the shipping rates under these conditions:

表4.1 For orders less than $300

Number of items	Delivery day	Shipping charge in USD, N = Number of items
3 or fewer	Next day	35
	2nd day	15
	Standard	10
4 or more	Next day	N*7.50
	2nd day	N*3.50
	Standard	N*2.50

表4.2 For orders more than $300

Number of items	Delivery day	Shipping charge in USD, N = Number of items
3 or fewer	Next day	25
	2nd day	10
	Standard	N*1.50
4 or more	Next day	N*5
	2nd day	N*2
	Standard	FREE

These conditions and rates are shown in the following example spreadsheet decision table:

図4.1 Decision table for shipping charges

	A	B	C	D	E	F
1		RuleSet	Charge Calculator			
2		Import	guvnor.feature.dtables.Order, guvnor.feature.dtables.Charge			
3		Variables	Integer totalCount			
4		Sequential	TRUE			
5		SequentialMaxPriority	10			
6						
7		RuleTable Basic				
8	DESCRIPTION	CONDITION	CONDITION	CONDITION	ACTION	
9		$order : Order				
10		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	insert(new Charge($1));	
11		Min items	Max items	Delivered in days	Pay charge in US dollars	
12	expensive	0	3	1	35	
13		0	3	2	15	
14		0	3		10	
15		4		1	$order.getItemsCount() * 7.5	
16		4		2	$order.getItemsCount() * 3.5	
17	cheap	4			$order.getItemsCount() * 2.5	
18						
19		RuleTable Expensive				
20	DESCRIPTION	CONDITION	CONDITION	CONDITION	CONDITION	ACTION
21		$order : Order				
22		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	totalPrice > $1	insert(new Charge($1));
23		Min items	Max items	Delivered in days	More expensive than	Pay charge in US dollars
24	expensive	0	3	1	300	25
25		0	3	2	300	10
26		0	3		300	$order.getItemsCount() * 1.5
27		4		1	300	$order.getItemsCount() * 5
28		4		2	300	$order.getItemsCount() * 2
29	cheap	4			300	0
30						

In order for a decision table to be uploaded in Business Central, the table must comply with certain structure and syntax requirements, within an XLS or XLSX spreadsheet, as shown in this example. For more information, see 5章*Defining spreadsheet decision tables* .

第5章 DEFINING SPREADSHEET DECISION TABLES

Spreadsheet decision tables (XLS or XLSX) require two key areas that define rule data: a **RuleSet** area and a **RuleTable** area. The **RuleSet** area of the spreadsheet defines elements that you want to apply globally to all rules in the same package (not only the spreadsheet), such as a rule set name or universal rule attributes. The **RuleTable** area defines the actual rules (rows) and the conditions, actions, and other rule attributes (columns) that constitute that rule table within the specified rule set. A spreadsheet of decision tables can contain multiple **RuleTable** areas, but only one **RuleSet** area.
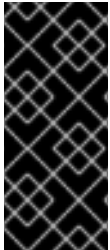
> **重要**
>
> You should typically upload only one spreadsheet of decision tables, containing all necessary **RuleTable** definitions, per rule package in Business Central. You can upload separate decision table spreadsheets for separate packages, but uploading multiple spreadsheets in the same package can cause compilation errors from conflicting **RuleSet** or **RuleTable** attributes and is therefore not recommended.

Refer to the following sample spreadsheet as you define your decision table:

図5.1 Sample spreadsheet decision table for shipping charges

	A	B	C	D	E	F
1		RuleSet	Charge Calculator			
2		Import	guvnor.feature.dtables.Order, guvnor.feature.dtables.Charge			
3		Variables	Integer totalCount			
4		Sequential	TRUE			
5		SequentialMaxPriority	10			
6						
7		RuleTable Basic				
8	DESCRIPTION	CONDITION	CONDITION	CONDITION	ACTION	
9		$order : Order				
10		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	insert(new Charge($1));	
11		Min items	Max items	Delivered in days	Pay charge in US dollars	
12	expensive	0	3	1	35	
13		0	3	2	15	
14		0	3		10	
15		4		1	$order.getItemsCount() * 7.5	
16		4		2	$order.getItemsCount() * 3.5	
17	cheap	4			$order.getItemsCount() * 2.5	
18						
19		RuleTable Expensive				
20	DESCRIPTION	CONDITION	CONDITION	CONDITION	CONDITION	ACTION
21		$order : Order				
22		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	totalPrice > $1	insert(new Charge($1));
23		Min items	Max items	Delivered in days	More expensive than	Pay charge in US dollars
24	expensive	0	3	1	300	25
25		0	3	2	300	10
26		0	3		300	$order.getItemsCount() * 1.5
27		4		1	300	$order.getItemsCount() * 5
28		4		2	300	$order.getItemsCount() * 2
29	cheap	4			300	0
30						

Procedure

1. In a new XLS or XLSX spreadsheet, go to the second or third column and label a cell **RuleSet** (row 1 in example). Reserve the column or columns to the left for descriptive metadata (optional).

2. In the next cell to the right, enter a name for the **RuleSet**. This named rule set will contain all **RuleTable** rules defined in the rule package.

3. Under the **RuleSet** cell, define any rule attributes (one per cell) that you want to apply globally to all rule tables in the package. Specify attribute values in the cells to the right. For example, you can enter an **Import** label and in the cell to the right, specify relevant data objects from other packages that you want to import into the package for the decision table (in the format **package.name.object.name**). For supported cell labels and values, see 「RuleSet definitions」.

4. Below the **RuleSet** area and in the same column as the **RuleSet** cell, skip a row and label a new cell **RuleTable** (row 7 in example) and enter a table name in the same cell. The name is used as the initial part of the name for all rules derived from this rule table, with the row number appended for distinction. You can override this automatic naming by inserting a **NAME** attribute column.

5. Use the next four rows to define the following elements as needed (rows 8-11 in example):

   - **Rule attributes:** Conditions, actions, or other attributes. For supported cell labels and values, see 「RuleTable definitions」.

   - **Object types:** The data objects to which the rule attributes apply. If the same object type applies to multiple columns, merge the object cells into one cell across multiple columns (as shown in the sample decision table), instead of repeating the object type in multiple cells. When an object type is merged, all columns below the merged range will be combined into one set of constraints within a single pattern for matching a single fact at a time. When an object is repeated in separate columns, the separate columns can create different patterns, potentially matching different or identical facts.

   - **Constraints:** Constraints on the object types.

   - **Column label:** (Optional) Any descriptive label for the column, as a visual aid. Leave blank if unused.

   > **注記**
   >
   > As an alternative to populating both the object type and constraint cells, you can leave the object type cell or cells empty and enter the full expression in the corresponding constraint cell or cells. For example, instead of **Order** as the object type and **itemsCount > $1** as a constraint (separate cells), you can leave the object type cell empty and enter **Order(itemsCount > $1)** in the constraint cell, and then do the same for other constraint cells.

6. After you have defined all necessary rule attributes (columns), enter values for each column as needed, row by row, to generate rules (rows 12-17 in example). Cells with no data are ignored (such as when a condition or action does not apply).
   If you need to add more rule tables to this decision table spreadsheet, skip a row after the last rule in the previous table, label another **RuleTable** cell in the same column as the previous **RuleTable** and **RuleSet** cells, and create the new table following the same steps in this section (rows 19-29 in example).

7. Save your XLS or XLSX spreadsheet to finish.

注記

Only the first worksheet in a spreadsheet workbook will be processed as a decision table when you upload the spreadsheet in Business Central. Each **RuleSet** name combined with the **RuleTable** name must be unique across all decision table files in the same package.

After you upload the decision table in Business Central, the rules are rendered as DRL rules like the following example, from the sample spreadsheet:

```
//row 12
rule "Basic_12"
salience 10
  when
    $order : Order( itemsCount > 0, itemsCount <= 3, deliverInDays == 1 )
  then
    insert( new Charge( 35 ) );
end
```

ENABLING WHITE SPACE USED IN CELL VALUES

By default, any white space before or after values in decision table cells is removed before the decision table is processed by the decision engine. To retain white space that you use intentionally before or after values in cells, set the **drools.trimCellsInDTable** system property to **false** in your Red Hat Decision Manager distribution.

For example, if you use Red Hat Decision Manager with Red Hat JBoss EAP, add the following system property to your **$EAP_HOME/standalone/configuration/standalone-full.xml** file:

```
<property name="drools.trimCellsInDTable" value="false"/>
```

If you use the decision engine embedded in your Java application, add the system property with the following command:

```
java -jar yourApplication.jar -Ddrools.trimCellsInDTable=false
```

5.1. RULESET DEFINITIONS

Entries in the **RuleSet** area of a decision table define DRL constructs and rule attributes that you want to apply to all rules in a package (not only in the spreadsheet). Entries must be in a vertically stacked sequence of cell pairs, where the first cell contains a label and the cell to the right contains the value. A decision table spreadsheet can have only one **RuleSet** area.

The following table lists the supported labels and values for **RuleSet** definitions:

表5.1 Supported **RuleSet** definitions

Label	Value	Usage

Label	Value	Usage
RuleSet	The package name for the generated DRL file. Optional, the default is **rule_table**.	Must be the first entry.
Sequential	**true** or **false**. If **true**, then salience is used to ensure that rules fire from the top down.	Optional, at most once. If omitted, no firing order is imposed.
SequentialMaxPriority	Integer numeric value	Optional, at most once. In sequential mode, this option is used to set the start value of the salience. If omitted, the default value is 65535.
SequentialMinPriority	Integer numeric value	Optional, at most once. In sequential mode, this option is used to check if this minimum salience value is not violated. If omitted, the default value is 0.
EscapeQuotes	**true** or **false**. If **true**, then quotation marks are escaped so that they appear literally in the DRL.	Optional, at most once. If omitted, quotation marks are escaped.
Import	A comma-separated list of Java classes to import from another package.	Optional, may be used repeatedly.
Variables	Declarations of DRL globals (a type followed by a variable name). Multiple global definitions must be separated by commas.	Optional, may be used repeatedly.
Functions	One or more function definitions, according to DRL syntax.	Optional, may be used repeatedly.
Queries	One or more query definitions, according to DRL syntax.	Optional, may be used repeatedly.
Declare	One or more declarative types, according to DRL syntax.	Optional, may be used repeatedly.

> ⚠️ **警告**
>
> In some cases, Microsoft Office, LibreOffice, and OpenOffice might encode a double quotation mark differently, causing a compilation error. For example, 「A」 will fail, but **"A"** will pass.

5.2. RULETABLE DEFINITIONS

Entries in the **RuleTable** area of a decision table define conditions, actions, and other rule attributes for the rules in that rule table. A spreadsheet of decision tables can contain multiple **RuleTable** areas.

The following table lists the supported labels (column headers) and values for **RuleTable** definitions. For column headers, you can use either the given labels or any custom labels that begin with the letters listed in the table.

表5.2 Supported **RuleTable** definitions

Label	Or custom label that begins with	Value	Usage
NAME	N	Provides the name for the rule generated from that row. The default is constructed from the text following the **RuleTable** tag and the row number.	At most one column.
DESCRIPTION	I	Results in a comment within the generated rule.	At most one column.
CONDITION	C	Code snippet and interpolated values for constructing a constraint within a pattern in a condition.	At least one per rule table.
ACTION	A	Code snippet and interpolated values for constructing an action for the consequence of the rule.	At least one per rule table.
METADATA	@	Code snippet and interpolated values for constructing a metadata entry for the rule.	Optional, any number of columns.

The following sections provide more details about how condition, action, and metadata columns use cell data:

Conditions

For columns headed **CONDITION**, the cells in consecutive lines result in a conditional element:

- **First cell:** Text in the first cell below **CONDITION** develops into a pattern for the rule

condition, and uses the snippet in the next line as a constraint. If the cell is merged with one or more neighboring cells, a single pattern with multiple constraints is formed. All constraints are combined into a parenthesized list and appended to the text in this cell.
If this cell is empty, the code snippet in the cell below it must result in a valid conditional element on its own. For example, instead of **Order** as the object type and **itemsCount > $1** as a constraint (separate cells), you can leave the object type cell empty and enter **Order(itemsCount > $1)** in the constraint cell, and then do the same for any other constraint cells.

To include a pattern without constraints, you can write the pattern in front of the text of another pattern, with or without an empty pair of parentheses. You can also append a **from** clause to the pattern.

If the pattern ends with **eval**, code snippets produce boolean expressions for inclusion into a pair of parentheses after **eval**.

- **Second cell:** Text in the second cell below **CONDITION** is processed as a constraint on the object reference in the first cell. The code snippet in this cell is modified by interpolating values from cells farther down in the column. If you want to create a constraint consisting of a comparison using **==** with the value from the cells below, then the field selector alone is sufficient. Any other comparison operator must be specified as the last item within the snippet, and the value from the cells below is appended. For all other constraint forms, you must mark the position for including the contents of a cell with the symbol **$param**. Multiple insertions are possible if you use the symbols **$1**, **$2**, and so on, and a comma-separated list of values in the cells below. However, do not separate **$1**, **$2**, and so on, by commas, or the table will fail to process.
  To expand a text according to the pattern **forall($delimiter){$snippet}**, repeat the **$snippet** once for each of the values of the comma-separated list in each of the cells below, insert the value in place of the symbol **$**, and join these expansions by the given **$delimiter**. Note that the **forall** construct may be surrounded by other text.

  If the first cell contains an object, the completed code snippet is added to the conditional element from that cell. A pair of parentheses is provided automatically, as well as a separating comma if multiple constraints are added to a pattern in a merged cell. If the first cell is empty, the code snippet in this cell must result in a valid conditional element on its own. For example, instead of **Order** as the object type and **itemsCount > $1** as a constraint (separate cells), you can leave the object type cell empty and enter **Order(itemsCount > $1)** in the constraint cell, and then do the same for any other constraint cells.

- **Third cell:** Text in the third cell below **CONDITION** is a descriptive label that you define for the column, as a visual aid.

- **Fourth cell:** From the fourth row on, non-blank entries provide data for interpolation. A blank cell omits the condition or constraint for this rule.

Actions

For columns headed **ACTION**, the cells in consecutive lines result in an action statement:

- **First cell:** Text in the first cell below **ACTION** is optional. If present, the text is interpreted as an object reference.

- **Second cell:** Text in the second cell below **ACTION** is a code snippet that is modified by interpolating values from cells farther down in the column. For a singular insertion, mark the position for including the contents of a cell with the symbol **$param**. Multiple insertions are possible if you use the symbols **$1**, **$2**, and so on, and a comma-separated list of values in the cells below. However, do not separate **$1**, **$2**, and so on, by commas, or the table will fail to process.

A text without any marker symbols can execute a method call without interpolation. In this case, use any non-blank entry in a row below the cell to include the statement. The **forall** construct is supported.

If the first cell contains an object, then the cell text (followed by a period), the text in the second cell, and a terminating semicolon are strung together, resulting in a method call that is added as an action statement for the consequence. If the first cell is empty, the code snippet in this cell must result in a valid action element on its own.

- **Third cell:** Text in the third cell below **ACTION** is a descriptive label that you define for the column, as a visual aid.

- **Fourth cell:** From the fourth row on, non-blank entries provide data for interpolation. A blank cell omits the condition or constraint for this rule.

Metadata

For columns headed **METADATA**, the cells in consecutive lines result in a metadata annotation for the generated rules:

- **First cell:** Text in the first cell below **METADATA** is ignored.

- **Second cell:** Text in the second cell below **METADATA** is subject to interpolation, using values from the cells in the rule rows. The metadata marker character @ is prefixed automatically, so you do not need to include that character in the text for this cell.

- **Third cell:** Text in the third cell below **METADATA** is a descriptive label that you define for the column, as a visual aid.

- **Fourth cell:** From the fourth row on, non-blank entries provide data for interpolation. A blank cell results in the omission of the metadata annotation for this rule.

5.3. ADDITIONAL RULE ATTRIBUTES FOR RULESET OR RULETABLE DEFINITIONS

The **RuleSet** and **RuleTable** areas also support labels and values for other rule attributes, such as **PRIORITY** or **NO-LOOP**. Rule attributes specified in a **RuleSet** area will affect all rule assets in the same package (not only in the spreadsheet). Rule attributes specified in a **RuleTable** area will affect only the rules in that rule table. You can use each rule attribute only once in a **RuleSet** area and once in a **RuleTable** area. If the same attribute is used in both **RuleSet** and **RuleTable** areas within the spreadsheet, then **RuleTable** takes priority and the attribute in the **RuleSet** area is overridden.

The following table lists the supported labels (column headers) and values for additional **RuleSet** or **RuleTable** definitions. For column headers, you can use either the given labels or any custom labels that begin with the letters listed in the table.

表5.3 Additional rule attributes for**RuleSet** or **RuleTable** definitions

Label	Or custom label that begins with	Value

Label	Or custom label that begins with	Value
PRIORITY	P	An integer defining the **salience** value of the rule. Rules with a higher salience value are given higher priority when ordered in the activation queue. Overridden by the **Sequential** flag. Example: **PRIORITY 10**
DATE-EFFECTIVE	V	A string containing a date and time definition. The rule can be activated only if the current date and time is after a **DATE-EFFECTIVE** attribute. Example: **DATE-EFFECTIVE "4-Sep-2018"**
DATE-EXPIRES	Z	A string containing a date and time definition. The rule cannot be activated if the current date and time is after the **DATE-EXPIRES** attribute. Example: **DATE-EXPIRES "4-Oct-2018"**
NO-LOOP	U	A Boolean value. When this option is set to **true**, the rule cannot be reactivated (looped) if a consequence of the rule re-triggers a previously met condition. Example: **NO-LOOP true**
AGENDA-GROUP	G	A string identifying an agenda group to which you want to assign the rule. Agenda groups allow you to partition the agenda to provide more execution control over groups of rules. Only rules in an agenda group that has acquired a focus are able to be activated. Example: **AGENDA-GROUP "GroupName"**
ACTIVATION-GROUP	X	A string identifying an activation (or XOR) group to which you want to assign the rule. In activation groups, only one rule can be activated. The first rule to fire will cancel all pending activations of all rules in the activation group. Example: **ACTIVATION-GROUP "GroupName"**
DURATION	D	A long integer value defining the duration of time in milliseconds after which the rule can be activated, if the rule conditions are still met. Example: **DURATION 10000**

Label	Or custom label that begins with	Value
TIMER	T	A string identifying either **int** (interval) or **cron** timer definition for scheduling the rule. Example: **TIMER "*/5 * * * *"** (every 5 minutes)
CALENDAR	E	A Quartz calendar definition for scheduling the rule. Example: **CALENDAR "* * 0-7,18-23 ? * *"** (exclude non-business hours)
AUTO-FOCUS	F	A Boolean value, applicable only to rules within agenda groups. When this option is set to **true**, the next time the rule is activated, a focus is automatically given to the agenda group to which the rule is assigned. Example: **AUTO-FOCUS true**
LOCK-ON-ACTIVE	L	A Boolean value, applicable only to rules within rule flow groups or agenda groups. When this option is set to **true**, the next time the ruleflow group for the rule becomes active or the agenda group for the rule receives a focus, the rule cannot be activated again until the ruleflow group is no longer active or the agenda group loses the focus. This is a stronger version of the **no-loop** attribute, because the activation of a matching rule is discarded regardless of the origin of the update (not only by the rule itself). This attribute is ideal for calculation rules where you have a number of rules that modify a fact and you do not want any rule re-matching and firing again. Example: **LOCK-ON-ACTIVE true**
RULEFLOW-GROUP	R	A string identifying a rule flow group. In rule flow groups, rules can fire only when the group is activated by the associated rule flow. Example: **RULEFLOW-GROUP "GroupName"**

図5.2 Sample decision table spreadsheet with attribute columns

	A	B	C	D	E	F	G	H
1		RuleSet	Charge Calculator					
2		Import	guvnor.feature.dtables.Order, guvnor.feature.dtables.Charge					
3		Variables	Integer totalCount					
4		Sequential	TRUE					
5		SequentialMaxPriority	10					
6								
7		RuleTable Basic						
8	DESCRIPTION	CONDITION	CONDITION	CONDITION	ACTION			
9		$order : Order						
10		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	insert(new Charge($1));			
11		Min items	Max items	Delivered in days	Pay charge in US dollars			
12	expensive	0	3	1	35			
13		0	3	2	15			
14		0	3		10			
15		4		1	$order.getItemsCount() * 7.5			
16		4		2	$order.getItemsCount() * 3.5			
17	cheap	4			$order.getItemsCount() * 2.5			
18								
19		RuleTable Expensive						
20	DESCRIPTION	CONDITION	CONDITION	CONDITION	CONDITION	RULEFLOW-GROUP	NOLOOP	ACTION
21		$order : Order						
22		itemsCount > $1	itemsCount <= $1	deliverInDays == $1	totalPrice > $1			insert(new Charge($1));
23		Min items	Max items	Delivered in days	More expensive than			Pay charge in US dollars
24	expensive	0	3	1	300		TRUE	25
25		0	3	2	300		TRUE	10
26		0	3		300		TRUE	$order.getItemsCount() * 1.5
27		4		1	300	discount assessment		$order.getItemsCount() * 5
28		4		2	300	discount assessment		$order.getItemsCount() * 2
29	cheap	4			300	discount assessment		0
30								

第6章 UPLOADING SPREADSHEET DECISION TABLES

After you define your rules in an external XLS or XLSX spreadsheet of decision tables, you can upload the spreadsheet file to your project in Business Central.

> **重要**
>
> You should typically upload only one spreadsheet of decision tables, containing all necessary **RuleTable** definitions, per rule package in Business Central. You can upload separate decision table spreadsheets for separate packages, but uploading multiple spreadsheets in the same package can cause compilation errors from conflicting **RuleSet** or **RuleTable** attributes and is therefore not recommended.

Procedure

1. In Business Central, go to **Menu → Design → Projects** and click the project name.

2. Click **Add Asset → Decision Table (Spreadsheet)**.

3. Enter an informative **Decision Table** name and select the appropriate **Package**.

4. Select the file type (**xls** or **xlsx**), click the **Choose File** icon, and select the spreadsheet. Click **Ok** to upload.

5. In the decision tables designer, click **Validate** in the upper-right toolbar to validate the table. If the table validation fails, open the XLS or XLSX file and address any syntax errors. For syntax help, see 5章*Defining spreadsheet decision tables* .

> **CONVERTING A SPREADSHEET DECISION TABLE TO A GUIDED DECISION TABLE**
>
> To convert a spreadsheet decision table to a guided decision table, click **Convert** in the upper-right toolbar in the decision tables designer. Converting the spreadsheet to a guided decision table enables you to edit the table data directly in Business Central. For more information about guided decision tables, see **Designing a decision service using guided decision tables**.

第7章 EXECUTING RULES

After you identify example rules or create your own rules in Business Central, you can build and deploy the associated project and execute rules locally or on Decision Server to test the rules.

Prerequisites

- Business Central and Decision Server are installed and running. For installation options, see **Planning a Red Hat Decision Manager installation**

Procedure

1. In Business Central, go to **Menu → Design → Projects** and click the project name.

2. In the upper-right corner of the project **Assets** page, click **Deploy** to build the project and deploy it to Decision Server. If the build fails, address any problems described in the **Alerts** panel at the bottom of the screen.
   For more information about project deployment options, see **Packaging and deploying a Red Hat Decision Manager project**.

3. Create a Maven or Java project outside of Business Central, if not created already, that you can use for executing rules locally or that you can use as a client application for executing rules on Decision Server. The project must contain a **pom.xml** file and any other required components for executing the project resources.
   For example test projects, see "Other methods for creating and executing DRL rules".

4. Open the **pom.xml** file of your test project or client application and add the following dependencies, if not added already:

   - **kie-ci**: Enables your client application to load Business Central project data locally using **ReleaseId**

   - **kie-server-client**: Enables your client application to interact remotely with assets on Decision Server

   - **slf4j**: (Optional) Enables your client application to use Simple Logging Facade for Java (SLF4J) to return debug logging information after you interact with Decision Server

   Example dependencies for Red Hat Decision Manager 7.3 in a client application **pom.xml** file:

```xml
<!-- For local execution -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>7.18.0.Final-redhat-00002</version>
</dependency>

<!-- For remote execution on Decision Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>7.18.0.Final-redhat-00002</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
```

```
    <groupId>org.slf4j</groupId>
    <artifactId>slf4j-simple</artifactId>
    <version>1.7.25</version>
  </dependency>
```

For available versions of these artifacts, search the group ID and artifact ID in the Nexus Repository Manager online.

注記

Instead of specifying a Red Hat Decision Manager **<version>** for individual dependencies, consider adding the Red Hat Business Automation bill of materials (BOM) dependency to your project **pom.xml** file. The Red Hat Business Automation BOM applies to both Red Hat Decision Manager and Red Hat Process Automation Manager. When you add the BOM files, the correct versions of transitive dependencies from the provided Maven repositories are included in the project.

Example BOM dependency:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.3.0.GA-redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

For more information about the Red Hat Business Automation BOM, see What is the mapping between Red Hat Decision Manager and the Maven library version?.

---

5. Ensure that the dependencies for artifacts containing model classes are defined in the client application **pom.xml** file exactly as they appear in the **pom.xml** file of the deployed project. If dependencies for model classes differ between the client application and your projects, execution errors can occur.

   To access the project **pom.xml** file in Business Central, select any existing asset in the project and then in the **Project Explorer** menu on the left side of the screen, click the **Customize View** gear icon and select **Repository View → pom.xml**.

   For example, the following **Person** class dependency appears in both the client and deployed project **pom.xml** files:

```
<dependency>
  <groupId>com.sample</groupId>
  <artifactId>Person</artifactId>
  <version>1.0.0</version>
</dependency>
```

6. If you added the **slf4j** dependency to the client application **pom.xml** file for debug logging, create a **simplelogger.properties** file on the relevant classpath (for example, in **src/main/resources/META-INF** in Maven) with the following content:

```
org.slf4j.simpleLogger.defaultLogLevel=debug
```

7. In your client application, create a **.java** main class containing the necessary imports and a **main()** method to load the KIE base, insert facts, and execute the rules.
For example, a **Person** object in a project contains getter and setter methods to set and retrieve the first name, last name, hourly rate, and the wage of a person. The following **Wage** rule in a project calculates the wage and hourly rate values and displays a message based on the result:

```
package com.sample;

import com.sample.Person;

dialect "java"

rule "Wage"
  when
    Person(hourlyRate * wage > 100)
    Person(name : firstName, surname : lastName)
  then
    System.out.println("Hello" + " " + name + " " + surname + "!");
    System.out.println("You are rich!");
end
```

To test this rule locally outside of Decision Server (if desired), configure the **.java** class to import KIE services, a KIE container, and a KIE session, and then use the **main()** method to fire all rules against a defined fact model:

Executing rules locally

```
import org.kie.api.KieServices;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;

public class RulesTest {

  public static final void main(String[] args) {
    try {
      // Identify the project in the local repository:
      ReleaseId rid = new ReleaseId();
      rid.setGroupId("com.myspace");
      rid.setArtifactId("MyProject");
      rid.setVersion("1.0.0");

      // Load the KIE base:
      KieServices ks = KieServices.Factory.get();
      KieContainer kContainer = ks.newKieContainer(rid);
      KieSession kSession = kContainer.newKieSession();

      // Set up the fact model:
      Person p = new Person();
      p.setWage(12);
      p.setFirstName("Tom");
      p.setLastName("Summers");
      p.setHourlyRate(10);

      // Insert the person into the session:
```

```
      kSession.insert(p);

      // Fire all rules:
      kSession.fireAllRules();
      kSession.dispose();
    }

    catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

To test this rule on Decision Server, configure the **.java** class with the imports and rule execution information similarly to the local example, and additionally specify KIE services configuration and KIE services client details:

Executing rules on Decision Server

```
package com.sample;

import java.util.ArrayList;
import java.util.HashSet;
import java.util.List;
import java.util.Set;

import org.kie.api.command.BatchExecutionCommand;
import org.kie.api.command.Command;
import org.kie.api.KieServices;
import org.kie.api.runtime.ExecutionResults;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.RuleServicesClient;

import com.sample.Person;

public class RulesTest {

  private static final String containerName = "testProject";
  private static final String sessionName = "myStatelessSession";

  public static final void main(String[] args) {
    try {
      // Define KIE services configuration and client:
      Set<Class<?>> allClasses = new HashSet<Class<?>>();
      allClasses.add(Person.class);
      String serverUrl = "http://$HOST:$PORT/kie-server/services/rest/server";
      String username = "$USERNAME";
      String password = "$PASSWORD";
      KieServicesConfiguration config =
        KieServicesFactory.newRestConfiguration(serverUrl,
```

```
                         username,
                         password);
      config.setMarshallingFormat(MarshallingFormat.JAXB);
      config.addExtraClasses(allClasses);
      KieServicesClient kieServicesClient =
        KieServicesFactory.newKieServicesClient(config);

      // Set up the fact model:
      Person p = new Person();
      p.setWage(12);
      p.setFirstName("Tom");
      p.setLastName("Summers");
      p.setHourlyRate(10);

      // Insert Person into the session:
      KieCommands kieCommands = KieServices.Factory.get().getCommands();
      List<Command> commandList = new ArrayList<Command>();
      commandList.add(kieCommands.newInsert(p, "personReturnId"));

      // Fire all rules:
      commandList.add(kieCommands.newFireAllRules("numberOfFiredRules"));
      BatchExecutionCommand batch = kieCommands.newBatchExecution(commandList,
  sessionName);

      // Use rule services client to send request:
      RuleServicesClient ruleClient =
  kieServicesClient.getServicesClient(RuleServicesClient.class);
      ServiceResponse<ExecutionResults> executeResponse =
  ruleClient.executeCommandsWithResults(containerName, batch);
      System.out.println("number of fired rules:" +
  executeResponse.getResult().getValue("numberOfFiredRules"));
    }

    catch (Throwable t) {
      t.printStackTrace();
    }
  }
}
```

8. Run the configured **.java** class from your project directory. You can run the file in your development platform (such as Red Hat JBoss Developer Studio) or in the command line. Example Maven execution (within project directory):

```
mvn clean install exec:java -Dexec.mainClass="com.sample.app.RulesTest"
```

Example Java execution (within project directory)

```
javac -classpath "./$DEPENDENCIES/*:." RulesTest.java
java -classpath "./$DEPENDENCIES/*:." RulesTest
```

9. Review the rule execution status in the command line and in the server log. If any rules do not execute as expected, review the configured rules in the project and the main class configuration to validate the data provided.

7.1. EXECUTABLE RULE MODELS

Executable rule models are embeddable models that provide a Java-based representation of a rule set for execution at build time. The executable model is a more efficient alternative to the standard asset packaging in Red Hat Decision Manager and enables KIE containers and KIE bases to be created more quickly, especially when you have large lists of DRL (Drools Rule Language) files and other Red Hat Decision Manager assets. The model is low level and enables you to provide all necessary execution information, such as the lambda expressions for the index evaluation.

Executable rule models provide the following specific advantages for your projects:

- **Compile time:** Traditionally, a packaged Red Hat Decision Manager project (KJAR) contains a list of DRL files and other Red Hat Decision Manager artifacts that define the rule base together with some pre-generated classes implementing the constraints and the consequences. Those DRL files must be parsed and compiled when the KJAR is downloaded from the Maven repository and installed in a KIE container. This process can be slow, especially for large rule sets. With an executable model, you can package within the project KJAR the Java classes that implement the executable model of the project rule base and re-create the KIE container and its KIE bases out of it in a much faster way. In Maven projects, you use the **kie-maven-plugin** to automatically generate the executable model sources from the DRL files during the compilation process.

- **Run time:** In an executable model, all constraints are defined as Java lambda expressions. The same lambda expressions are also used for constraints evaluation, so you no longer need to use **mvel** expressions for interpreted evaluation nor the just-in-time (JIT) process to transform the **mvel**-based constraints into bytecode. This creates a quicker and more efficient run time.

- **Development time:** An executable model enables you to develop and experiment with new features of the decision engine without needing to encode elements directly in the DRL format or modify the DRL parser to support them.

> **注記**
>
> For query definitions in executable rule models, you can use up to 10 arguments only.
>
> For variables within rule consequences in executable rule models, you can use up to 12 bound variables only (including the built-in **drools** variable). For example, the following rule consequence uses more than 12 bound variables and creates a compilation error:
>
> ```
> ...
> then
>   $input.setNo13Count(functions.sumOf(new Object[]{$no1Count_1, $no2Count_1,
> $no3Count_1, ..., $no13Count_1}).intValue());
>   $input.getFirings().add("fired");
>   update($input);
> ```

7.1.1. Embedding an executable rule model in a Maven project

You can embed an executable rule model in your Maven project to compile your rule assets more efficiently at build time.

Prerequisite

You have a Mavenized project that contains Red Hat Decision Manager business assets.

Procedure

1. In the **pom.xml** file of your Maven project, ensure that the packaging type is set to **kjar** and add the **kie-maven-plugin** build component:

```
<packaging>kjar</packaging>
...
<build>
  <plugins>
    <plugin>
      <groupId>org.kie</groupId>
      <artifactId>kie-maven-plugin</artifactId>
      <version>${rhdm.version}</version>
      <extensions>true</extensions>
    </plugin>
  </plugins>
</build>
```

The **kjar** packaging type activates the **kie-maven-plugin** component to validate and pre-compile artifact resources. The **<version>** is the Maven artifact version for Red Hat Decision Manager currently used in your project (for example, 7.18.0.Final-redhat-00002). These settings are required to properly package the Maven project.

> **注記**
>
> Instead of specifying a Red Hat Decision Manager **<version>** for individual dependencies, consider adding the Red Hat Business Automation bill of materials (BOM) dependency to your project **pom.xml** file. The Red Hat Business Automation BOM applies to both Red Hat Decision Manager and Red Hat Process Automation Manager. When you add the BOM files, the correct versions of transitive dependencies from the provided Maven repositories are included in the project.
>
> Example BOM dependency:
>
> ```
> <dependency>
>   <groupId>com.redhat.ba</groupId>
>   <artifactId>ba-platform-bom</artifactId>
>   <version>7.3.0.GA-redhat-00002</version>
>   <scope>import</scope>
>   <type>pom</type>
> </dependency>
> ```
>
> For more information about the Red Hat Business Automation BOM, see What is the mapping between RHDM product and maven library version?.

2. Add the following dependencies to the **pom.xml** file to enable rule assets to be built from an executable model:

   - **drools-canonical-model**: Enables an executable canonical representation of a rule set model that is independent from Red Hat Decision Manager

   - **drools-model-compiler**: Compiles the executable model into Red Hat Decision Manager internal data structures so that it can be executed by the decision engine

```
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-canonical-model</artifactId>
  <version>${rhdm.version}</version>
</dependency>

<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-model-compiler</artifactId>
  <version>${rhdm.version}</version>
</dependency>
```

3. In a command terminal, navigate to your Maven project directory and run the following command to build the project from an executable model:

```
mvn clean install -DgenerateModel=<VALUE>
```

The **-DgenerateModel=<VALUE>** property enables the project to be built as a model-based KJAR instead of a DRL-based KJAR.

Replace **<VALUE>** with one of three values:

- **YES**: Generates the executable model corresponding to the DRL files in the original project and excludes the DRL files from the generated KJAR.

- **WITHDRL**: Generates the executable model corresponding to the DRL files in the original project and also adds the DRL files to the generated KJAR for documentation purposes (the KIE base is built from the executable model regardless).

- **NO**: Does not generate the executable model.

Example build command:

```
mvn clean install -DgenerateModel=YES
```

For more information about packaging Maven projects, see **Packaging and deploying a Red Hat Decision Manager project**.

7.1.2. Embedding an executable rule model in a Java application

You can embed an executable rule model programmatically within your Java application to compile your rule assets more efficiently at build time.

Prerequisite

You have a Java application that contains Red Hat Decision Manager business assets.

Procedure

1. Add the following dependencies to the relevant classpath for your Java project:

   - **drools-canonical-model**: Enables an executable canonical representation of a rule set model that is independent from Red Hat Decision Manager

- **drools-model-compiler**: Compiles the executable model into Red Hat Decision Manager internal data structures so that it can be executed by the decision engine

```xml
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-canonical-model</artifactId>
  <version>${rhdm.version}</version>
</dependency>

<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-model-compiler</artifactId>
  <version>${rhdm.version}</version>
</dependency>
```

The **<version>** is the Maven artifact version for Red Hat Decision Manager currently used in your project (for example, 7.18.0.Final-redhat-00002).

> **注記**
>
> Instead of specifying a Red Hat Decision Manager **<version>** for individual dependencies, consider adding the Red Hat Business Automation bill of materials (BOM) dependency to your project **pom.xml** file. The Red Hat Business Automation BOM applies to both Red Hat Decision Manager and Red Hat Process Automation Manager. When you add the BOM files, the correct versions of transitive dependencies from the provided Maven repositories are included in the project.
>
> Example BOM dependency:
>
> ```xml
> <dependency>
>   <groupId>com.redhat.ba</groupId>
>   <artifactId>ba-platform-bom</artifactId>
>   <version>7.3.0.GA-redhat-00002</version>
>   <scope>import</scope>
>   <type>pom</type>
> </dependency>
> ```
>
> For more information about the Red Hat Business Automation BOM, see What is the mapping between RHDM product and maven library version?.

2. Add rule assets to the KIE virtual file system **KieFileSystem** and use **KieBuilder** with **buildAll(ExecutableModelProject.class)** specified to build the assets from an executable model:

```java
import org.kie.api.KieServices;
import org.kie.api.builder.KieFileSystem;
import org.kie.api.builder.KieBuilder;

  KieServices ks = KieServices.Factory.get();
  KieFileSystem kfs = ks.newKieFileSystem()
  kfs.write("src/main/resources/KBase1/ruleSet1.drl", stringContainingAValidDRL)
  .write("src/main/resources/dtable.xls",
    kieServices.getResources().newInputStreamResource(dtableFileStream));

  KieBuilder kieBuilder = ks.newKieBuilder( kfs );
```

```
// Build from an executable model
kieBuilder.buildAll( ExecutableModelProject.class )
assertEquals(0, kieBuilder.getResults().getMessages(Message.Level.ERROR).size());
```

After **KieFileSystem** is built from the executable model, the resulting **KieSession** uses constraints based on lambda expressions instead of less-efficient **mvel** expressions. If **buildAll()** contains no arguments, the project is built in the standard method without an executable model.

As a more manual alternative to using **KieFileSystem** for creating executable models, you can define a **Model** with a fluent API and create a **KieBase** from it:

```
Model model = new ModelImpl().addRule( rule );
KieBase kieBase = KieBaseBuilder.createKieBaseFromModel( model );
```

For more information about packaging projects programmatically within a Java application, see Packaging and deploying a Red Hat Decision Manager project

第8章 NEXT STEPS

- **Testing a decision service using test scenarios**

- **Packaging and deploying a Red Hat Decision Manager project**

付録A VERSIONING INFORMATION

Documentation last updated on Monday, November 15, 2021.