Red Hat Process Automation Manager 7.8

Testing a decision service using test scenarios

Last Updated: 2020-08-05

Red Hat Process Automation Manager 7.8 Testing a decision service using test scenarios

Red Hat Customer Content Services
brms-docs@redhat.com

Legal Notice

Abstract

This document describes how to test a decision service using test scenarios in Red Hat Process Automation Manager 7.8.

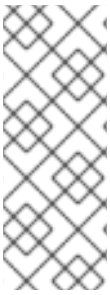
Table of Contents

PREFACE

As a business analyst or business rules developer, you can use test scenarios in Business Central to test a decision service before a project is deployed. You can test DMN-based and rules-based decision services to ensure these are functioning properly and as expected. Also, you can test a decision service at any time during project development.

Prerequisites

- The space and project for the decision service have been created in Business Central. For details, see *Getting started with decision services* .
- Business rules and their associated data objects have been defined for the rules-based decision service. For details, see *Designing a decision service using guided decision tables* .
- DMN decision logic and its associated custom data types have been defined for the DMN-based decision service. For details, see *Designing a decision service using DMN models* .



NOTE

Having defined business rules is not a technical prerequisite for test scenarios, because the scenarios can test the defined data that constitutes the business rules. However, creating the rules first is helpful so that you can also test entire rules in test scenarios and so that the scenarios more closely match the intended decision service. For DMN-based test scenarios ensure that the DMN decision logic and its associated custom data types are defined for the decision service.

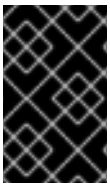
CHAPTER 1. TEST SCENARIOS

Test scenarios in Red Hat Process Automation Manager enable you to validate the functionality of business rules and business rule data (for rules-based test scenarios) or of DMN models (for DMN-based test scenarios) before deploying them into a production environment. With a test scenario, you use data from your project to set given conditions and expected results based on one or more defined business rules. When you run the scenario, the expected results and actual results of the rule instance are compared. If the expected results match the actual results, the test is successful. If the expected results do not match the actual results, then the test fails.

Red Hat Process Automation Manager currently supports both the new **Test Scenarios** designer and the former **Test Scenarios (Legacy)** designer. The default designer is the new test scenarios designer, which supports testing of both rules and DMN models and provides an enhanced overall user experience with test scenarios. If required, you can continue to use the legacy test scenarios designer, which supports rule-based test scenarios only.

You can run the defined test scenarios in a number of ways, for example, you can run available test scenarios at the project level or inside a specific test scenario asset. Test scenarios are independent and cannot affect or modify other test scenarios. You can run test scenarios at any time during project development in Business Central. You do not have to compile or deploy your decision service to run test scenarios.

You can import data objects from different packages to the same project package as the test scenario. Assets in the same package are imported by default. After you create the necessary data objects and the test scenario, you can use the **Data Objects** tab of the test scenarios designer to verify that all required data objects are listed or to import other existing data objects by adding a **New item**.



IMPORTANT

Throughout the test scenarios documentation, all references to *test scenarios* and the *test scenarios designer* are for the new version, unless explicitly noted as the legacy version.

CHAPTER 2. DATA OBJECTS

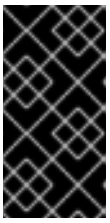
Data objects are the building blocks for the rule assets that you create. Data objects are custom data types implemented as Java objects in specified packages of your project. For example, you might create a **Person** object with data fields **Name**, **Address**, and **DateOfBirth** to specify personal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

2.1. 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.

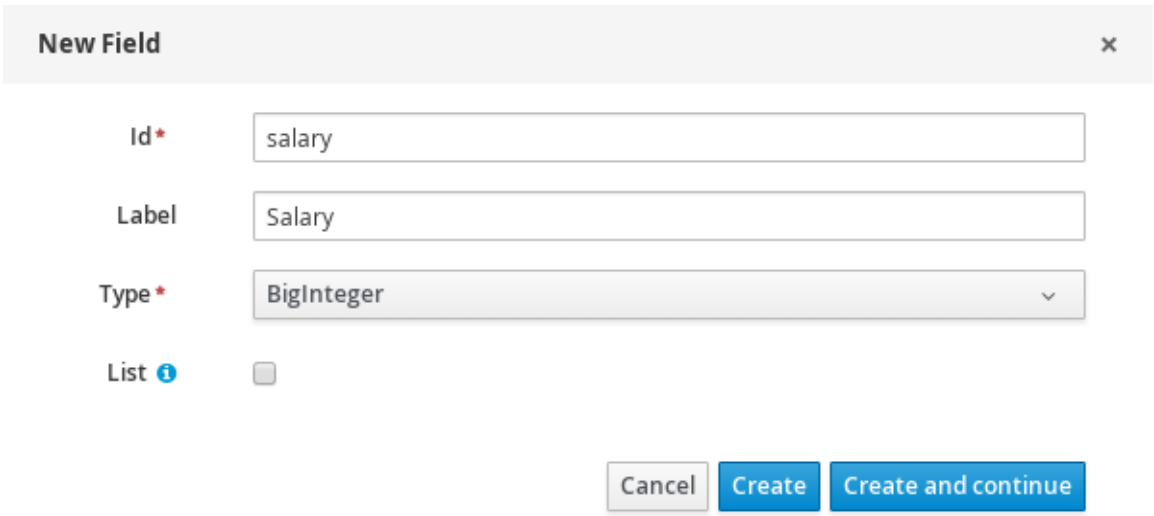


IMPORTING DATA OBJECTS FROM OTHER PACKAGES

You can import an existing data object from another package directly into the asset designers like guided rules or guided decision table designers. 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:** (Optional) Select this check box to enable the field to hold multiple items for the specified type.

Figure 2.1. Add data fields to a data object



7. Click **Create** to add the new field, or click **Create and continue** to add the new field and continue adding other fields.

NOTE

To edit a field, select the field row and use the **general properties** on the right side of the screen.

CHAPTER 3. TEST SCENARIOS DESIGNER IN BUSINESS CENTRAL

The test scenarios designer provides a tabular layout that helps you in defining a scenario template and all the associated test cases. The designer layout consists of a table which has a header and the individual rows. The header consists of three parts, the **GIVEN** and **EXPECT** row, a row with instances, and a row with corresponding fields. The header is also known as test scenario template and the individual rows are called test scenarios definitions.

The test scenario template or header has the following two parts:

- **GIVEN** data objects and their fields - represents the input information
- **EXPECT** data objects and their fields - represents the objects and their fields whose exact values are checked based on the given information and which also constitutes the expected result.

The test scenarios definitions represent the separate test cases of a template.

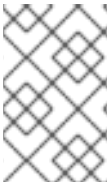
You can access the **Project Explorer** from the left panel of the designer whereas from the right panel you can access the **Settings**, **Test Tools**, **Scenario Cheatsheet**, **Test Report** and the **Coverage Report** tabs. You can access the **Settings** tab to view and edit the global settings of rule-based and DMN-based test scenarios. You can use the **Test Tools** to configure the data object mappings. **Scenario Cheatsheet** tab contains notes and the cheat sheet which you can use as reference. The **Test Report** tab displays the overview of the tests and the scenario status. To view the test coverage statistics, you can use the **Coverage Report** tab from the right side of the test scenario designer.

3.1. IMPORTING DATA OBJECTS

The test scenarios designer loads all data objects that are located in the same package as the test scenario. You can view all the data objects from the **Data Objects** tab in the designer. The loaded data objects are also displayed in the **Test Tools** panel.

You need to close and reopen the designer in case the data objects change (for example, when a new data object is created or when an existing one is deleted). Select a data object from the list to display its fields and the field types.

In case you want to use a data object located in a different package than the test scenario, you need to import the data object first. Follow the procedure below to import a data object for rules-based test scenarios.



NOTE

You cannot import any data objects while creating DMN-based test scenarios. DMN-based test scenarios do not use any data objects from the project but uses the custom data types defined in the DMN file.

Procedure

1. Go to **Project Explorer** panel in the test scenarios designer.
2. From **Test Scenario**, select a test scenario.
3. Select **Data Objects** tab and click **New Item**.

4. In the **Add import** window, choose the data object from the drop-down list.
5. Click **Ok** and then **Save**.
6. Close and reopen the test scenarios designer to view the new data object from the data objects list.

3.2. IMPORTING A TEST SCENARIO

You can import an existing test scenario using the **Import Asset** button in the **Asset** tab from the project view.

Procedure

1. In Business Central, go to **Menu → Design → Projects** and click the project name.
2. From the project's **Asset** tab, click **Import Asset**.
3. In the **Create new Import Asset** window,
   - Enter the name of the import asset.
   - Select the package from the **Package** drop-down list.
   - From **Please select a file to upload**, click **Choose File...** to browse to test scenario file.
4. Select the file and click **Open**.
5. Click **Ok** and the test scenario opens in the test scenario designer.

3.3. SAVING A TEST SCENARIO

You can save a test scenario at any time while creating a test scenario template or defining the test scenarios.

Procedure

1. From the test scenarios designer toolbar on the upper-right, click **Save**.
2. On the **Confirm Save** window,
   a. If you wish to add a comment regarding the test scenario, click **add a comment**.
   b. Click **Save** again.

A message stating that the test scenario was saved successfully appears on the screen.

3.4. COPYING A TEST SCENARIO

You can copy an existing test scenario to the same package or to some other package by using the **Copy** button from the upper-right toolbar.

Procedure

1. From the test scenarios designer toolbar on the upper-right, click **Copy**.

2. In the **Make a Copy** window,
    a. Enter a name in the **New Name** field.
    b. Select the package you want to copy the test scenario to.
    c. Optional: To add a comment, click **add a comment**.
    d. Click **Make a Copy**.

A message stating that the test scenario was copied successfully appears on the screen.

3.5. DOWNLOADING A TEST SCENARIO

You can download a copy of the test scenario to your local machine for future reference or as backup.

Procedure

In the test scenarios designer toolbar on the upper-right, click the **Download** icon.

The **.scesim** file is downloaded to your local machine.

3.6. SWITCHING BETWEEN VERSIONS OF A TEST SCENARIO

Business Central provides you the ability to switch between the various versions of a test scenario. Every time you save the scenario, a new version of the scenario is listed under **Latest Versions**. To use this feature, you must save the test scenario file at least once.

Procedure

1. From the test scenarios designer toolbar on the upper-right, click **Latest Version**. All the versions of the file are listed under **Latest Version**, if they exist.
2. Click the version you want to work on.
   The selected version of the test scenario opens in the test scenarios designer.
3. From the designer toolbar, click **Restore**.
4. In the **Confirm Restore**,
    a. To add a comment, click **add a comment**.
    b. Click **Restore** to confirm.

A message stating that the selected version has been reloaded successfully in the designer appears on the screen.

3.7. VIEW OR HIDE THE ALERTS PANEL

The **Alerts** panel appears at the bottom of the test scenarios designer or the project view. It contains the build information and error messages in case the executed tests are failed.

Procedure

From the designer toolbar on the upper-right, click **Hide Alerts**/**View Alerts** to enable or disable the reporting panel.

3.8. CONTEXTUAL MENU OPTIONS

The test scenarios designer provides contextual menu options, which enables you to perform basic operations on the table such as adding, deleting, and, duplicating rows and columns. To use the contextual menus, you need to right-click a table element. Menu options differ based on the table element you select.

Table 3.1. Contextual menu options

Table element	Cell label	Available context menu options
Header	# & Scenario description	Insert row below
	GIVEN & EXPECT	Insert leftmost column, Insert rightmost column, Insert row below
	INSTANCE 1, INSTANCE 2 & PROPERTY 1, PROPERTY 2	Insert column left, Insert column right, Delete column, Duplicate Instance, Insert row below
Rows	All the cells with row numbers, test scenarios description or test scenarios definition	Insert row above, Insert row below, Duplicate row, Delete row, Run scenario

Table 3.2. Description of table interactions

Table interaction	Description
Insert leftmost column	Inserts a new leftmost column (in either the GIVEN or EXPECT section of the table based on user selection).
Insert rightmost column	Inserts a new rightmost column (in either the GIVEN or EXPECT section of the table based on user selection).
Insert column left	Inserts a new column to the left of the selected column. The new column is of the same type as the selected column (in either the GIVEN or EXPECT section of the table based on user selection).
Insert column right	Inserts a new column to the right of the selected column. The new column is of the same type as the selected column (in either the GIVEN or EXPECT section of the table based on user selection).
Delete column	Deletes the selected column.
Insert row above	Inserts a new row above the selected row.
Insert row below	Inserts a new row below the selected row. If invoked from a header cell, inserts a new row with index 1.
Duplicate row	Duplicates the selected row.

Table interaction	Description
Duplicate Instance	Duplicates the selected instance.
Delete row	Deletes the selected row.
Run scenario	Runs a single test scenario.

The **Insert column right** or **Insert column left** context menu options behave differently.

- If the selected column does not have a type defined, a new column without a type is added.
- If the selected column has a type defined, either a new empty column or a column with the parent instance type is created.
  - If the action is performed from an instance header, a new column without a type is created.
  - If the action is performed from a property header, a new column with the parent instance type is created.

3.9. GLOBAL SETTINGS FOR TEST SCENARIOS

You can use the global **Settings** tab on the right side of the test scenarios designer to set and modify the additional properties of assets.

3.9.1. Configuring global settings for rule-based test scenarios

Follow the procedure below to view and edit the global settings of rule-based test scenarios.

Procedure

1. Click **Settings** tab on the right side of the test scenario designer to display the attributes.
2. Configure the following attributes in the **Settings** panel:
   - **Name**: You can change the name of the existing test scenarios by using the **Rename** option from the upper-right toolbar in the designer.
   - **Type**: This attribute specifies it is a rule-based test scenario and it is read-only.
   - **Stateless Session**: Select or clear this check box to specify if the KieSession is stateless or not.

     NOTE

     If the current KieSession is stateless and the check box is not selected, the tests will fail.

   - **KieSession**: (Optional) Enter the KieSession for the test scenario.
   - **RuleFlowGroup/AgendaGroup**: (Optional) Enter the RuleFlowGroup or AgendaGroup for the test scenario.

3. Optional: To skip the entire simulation from project level after test execution, select the check box.
4. Click **Save**.

3.9.2. Configuring global settings for DMN-based test scenarios

Follow the procedure below to view and edit the global settings of DMN-based test scenarios.

Procedure

1. Click **Settings** tab on the right side of the test scenario designer to display the attributes.
2. Configure the following attributes in the **Settings** panel:
   - **Name**: You can change the name of the existing test scenarios by using the **Rename** option from the upper-right toolbar in the designer.
   - **Type**: This attribute specifies it is a DMN-based test scenario and it is read-only.
   - **DMN model**: (Optional) Enter the DMN model for the test scenario.
   - **DMN name**: This is the name of the DMN model and it is read-only.
   - **DMN namespace**: This is the default namespace for DMN model and it is read-only.
3. Optional: To skip the entire simulation from project level after test execution, select the check box.
4. Click **Save**.

CHAPTER 4. TEST SCENARIO TEMPLATE

Before specifying test scenario definitions, you need to create a test scenario template. The header of the test scenario table defines the template for each scenario. You need to set the types of the instance and property headers for both the GIVEN and EXPECT sections. Instance headers map to a particular data object (a fact), whereas the property headers map to a particular field of the corresponding data object.

Using the test scenarios designer, you can create test scenario templates for both rule-based and DMN-based test scenarios.

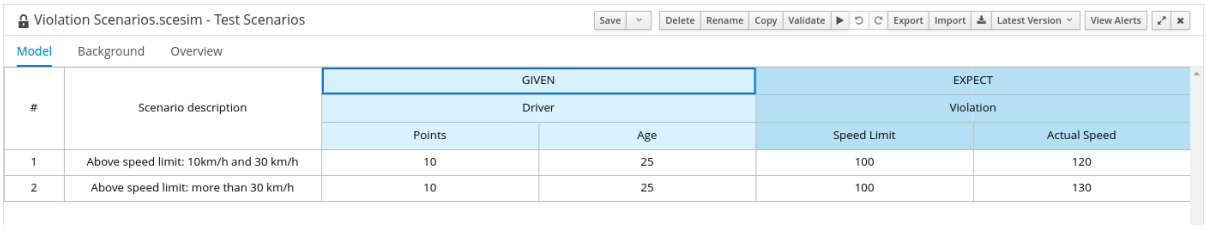
4.1. CREATING A TEST SCENARIO TEMPLATE FOR RULE-BASED TEST SCENARIOS

Create a test scenario template for rule-based test scenarios by following the procedure below to validate your rules and data.

Procedure

1. In Business Central, go to **Menu → Design → Projects** and click the project for which you want to create the test scenario.
2. Click **Add Asset → Test Scenario**.
3. Enter a **Test Scenario** name and select the appropriate **Package**. The package you select must contain all the required data objects and rule assets have been assigned or will be assigned.
4. Select **RULE** as the **Source type**.
5. Click **Ok** to create and open the test scenario in the test scenarios designer.
6. To map the **GIVEN** column header to a data object:

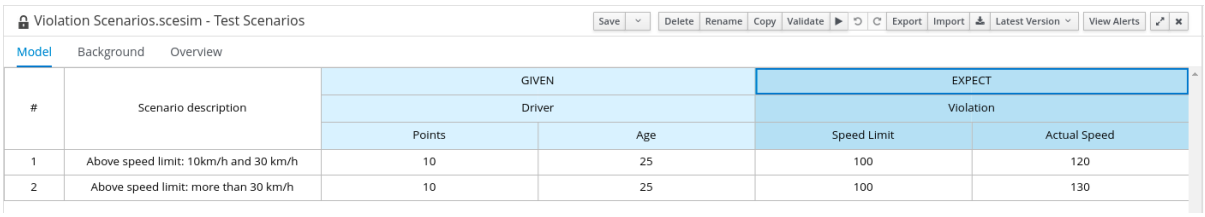
Figure 4.1. Test scenario GIVEN header cells



   a. Select an instance header cell in the **GIVEN** section.
   b. Select the data object from the **Test Tools** tab.
   c. Click **Insert Data Object**.

7. To map the **EXPECT** column header to a data object:

Figure 4.2. Test scenario EXPECT header cells



a. Select an instance header cell in the **EXPECT** section.

b. Select the data object from the **Test Tools** tab.

c. Click **Insert Data Object**.

8. To map a data object field to a property cell:

a. Select an instance header cell or property header cell.

b. Select the data object field from the **Test Tools** tab.

c. Click **Insert Data Object**.

9. To insert more properties of the data object, right-click the property header and select **Insert column right** or **Insert column left** as required.

10. To define a java method to a property cell during test scenarios execution:

a. Select an instance header cell or property header cell.

b. Select the data object field from the **Test Tools** tab.

c. Click **Insert Data Object**.

d. Use the MVEL expression with the prefix **#** to define a java method for test scenario execution.

e. To insert more properties of the data object, right-click the property header cell and select **Insert column right** or **Insert column left** as required.

11. Use the contextual menu to add or remove columns and rows as needed.

For more details about the expression syntax in rule-based scenarios, see Section 9.1, “Expression syntax in rule-based test scenarios”.

4.2. USING ALIASES IN RULE-BASED TEST SCENARIOS

In the test scenarios designer, once you map a header cell with a data object, the data object is removed from the **Test Tools** tab. You can re-map a data object to another header cell by using an alias. Aliases enable you to specify multiple instances of the same data object in a test scenario. You can also create property aliases to rename the used properties directly in the table.

Procedure

In the test scenarios designer in Business Central, double-click a header cell and manually change the name. Ensure that the aliases are uniquely named.

The instance now appears in the list of data objects in the **Test Tools** tab.

CHAPTER 5. TEST TEMPLATE FOR DMN-BASED TEST SCENARIOS

Business Central automatically generates the template for every DMN-based test scenario asset and it contains all the specified inputs and decisions of the related DMN model. For each input node in the DMN model, a **GIVEN** column is added, whereas each decision node is represented by an **EXPECT** column. You can modify the default template at any time as per your needs. Also, to test only a specific part of the whole DMN model, its possible to remove the generated columns as well as move decision nodes from the EXPECT to the GIVEN section.

5.1. CREATING A TEST SCENARIO TEMPLATE FOR DMN-BASED TEST SCENARIOS

Create a test scenario template for DMN-based scenarios by following the procedure below to validate your DMN models.

Procedure

1. In Business Central, go to **Menu → Design → Projects** and click the project that you want to create the test scenario for.
2. Click **Add Asset → Test Scenario**.
3. Enter a **Test Scenario** name and select the appropriate **Package**.
4. Select **DMN** as the **Source type**.
5. Select an existing DMN asset using the **Choose DMN asset** option.
6. Click **Ok** to create and open the test scenario in the test scenarios designer.
   The template is automatically generated and you can modify it as per your needs.
7. To define a java method to a property cell during test scenario execution:
   a. Click an instance header cell or property header cell.
   b. Select the data object field from the **Test Tools** tab.
   c. Click **Insert Data Object**.
   d. Use an expression to define a java method for test scenario execution.
   e. To add more properties to the data object, right-click the property header cell and select **Insert column right** or **Insert column left** as required.
8. Use the contextual menu to add or remove columns and rows as needed.

For more details about the expression syntax in DMN-based scenarios, see Section 9.2, “Expression syntax in DMN-based scenarios”.

CHAPTER 6. DEFINING A TEST SCENARIO

After creating a test scenario template you have to define the test scenario next. The rows of the test scenario table define the individual test scenarios. A test scenario has a unique index number, description, set of input values (the **Given** values), and a set of output values (the **Expect** values).

Prerequisites

- The test scenario template has been created for the selected test scenario.

Procedure

1. Open the test scenario in the test scenarios designer.
2. Enter a description of the test scenario and fill in required values in each cell of the row.
3. Use the contextual menu to add or remove rows as required.
   Double click a cell to start inline editing. To skip a particular cell from test evaluation, leave it empty.

After defining the test scenario, you can run the test next.

CHAPTER 7. BACKGROUND INSTANCE IN TEST SCENARIOS

In test scenario designer, you can use the **Background** tab to add and set background data for rules-based and DMN-based test scenarios. You can add and define the GIVEN data which is common for the entire test scenario simulation, based on the available data objects. **Background** tab has the ability to add and share the data among every test scenario. Data added using the **Background** tab can not be overridden by **Model** tab data.

For example, if the test scenario example requires the same value for the person **Age** in all test scenarios, you can define the **Age** value in the **Background** page and exclude that column from the test scenario table template. In this case, the **Age** is set to **25** for all test scenarios.

Figure 7.1. Example test scenarios with repeated value for Age

Model Background Overview Data Objects

#	Scenario description	GIVEN			EXPECT	
		Driver			Violation	
		Age	Points	Name	Speed Limit	Actual Speed
1	*Insert value*	25	*Insert value*	*Insert value*	*Insert value*	*Insert value*
2	*Insert value*	25	*Insert value*	*Insert value*	*Insert value*	*Insert value*

Figure 7.2. Example background definition of repeated value for Age

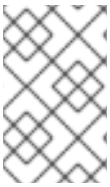
Model Background Overview Data Objects

GIVEN
Driver
Age
25

Figure 7.3. Modified test scenario template with excluded Age column

Model Background Overview Data Objects

#	Scenario description	GIVEN		EXPECT	
		Driver		Violation	
		Points	Name	Speed Limit	Actual Speed
1	*Insert value*	*Insert value*	*Insert value*	*Insert value*	*Insert value*
2	*Insert value*	*Insert value*	*Insert value*	*Insert value*	*Insert value*



NOTE

The GIVEN data which is defined in the **Background** tab can only be shared between the test scenarios of the same ***.scesim** file and will not be shared among different test scenarios.

7.1. ADDING A BACKGROUND DATA IN RULE-BASED TEST SCENARIOS

Follow the procedure below to add and set a background data in rule-based test scenarios.

Prerequisites

- The rule-based test scenario template are created for the selected test scenario. For more information about creating rule-based test scenarios, see Section 4.1, “Creating a test scenario template for rule-based test scenarios”.

- The individual test scenarios are defined. For more information about defining a test scenario, see Chapter 6, *Defining a test scenario* .

Procedure

1. Open the rule-based test scenarios in the test scenario designer.
2. Click the **Background** tab of the test scenarios designer.
3. Select an instance header cell in the **GIVEN** section to add a background data object field.
4. From the **Test Tools** panel, select the data object.
5. Click **Insert Data Object**.
6. Select a property header cell to add a background data object field.
7. From the **Test Tools** panel, select the data object.
8. Click **Insert Data Object**.
9. To add more properties to the data object, right-click the property header cell and select **Insert column right** or **Insert column left** as required.
10. Use the contextual menu to add or remove columns and rows as needed.
11. Run the defined test scenarios.

7.2. ADDING A BACKGROUND DATA IN DMN-BASED TEST SCENARIOS

Follow the procedure below to add and set a background data in DMN-based test scenarios.

Prerequisites

- The DMN-based test scenario template is created for the selected test scenario. For more information about creating DMN-based test scenarios, see Section 5.1, “Creating a test scenario template for DMN-based test scenarios”.
- The individual test scenarios are defined. For more information about defining a test scenario, see Chapter 6, *Defining a test scenario* .

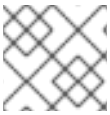
Procedure

1. Open the DMN-based test scenarios in the test scenario designer.
2. Click the **Background** tab of the test scenarios designer.
3. Select an instance header cell in the **GIVEN** section to add a background data object field.
4. From the **Test Tools** panel, select the data object.
5. Click **Insert Data Object**.
6. Select a property header cell to add a background data object field.
7. From the **Test Tools** panel, select the data object.

8. Click **Insert Data Object**.
9. To add more properties to the data object, right-click the property header cell and select **Insert column right** or **Insert column left** as required.
10. Use the contextual menu to add or remove columns and rows as needed.
11. Run the defined test scenarios.

CHAPTER 8. USING LIST AND MAP COLLECTIONS IN TEST SCENARIOS

The test scenarios designer supports list and map collections for both DMN-based as well as rules-based test scenarios. You can create and define a collection like a list or a map as the value of a particular cell in both **GIVEN** and **EXPECT** columns.



NOTE

For map entries, an entry key must be a **String** data type.

To pass the parameter in the **EXPECT** column of Rule-based collection editor use the **actualValue** keyword whereas use the **?** keyword in DMN-based test scenario.

Procedure

1. Set the column type first (use a field whose type is a list or a map).
2. Double click a cell in the column to input a value.
3. To create the list values for the data objects in the collection editor popup:
   a. Select **Create List**
   b. Click **Add new item**.
   c. Enter the required value and click the check icon  to save each collection item that you add.
   d. Click **Save**.
   e. To edit an item from the collection, click the pencil icon in the collection popup editor.
   f. Click **Save changes**.
   g. To delete an item from the collection, click the bin icon in the collection popup editor.
4. To define the list values for the data objects in the collection editor popup:
   a. Select **Define List**
   b. Use the MVEL or FEEL expression to define a list value in the text field.
   Rule-based test scenario uses MVEL expression language and DMN-based test scenario uses FEEL expression language.
   c. Click **Save**.
5. To create the map values for the data objects in the collection editor popup:
   a. Select **Create Map**.
   b. Click **Add new item**.

c. Enter the required value and click the check icon  to save each collection item that you add.

d. Click **Save**.

e. To edit an item from the collection, click the pencil icon in the collection popup editor.

f. Click **Save changes**.

g. To delete an item from the collection, click the bin icon in the collection popup editor.

6. To define the map values for the data objects in the collection editor popup:

a. Select **Define Map**.

b. Use the MVEL or FEEL expression to define a map value in the text field.
Rule-based test scenario uses MVEL expression language and DMN-based test scenario uses FEEL expression language.

c. Click **Save**.

NOTE

To define the map values for DMN-based test scenario, you can add a fact and use the FEEL expression, instead of using the collection editor.

7. Click **Remove** to delete the entire collection.

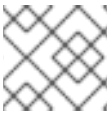
CHAPTER 9. EXPRESSION SYNTAX IN TEST SCENARIOS

The test scenarios designer supports different expression languages for both rule-based and DMN-based test scenarios. While rule-based test scenarios support the MVFLEX Expression Language (MVEL) and DMN-based test scenarios support the Friendly Enough Expression Language (FEEL).

9.1. EXPRESSION SYNTAX IN RULE-BASED TEST SCENARIOS

Rule-based test scenario supports the following built-in data types:

- String
- Boolean
- Integer
- Long
- Double
- Float
- Character
- Byte
- Short
- LocalDate



NOTE

For any other data types, use the MVEL expression with the prefix **#**.

Follow the BigDecimal example in the test scenario designer to use the **#** prefix to set the java expression:

- Enter **# java.math.BigDecimal.valueOf(10)** for the **GIVEN** column value.
- Enter **# actualValue.intValue() == 10** for the **EXPECT** column value.

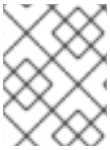
You can refer to the actual value of the **EXPECT** column in the java expression to execute a condition.

The following rule-based test scenario definition expressions are supported by the test scenarios designer:

Table 9.1. Description of expressions syntax

Operator	Description
=	Specifies equal to a value. This is default for all columns and is the only operator supported by the GIVEN column.

Operator	Description
=, =!, <>	Specifies inequality of a value. This operator can be combined with other operators.
<, >, <=, >=	Specifies a comparison: less than, greater than, less or equals than, and greater or equals than.
#	This operator is used to set the java expression value to a property header cell which can be executed as a java method.
[value1, value2, value3]	Specifies a list of values. If one or more values are valid, the scenario definition is evaluated as true.
expression1; expression2; expression3	Specifies a list of expressions. If all expressions are valid, the scenario definition is evaluated as true.

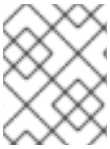


NOTE

An empty cell is skipped from evaluation. To define an empty string, use **=**,**[]**, or **;**. To define a null value, use **null**.

Table 9.2. Example expressions

Expression	Description
-1	The actual value is equal to -1.
< 0	The actual value is less than 0.
! > 0	The actual value is not greater than 0.
[-1, 0, 1]	The actual value is equal to either -1 or 0 or 1.
<> [1, -1]	The actual value is neither equal to 1 nor -1.
! 100; 0	The actual value is not equal to 100 but is equal to 0.
!= < 0; <> > 1	The actual value is neither less than 0 nor greater than 1.
<> <= 0; >= 1	The actual value is neither less than 0 nor equal to 0 but is greater than or equal to 1.



NOTE

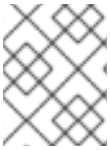
You can refer to the supported commands and syntax in the **Scenario Cheatsheet** tab on the right of the rule-based test scenarios designer.

9.2. EXPRESSION SYNTAX IN DMN-BASED SCENARIOS

The following data types are supported by the DMN-based test scenarios in the test scenarios designer:

Table 9.3. Data types supported by DMN-based scenarios

Supported data types	Description
numbers & strings	Strings must be delimited by quotation marks, for example, **"John Doe"**, **"Brno"** or **""**.
boolean values	**true**, **false**, and **null**.
dates and time	For example, **date("2019-05-13")** or **time("14:10:00+02:00")**.
functions	Supports built-in math functions, for example, **avg**, **max**.
contexts	For example, **{x : 5, y : 3}**.
ranges and lists	For example, **[1 .. 10]** or **[2, 3, 4, 5]**.



NOTE

You can refer to the supported commands and syntax in the **Scenario Cheatsheet** tab on the right of the DMN-based test scenarios designer.

CHAPTER 10. RUNNING THE TEST SCENARIOS

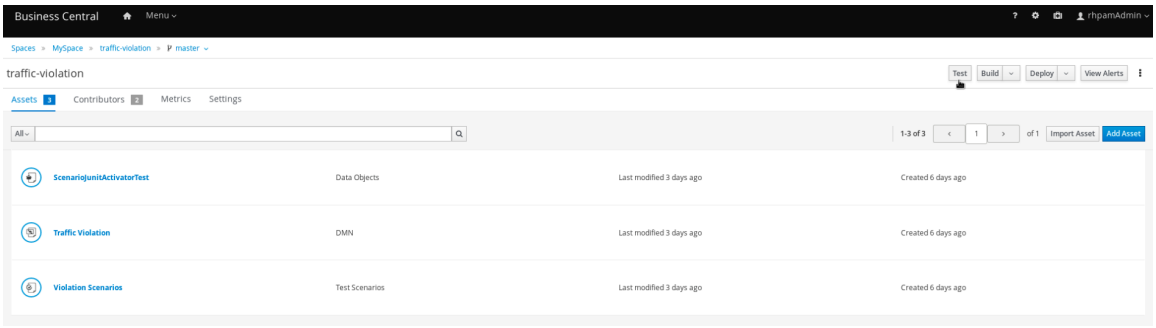
After creating a test scenario template and defining the test scenarios, you can run the tests to validate your business rules and data.

Procedure

1. To run defined test scenarios, do any of the following tasks:

   - To execute all the available test scenarios in your project inside multiple assets, in the upper-right corner of your project page, click **Test**.

     Figure 10.1. Run all the test scenarios from the project view

   - To execute all available test scenarios defined in a **.scesim** file, at the top of the Test Scenario designer, click the **Run Test**  icon.

   - To run a single test scenario defined in a single **.scesim** file, right-click the row of the test scenario you want to run and select **Run scenario**.

2. The **Test Report** panel displays the overview of the tests and the scenario status.
   After the tests execute, if the values entered in the test scenario table do not match with the expected values, then the corresponding cells are highlighted.

3. If tests fail, you can do the following tasks to troubleshoot the failure:

   - To review the error message in the pop-up window, hover your mouse cursor over the highlighted cell.

   - To open the **Alerts** panel at the bottom of the designer or the project view for the error messages, click **View Alerts**.

   - Make the necessary changes and run the test again until the scenario passes.

CHAPTER 11. RUNNING A TEST SCENARIO LOCALLY

In Red Hat Process Automation Manager, you can either run the test scenarios directly in Business Central or locally using the command line.

Procedure

1. In Business Central, go to **Menu → Design → Projects** and click the project name.
2. On the Project's home page, select the **Settings** tab.
3. Select **git URL** and click the **Clipboard**  to copy the git url.
4. Open a command terminal and navigate to the directory where you want to clone the git project.
5. Run the following command:

```
git clone your_git_project_url
```

   Replace **your_git_project_url** with relevant data like **git://localhost:9418/MySpace/ProjectTestScenarios**.

6. Once the project is successfully cloned, navigate to the git project directory and execute the following command:

```
mvn clean test
```

   Your project's build information and the test results (such as, the number of tests run and whether the test run was a success or not) are displayed in the command terminal. In case of failures, make the necessary changes in Business Central, pull the changes and run the command again.

CHAPTER 12. EXPORTING AND IMPORTING TEST SCENARIO SPREADSHEETS

These sections show how to export and import test scenario spreadsheets in the test scenario designer. You can analyze and manage test scenario spreadsheets with software such as Microsoft Excel or LibreOffice Calc. Test scenario designer supports the **.CSV** file format. For more information about the RFC specification for the Comma-Separated Values (CSV) format, see Common Format and MIME Type for Comma-Separated Values (CSV) Files.

12.1. EXPORTING A TEST SCENARIO SPREADSHEET

Follow the procedure below to export a test scenario spreadsheet using the Test Scenario designer.

Procedure

1. In the Test Scenario designer toolbar on the upper-right, click **Export** 

button.
2. Select a destination in your local file directory and confirm to save the **.CSV** file.

The **.CSV** file is exported to your local machine.

12.2. IMPORTING A TEST SCENARIO SPREADSHEET

Follow the procedure below to import a test scenario spreadsheet using the Test Scenario designer.

Procedure

1. In the Test Scenario designer toolbar on the upper-right, click **Import** 

button.
2. In the **Select file to Import** prompt, click **Choose File...** and select the **.CSV** file you would like to import from your local file directory.
3. Click **Import**.

The **.CSV** file is imported to the Test Scenario designer.

WARNING

You must not modify the headers in the selected **.CSV** file. Otherwise, the spreadsheet may not be successfully imported.

CHAPTER 13. COVERAGE REPORTS FOR TEST SCENARIOS

The test scenario designer provides a clear and coherent way of displaying the test coverage statistics using the **Coverage Report** tab on the right side of the test scenario designer. You can also download the coverage report to view and analyze the test coverage statistics. Downloaded test scenario coverage report supports the **.CSV** file format. For more information about the RFC specification for the Comma-Separated Values (CSV) format, see Common Format and MIME Type for Comma-Separated Values (CSV) Files.

You can view the coverage report for rule-based and DMN-based test scenarios.

13.1. GENERATING COVERAGE REPORTS FOR RULE-BASED TEST SCENARIOS

In rule-based test scenarios, the **Coverage Report** tab contains the detailed information about the following:

- Number of available rules
- Number of fired rules
- Percentage of fired rules
- Percentage of executed rules represented as a pie chart
- Number of times each rule has executed
- The rules that are executed for each defined test scenario

Follow the procedure to generate a coverage report for rule-based test scenarios:

Prerequisites

- The rule-based test scenario template are created for the selected test scenario. For more information about creating rule-based test scenarios, see Section 4.1, “Creating a test scenario template for rule-based test scenarios”.
- The individual test scenarios are defined. For more information about defining a test scenario, see Chapter 6, *Defining a test scenario* .

NOTE

To generate the coverage report for rule-based test scenario, you must create at least one rule.

Procedure

1. Open the rule-based test scenarios in the test scenario designer.
2. Run the defined test scenarios.
3. Click **Coverage Report** on the right of the test scenario designer to display the test coverage statistics.
4. Optional: To download the test scenario coverage report, Click **Download report**.

13.2. GENERATING COVERAGE REPORTS FOR DMN-BASED TEST SCENARIOS

In DMN-based test scenarios, the **Coverage Report** tab contains the detailed information about the following:

- Number of available decisions
- Number of executed decisions
- Percentage of executed decisions
- Percentage of executed decisions represented as a pie chart
- Number of times each decision has executed
- Decisions that are executed for each defined test scenario

Follow the procedure to generate a coverage report for DMN-based test scenarios:

Prerequisites

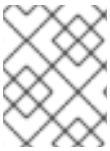
- The DMN-based test scenario template is created for the selected test scenario. For more information about creating DMN-based test scenarios, see Section 5.1, "Creating a test scenario template for DMN-based test scenarios".
- The individual test scenarios are defined. For more information about defining a test scenario, see Chapter 6, *Defining a test scenario* .

Procedure

1. Open the DMN-based test scenarios in the test scenario designer.
2. Run the defined test scenarios.
3. Click **Coverage Report** on the right of the test scenario designer to display the test coverage statistics.
4. Optional: To download the test scenario coverage report, Click **Download report**.

CHAPTER 14. EXECUTING A TEST SCENARIO USING THE KIE SERVER REST API

Directly interacting with the REST endpoints of KIE Server provides the most separation between the calling code and the decision logic definition. You can use the KIE Server REST API to execute the test scenarios externally. It executes the test scenarios against the deployed project.



NOTE

This functionality is disabled by default, use **org.kie.scenariosimulation.server.ext.disabled** system property to enable it.

For more information about the KIE Server REST API, see *Interacting with Red Hat Process Automation Manager using KIE APIs*.

Prerequisites

- KIE Server is installed and configured, including a known user name and credentials for a user with the **kie-server** role. For installation options, see *Planning a Red Hat Process Automation Manager installation*.
- You have built the project as a KJAR artifact and deployed it to KIE Server.
- You have the ID of the KIE container.

Procedure

1. Determine the base URL for accessing the KIE Server REST API endpoints. This requires knowing the following values (with the default local deployment values as an example):
   - Host (**localhost**)
   - Port (**8080**)
   - Root context (**kie-server**)
   - Base REST path (**services/rest/**)

   Example base URL in local deployment for the traffic violations project:

   http://localhost:8080/kie-server/services/rest/server/containers/traffic_1.0.0-SNAPSHOT

2. Determine user authentication requirements.
   When users are defined directly in the KIE Server configuration, HTTP Basic authentication is used and requires the user name and password. Successful requests require that the user have the **kie-server** role.

   The following example demonstrates how to add credentials to a curl request:

   ```
   curl -u username:password <request>
   ```

   If KIE Server is configured with Red Hat Single Sign-On, the request must include a bearer token:

```
curl -H "Authorization: bearer $TOKEN" <request>
```

3. Specify the format of the request and response. The REST API endpoints work with XML format and are set using request headers:

   XML

   ```
   curl -H "accept: application/xml" -H "content-type: application/xml"
   ```

4. Execute the test scenario:
   [POST] server/containers/{containerId}/scesim

   Example curl request:

   ```
   curl -X POST "http://localhost:8080/kie-server/services/rest/server/containers/traffic_1.0.0-SNAPSHOT/scesim"\ -u 'wbadmin:wbadmin;' \ -H "accept: application/xml" -H "content-type: application/xml"\ -d @Violation.scesim
   ```

   Example XML request:

```
<ScenarioSimulationModel version="1.8">
 <simulation>
  <scesimModelDescriptor>
   <factMappings>
    <FactMapping>
     <expressionElements/>
     <expressionIdentifier>
      <name>Index</name>
      <type>OTHER</type>
     </expressionIdentifier>
     <factIdentifier>
      <name>#</name>
      <className>java.lang.Integer</className>
     </factIdentifier>
     <className>java.lang.Integer</className>
     <factAlias>#</factAlias>
     <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
     <columnWidth>70.0</columnWidth>
    </FactMapping>
    <FactMapping>
     <expressionElements/>
     <expressionIdentifier>
      <name>Description</name>
      <type>OTHER</type>
     </expressionIdentifier>
     <factIdentifier>
      <name>Scenario description</name>
      <className>java.lang.String</className>
     </factIdentifier>
     <className>java.lang.String</className>
     <factAlias>Scenario description</factAlias>
     <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
     <columnWidth>300.0</columnWidth>
    </FactMapping>
```

```
<FactMapping>
  <expressionElements>
    <ExpressionElement>
      <step>Driver</step>
    </ExpressionElement>
    <ExpressionElement>
      <step>Points</step>
    </ExpressionElement>
  </expressionElements>
  <expressionIdentifier>
    <name>0|1</name>
    <type>GIVEN</type>
  </expressionIdentifier>
  <factIdentifier>
    <name>Driver</name>
    <className>Driver</className>
  </factIdentifier>
  <className>number</className>
  <factAlias>Driver</factAlias>
  <expressionAlias>Points</expressionAlias>
  <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
  <columnWidth>114.0</columnWidth>
</FactMapping>
<FactMapping>
  <expressionElements>
    <ExpressionElement>
      <step>Violation</step>
    </ExpressionElement>
    <ExpressionElement>
      <step>Type</step>
    </ExpressionElement>
  </expressionElements>
  <expressionIdentifier>
    <name>0|6</name>
    <type>GIVEN</type>
  </expressionIdentifier>
  <factIdentifier>
    <name>Violation</name>
    <className>Violation</className>
  </factIdentifier>
  <className>Type</className>
  <factAlias>Violation</factAlias>
  <expressionAlias>Type</expressionAlias>
  <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
  <columnWidth>114.0</columnWidth>
</FactMapping>
<FactMapping>
  <expressionElements>
    <ExpressionElement>
      <step>Violation</step>
    </ExpressionElement>
    <ExpressionElement>
      <step>Speed Limit</step>
    </ExpressionElement>
  </expressionElements>
  <expressionIdentifier>
```

```
      <name>0|7</name>
      <type>GIVEN</type>
    </expressionIdentifier>
    <factIdentifier>
      <name>Violation</name>
      <className>Violation</className>
    </factIdentifier>
    <className>number</className>
    <factAlias>Violation</factAlias>
    <expressionAlias>Speed Limit</expressionAlias>
    <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
    <columnWidth>114.0</columnWidth>
  </FactMapping>
  <FactMapping>
    <expressionElements>
      <ExpressionElement>
        <step>Violation</step>
      </ExpressionElement>
      <ExpressionElement>
        <step>Actual Speed</step>
      </ExpressionElement>
    </expressionElements>
    <expressionIdentifier>
      <name>0|8</name>
      <type>GIVEN</type>
    </expressionIdentifier>
    <factIdentifier>
      <name>Violation</name>
      <className>Violation</className>
    </factIdentifier>
    <className>number</className>
    <factAlias>Violation</factAlias>
    <expressionAlias>Actual Speed</expressionAlias>
    <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
    <columnWidth>114.0</columnWidth>
  </FactMapping>
  <FactMapping>
    <expressionElements>
      <ExpressionElement>
        <step>Fine</step>
      </ExpressionElement>
      <ExpressionElement>
        <step>Points</step>
      </ExpressionElement>
    </expressionElements>
    <expressionIdentifier>
      <name>0|11</name>
      <type>EXPECT</type>
    </expressionIdentifier>
    <factIdentifier>
      <name>Fine</name>
      <className>Fine</className>
    </factIdentifier>
    <className>number</className>
    <factAlias>Fine</factAlias>
    <expressionAlias>Points</expressionAlias>
```

```
      <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
      <columnWidth>114.0</columnWidth>
    </FactMapping>
    <FactMapping>
      <expressionElements>
        <ExpressionElement>
          <step>Fine</step>
        </ExpressionElement>
        <ExpressionElement>
          <step>Amount</step>
        </ExpressionElement>
      </expressionElements>
      <expressionIdentifier>
        <name>0|12</name>
        <type>EXPECT</type>
      </expressionIdentifier>
      <factIdentifier>
        <name>Fine</name>
        <className>Fine</className>
      </factIdentifier>
      <className>number</className>
      <factAlias>Fine</factAlias>
      <expressionAlias>Amount</expressionAlias>
      <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
      <columnWidth>114.0</columnWidth>
    </FactMapping>
    <FactMapping>
      <expressionElements>
        <ExpressionElement>
          <step>Should the driver be suspended?</step>
        </ExpressionElement>
      </expressionElements>
      <expressionIdentifier>
        <name>0|13</name>
        <type>EXPECT</type>
      </expressionIdentifier>
      <factIdentifier>
        <name>Should the driver be suspended?</name>
        <className>Should the driver be suspended?</className>
      </factIdentifier>
      <className>string</className>
      <factAlias>Should the driver be suspended?</factAlias>
      <expressionAlias>value</expressionAlias>
      <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
      <columnWidth>114.0</columnWidth>
    </FactMapping>
  </factMappings>
</scesimModelDescriptor>
<scesimData>
  <Scenario>
    <factMappingValues>
      <FactMappingValue>
        <factIdentifier>
          <name>Scenario description</name>
          <className>java.lang.String</className>
        </factIdentifier>
```

```
    <expressionIdentifier>
      <name>Description</name>
      <type>OTHER</type>
    </expressionIdentifier>
    <rawValue class="string">Above speed limit: 10km/h and 30 km/h</rawValue>
  </FactMappingValue>
  <FactMappingValue>
    <factIdentifier>
      <name>Driver</name>
      <className>Driver</className>
    </factIdentifier>
    <expressionIdentifier>
      <name>0|1</name>
      <type>GIVEN</type>
    </expressionIdentifier>
    <rawValue class="string">10</rawValue>
  </FactMappingValue>
  <FactMappingValue>
    <factIdentifier>
      <name>Violation</name>
      <className>Violation</className>
    </factIdentifier>
    <expressionIdentifier>
      <name>0|6</name>
      <type>GIVEN</type>
    </expressionIdentifier>
    <rawValue class="string">&amp;quot;speed&amp;quot;</rawValue>
  </FactMappingValue>
  <FactMappingValue>
    <factIdentifier>
      <name>Violation</name>
      <className>Violation</className>
    </factIdentifier>
    <expressionIdentifier>
      <name>0|7</name>
      <type>GIVEN</type>
    </expressionIdentifier>
    <rawValue class="string">100</rawValue>
  </FactMappingValue>
  <FactMappingValue>
    <factIdentifier>
      <name>Violation</name>
      <className>Violation</className>
    </factIdentifier>
    <expressionIdentifier>
      <name>0|8</name>
      <type>GIVEN</type>
    </expressionIdentifier>
    <rawValue class="string">120</rawValue>
  </FactMappingValue>
  <FactMappingValue>
    <factIdentifier>
      <name>Fine</name>
      <className>Fine</className>
    </factIdentifier>
    <expressionIdentifier>
```

```
            <name>0|11</name>
            <type>EXPECT</type>
          </expressionIdentifier>
          <rawValue class="string">3</rawValue>
        </FactMappingValue>
        <FactMappingValue>
          <factIdentifier>
            <name>Fine</name>
            <className>Fine</className>
          </factIdentifier>
          <expressionIdentifier>
            <name>0|12</name>
            <type>EXPECT</type>
          </expressionIdentifier>
          <rawValue class="string">500</rawValue>
        </FactMappingValue>
        <FactMappingValue>
          <factIdentifier>
            <name>Should the driver be suspended?</name>
            <className>Should the driver be suspended?</className>
          </factIdentifier>
          <expressionIdentifier>
            <name>0|13</name>
            <type>EXPECT</type>
          </expressionIdentifier>
          <rawValue class="string">&amp;quot;No&amp;quot;</rawValue>
        </FactMappingValue>
        <FactMappingValue>
          <factIdentifier>
            <name>#</name>
            <className>java.lang.Integer</className>
          </factIdentifier>
          <expressionIdentifier>
            <name>Index</name>
            <type>OTHER</type>
          </expressionIdentifier>
          <rawValue class="string">1</rawValue>
        </FactMappingValue>
      </factMappingValues>
    </Scenario>
  </scesimData>
</simulation>
<background>
  <scesimModelDescriptor>
    <factMappings>
      <FactMapping>
        <expressionElements/>
        <expressionIdentifier>
          <name>1|1</name>
          <type>GIVEN</type>
        </expressionIdentifier>
        <factIdentifier>
          <name>Empty</name>
          <className>java.lang.Void</className>
        </factIdentifier>
        <className>java.lang.Void</className>
```

```
          <factAlias>Instance 1</factAlias>
          <expressionAlias>PROPERTY 1</expressionAlias>
          <factMappingValueType>NOT_EXPRESSION</factMappingValueType>
          <columnWidth>114.0</columnWidth>
        </FactMapping>
      </factMappings>
    </scesimModelDescriptor>
    <scesimData>
      <BackgroundData>
        <factMappingValues>
          <FactMappingValue>
            <factIdentifier>
              <name>Empty</name>
              <className>java.lang.Void</className>
            </factIdentifier>
            <expressionIdentifier>
              <name>1|1</name>
              <type>GIVEN</type>
            </expressionIdentifier>
          </FactMappingValue>
        </factMappingValues>
      </BackgroundData>
    </scesimData>
  </background>
  <settings>
    <dmnFilePath>src/main/resources/org/kie/example/traffic/traffic_violation/Traffic
Violation.dmn</dmnFilePath>
    <type>DMN</type>
    <fileName></fileName>
    <dmnNamespace>https://github.com/kiegroup/drools/kie-dmn/_A4BCA8B8-CF08-433F-
93B2-A2598F19ECFF</dmnNamespace>
    <dmnName>Traffic Violation</dmnName>
    <skipFromBuild>false</skipFromBuild>
    <stateless>false</stateless>
  </settings>
  <imports>
    <imports/>
  </imports>
</ScenarioSimulationModel>
```

Example XML response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response type="SUCCESS" msg="Test Scenario successfully executed">
    <scenario-simulation-result>
        <run-count>5</run-count>
        <ignore-count>0</ignore-count>
        <run-time>31</run-time>
    </scenario-simulation-result>
</response>
```

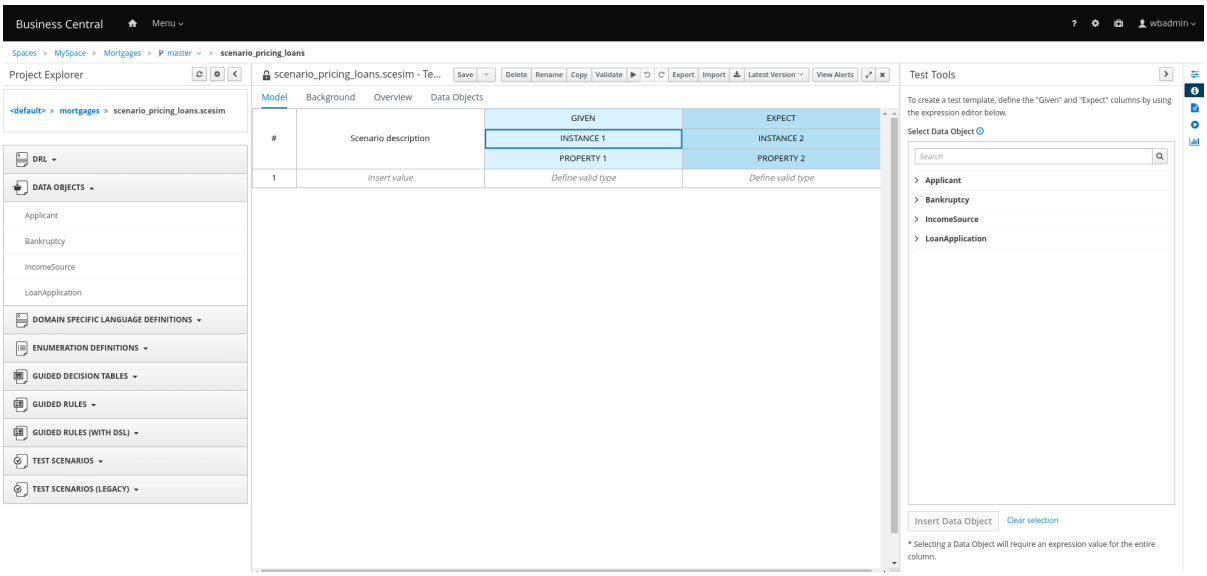
CHAPTER 15. CREATING TEST SCENARIO USING THE SAMPLE MORTGAGES PROJECT

This chapter illustrates creating and executing a test scenario from the sample **Mortgages** project shipped with Business Central using the test scenario designer. The test scenario example in this chapter is based on the **Pricing loans** guided decision table from the **Mortgages** project.

Procedure

1. In Business Central, go to **Menu → Design → Projects** and click **Mortgages**.
2. If the project is not listed under **Projects**, from **MySpace**, click **Try Samples → Mortgages → OK**.
   The Assets window appears.
3. Click **Add Asset → Test Scenario**.
4. Enter **scenario_pricing_loans** as the **Test Scenario** name and select the default **mortgages.mortgages** package from the **Package** drop-down list.
   The package you select must contain all the required rule assets.
5. Select **RULE** as the **Source type**.
6. Click **Ok** to create and open the test scenario in the test scenario designer.
7. Expand **Project Explorer** and verify the following:
   - **Applicant**, **Bankruptcy**, **IncomeSource**, and **LoanApplication** data objects exist.
   - **Pricing loans** guided decision table exists.
   - Verify that the new test scenario is listed under **Test Scenario**
8. After verifying that everything is in place, return to the **Model** tab of the test scenario designer and define the **GIVEN** and **EXPECT** data for the scenario, based on the available data objects.

Figure 15.1. A blank test scenario designer



9. Define the **GIVEN** column details:

a. Click the cell named **INSTANCE 1** under the **GIVEN** column header.

b. From the **Test Tools** panel, select the **LoanApplication** data object.

c. Click **Insert Data Object**.

10. To create properties for the data object, right-click the property header cell and select **Insert column right** or **Insert column left** as required. For this example, you need to create two more property cells under the **GIVEN** column.

11. Select the first property header cell:

a. From the **Test Tools** panel, select and expand the **LoanApplication** data object.

b. Click **amount**.

c. Click **Insert Data Object** to map the data object field to the property header cell.

12. Select the second property header cell:

a. From the **Test Tools** panel, select and expand the **LoanApplication** data object.

b. Click **deposit**.

c. Click **Insert Data Object**.

13. Select the third property header cell:

a. From the **Test Tools** panel, select and expand the **LoanApplication** data object.

b. Click **lengthYears**

c. Click **Insert Data Object**.

14. Right-click the **LoanApplication** header cell and select **Insert column right**. A new **GIVEN** column to the right is created.

15. Select the new header cell:

a. From the **Test Tools** panel, select the **IncomeSource** data object.

b. Click **Insert Data Object** to map the data object to the header cell.

16. Select the property header cell below **IncomeSource**:

a. From the **Test Tools** panel, select and expand the **IncomeSource** data object.

b. Click **type**.

c. Click **Insert Data Object** to map the data object field to the property header cell. You have now defined all the **GIVEN** column cells.

17. Next, define the **EXPECT** column details:

a. Click the cell named **INSTANCE 2** under the **EXPECT** column header.

b. From the **Test Tools** panel, select **LoanApplication** data object.

c. Click **Insert Data Object**.

18. To create properties for the data object, right-click the property header cell and select **Insert column right** or **Insert column left** as required. Create two more property cells under the **EXPECT** column.

19. Select the first property header cell:

    a. From the **Test Tools** panel, select and expand the **LoanApplication** data object.

    b. Click **approved**.

    c. Click **Insert Data Object** to map the data object field to the property header cell.

20. Select the second property header cell:

    a. From the **Test Tools** panel, select and expand the **LoanApplication** data object.

    b. Click **insuranceCost**.

    c. Click **Insert Data Object** to map the data object field to the property header cell.

21. Select the third property header cell:

    a. From the **Test Tools** panel, select and expand the **LoanApplication** data object.

    b. Click **approvedRate**.

    c. Click **Insert Data Object** to map the data object field to the property header cell.

22. To define the test scenario, enter the following data in the first row:

    - Enter **Row 1 test scenario** as the **Scenario Description**, **150000** as the **amount**, **19000** as the **deposit**, **30** as the **lengthYears**, and **Asset** as the **type** for the **GIVEN** column values.

    - Enter **true** as **approved**, **0** as the **insuranceCost** and **2** as the **approvedRate** for the **EXPECT** column values.

23. Next enter the following data in the second row:

    - Enter **Row 2 test scenario** as the **Scenario Description**, **100002** as the **amount**, **2999** as the **deposit**, **20** as the **lengthYears**, and **Job** as the **type** for the **GIVEN** column values.

    - Enter **true** as **approved**, **10** as the **insuranceCost** and **6** as the **approvedRate** for the **EXPECT** column values.

24. After you have defined all **GIVEN**, **EXPECT**, and other data for the scenario, click **Save** in the test scenario designer to save your work.

25. Click **Run Test** in the upper-right corner to run the **.scesim** file.
    The test result is displayed in the **Test Report** panel. Click **View Alerts** to display messages from the **Alerts** section. If a test fails, refer to the messages in the **Alerts** section at the bottom of the window, review and correct all components in the scenario, and try again to validate the scenario until the scenario passes.

26. Click **Save** in the test scenario designer to save your work after you have made all necessary changes.

CHAPTER 16. TEST SCENARIOS (LEGACY) DESIGNER IN BUSINESS CENTRAL

Red Hat Process Automation Manager currently supports both the new **Test Scenarios** designer and the former **Test Scenarios (Legacy)** designer. The default designer is the new test scenarios designer, which supports testing of both rules and DMN models and provides an enhanced overall user experience with test scenarios. If required, you can continue to use the legacy test scenarios designer, which supports rule-based test scenarios only.

16.1. CREATING AND RUNNING A TEST SCENARIO (LEGACY)

You can create test scenarios in Business Central to test the functionality of business rule data before deployment. A basic test scenario must have at least the following data:

- Related data objects
- **GIVEN** facts
- **EXPECT** results

NOTE

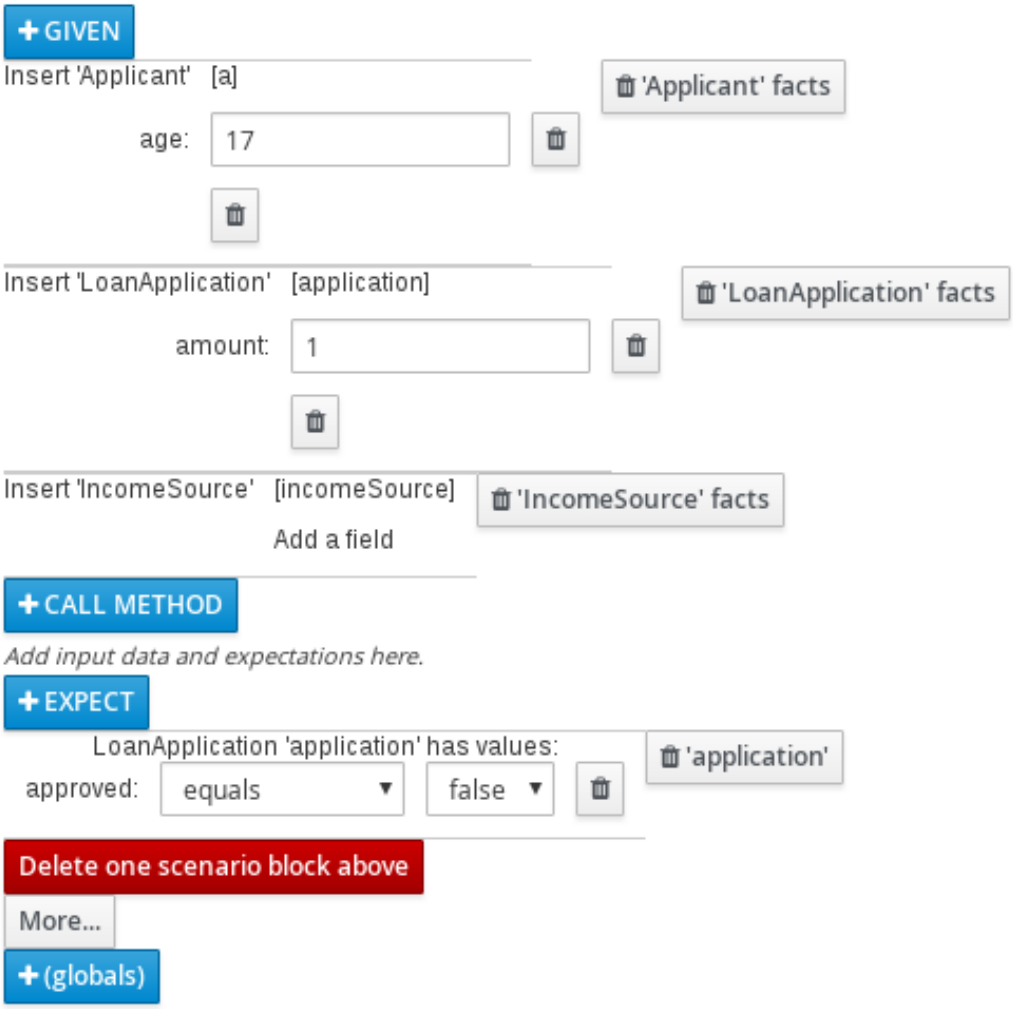
The legacy test scenarios designer supports the **LocalDate** java built-in data type. You can use the **LocalDate** java built-in data type in the **dd-mmm-yyyy** date format. For example, you can set this in the **17-Oct-2020** date format.

With this data, the test scenario can validate the expected and actual results for that rule instance based on the defined facts. You can also add a **CALL METHOD** and any available **globals** to a test scenario, but these scenario settings are optional.

Procedure

1. In Business Central, go to **Menu → Design → Projects** and click the project name.
2. Click **Add Asset → Test Scenarios (Legacy)**.
3. Enter an informative **Test Scenario** name and select the appropriate **Package**. The package that you specify must be the same package where the required rule assets have been assigned or will be assigned. You can import data objects from any package into the asset's designer.
4. Click **Ok** to create the test scenario.
   The new test scenario is now listed in the **Test Scenarios** panel of the **Project Explorer**,
5. Click the **Data Objects** tab to verify that all data objects required for the rules that you want to test are listed. If not, click **New item** to import the needed data objects from other packages, or create data objects within your package.
6. After all data objects are in place, return to the **Model** tab of the test scenarios designer and define the **GIVEN** and **EXPECT** data for the scenario, based on the available data objects.

Figure 16.1. The test scenarios designer



The **GIVEN** section defines the input facts for the test. For example, if an **Underage** rule in the project declines loan applications for applicants under the age of 21, then the **GIVEN** facts in the test scenario could be **Applicant** with **age** set to some integer less than 21.

The **EXPECT** section defines the expected results based on the **GIVEN** input facts. That is, **GIVEN** the input facts, **EXPECT** these other facts to be valid or entire rules to be activated. For example, with the given facts of an applicant under the age of 21 in the scenario, the **EXPECT** results could be **LoanApplication** with **approved** set to **false** (as a result of the underage applicant), or could be the activation of the **Underage** rule as a whole.

7. Optionally, add a **CALL METHOD** and any **globals** to the test scenario:

   - **CALL METHOD:** Use this to invoke a method from another fact when the rule execution is initiated. Click **CALL METHOD**, select a fact, and click to select the method to invoke. You can invoke any Java class methods (such as methods from an ArrayList) from the Java library or from a JAR that was imported for the project (if applicable).

   - **globals:** Use this to add any global variables in the project that you want to validate in the test scenario. Click **globals** to select the variable to be validated, and then in the test scenarios designer, click the global name and define field values to be applied to the global variable. If no global variables are available, then they must be created as new assets in Business Central. Global variables are named objects that are visible to the decision engine but are different from the objects for facts. Changes in the object of a global do not trigger the re-evaluation of rules.

8. Click **More** at the bottom of the test scenarios designer to add other data blocks to the same scenario file as needed.

9. After you have defined all **GIVEN**, **EXPECT**, and other data for the scenario, click **Save** in the test scenarios designer to save your work.

10. Click **Run scenario** in the upper-right corner to run this **.scenario** file, or click **Run all scenarios** to run all saved **.scenario** files in the project package (if there are multiple). Although the **Run scenario** option does not require the individual **.scenario** file to be saved, the **Run all scenarios** option does require all **.scenario** files to be saved.
If the test fails, address any problems described in the **Alerts** message at the bottom of the window, review all components in the scenario, and try again to validate the scenario until the scenario passes.

11. Click **Save** in the test scenarios designer to save your work after all changes are complete.

16.1.1. Adding GIVEN facts in test scenarios (legacy)

The **GIVEN** section defines input facts for the test. For example, if an **Underage** rule in the project declines loan applications for applicants under the age of 21, then the **GIVEN** facts in the test scenario could be **Applicant** with **age** set to some integer less than 21.

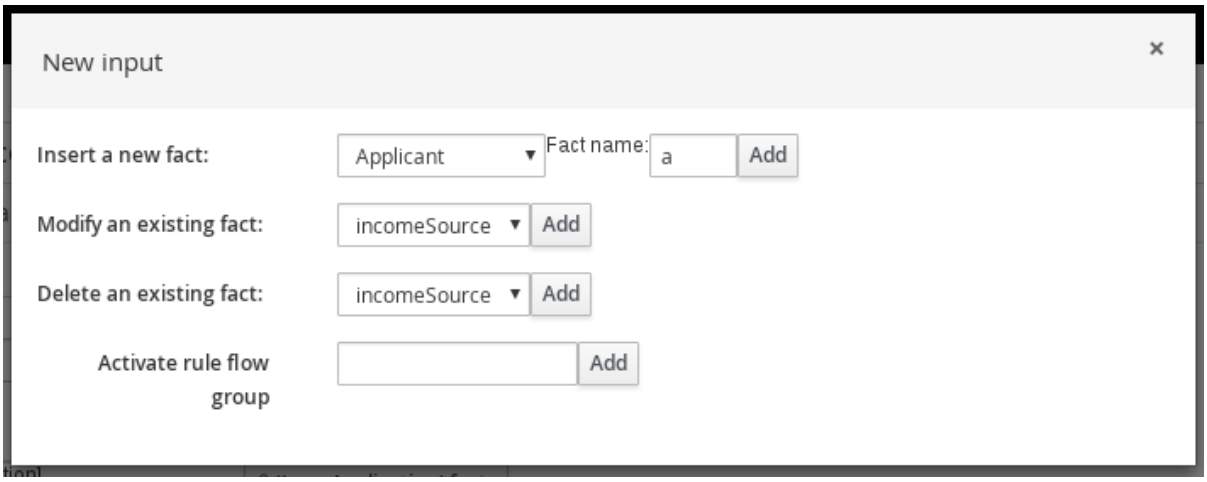
Prerequisites

- All data objects required for your test scenario have been created or imported and are listed in the **Data Objects** tab of the **Test Scenarios (Legacy)** designer.

Procedure

1. In the **Test Scenarios (Legacy)** designer, click **GIVEN** to open the **New input** window with the available facts.

Figure 16.2. Add GIVEN input to the test scenario



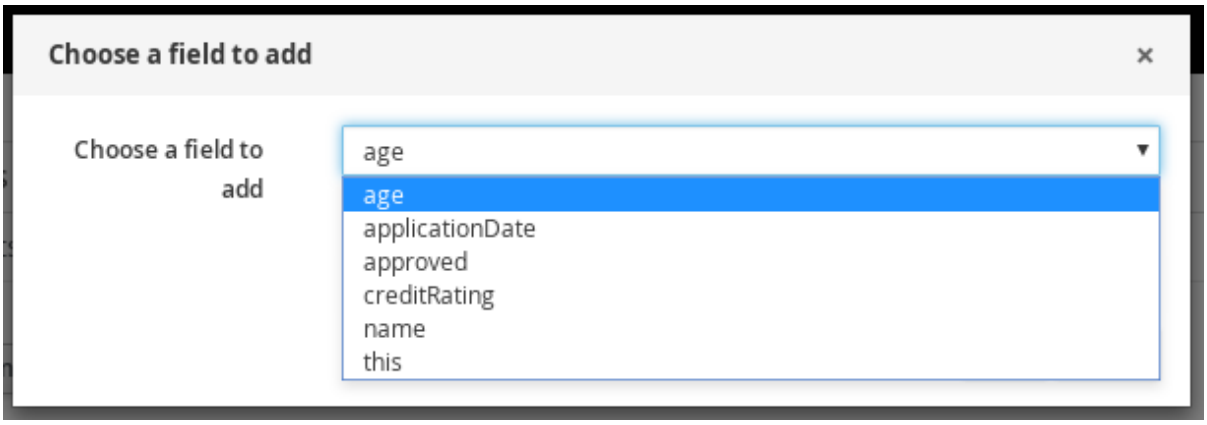
The list includes the following options, depending on the data objects available in the **Data Objects** tab of the test scenarios designer:

- **Insert a new fact:** Use this to add a fact and modify its field values. Enter a variable for the fact as the **Fact name**.

- **Modify an existing fact:** (Appears only after another fact has been added.) Use this to specify a previously inserted fact to be modified in the decision engine between executions of the scenario.
- **Delete an existing fact:** (Appears only after another fact has been added.) Use this to specify a previously inserted fact to be deleted from the decision engine between executions of the scenario.
- **Activate rule flow group:** Use this to specify a rule flow group to be activated so that all rules within that group can be tested.

2. Choose a fact for the desired input option and click **Add**. For example, set **Insert a new fact:** to **Applicant** and enter **a** or **app** or any other variable for the **Fact name**.
3. Click the fact in the test scenarios designer and select the field to be modified.

Figure 16.3. Modify a fact field



4. Click the edit icon ( ) and select from the following field values:
   - **Literal value:** Creates an open field in which you enter a specific literal value.
   - **Bound variable:** Sets the value of the field to the fact bound to a selected variable. The field type must match the bound variable type.
   - **Create new fact:** Enables you to create a new fact and assign it as a field value of the parent fact. Then you can click the child fact in the test scenarios designer and likewise assign field values or nest other facts similarly.
5. Continue adding any other **GIVEN** input data for the scenario and click **Save** in the test scenarios designer to save your work.

16.1.2. Adding EXPECT results in test scenarios (legacy)

The **EXPECT** section defines the expected results based on the **GIVEN** input facts. That is, **GIVEN** the input facts, **EXPECT** other specified facts to be valid or entire rules to be activated. For example, with the given facts of an applicant under the age of 21 in the scenario, the **EXPECT** results could be **LoanApplication** with **approved** set to **false** (as a result of the underage applicant), or could be the activation of the **Underage** rule as a whole.

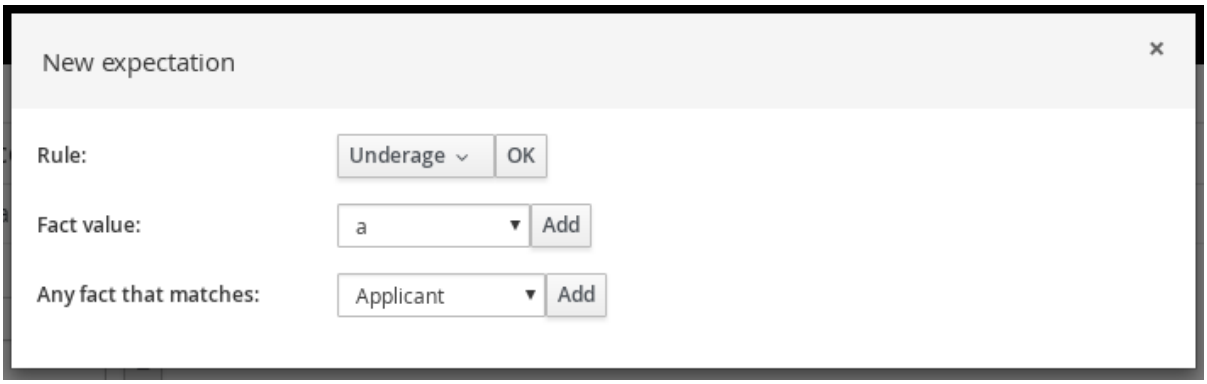
Prerequisites

- All data objects required for your test scenario have been created or imported and are listed in the **Data Objects** tab of the **Test Scenarios (Legacy)** designer.

Procedure

1. In the **Test Scenarios (Legacy)** designer, click **EXPECT** to open the **New expectation** window with the available facts.

   Figure 16.4. Add EXPECT results to the test scenario

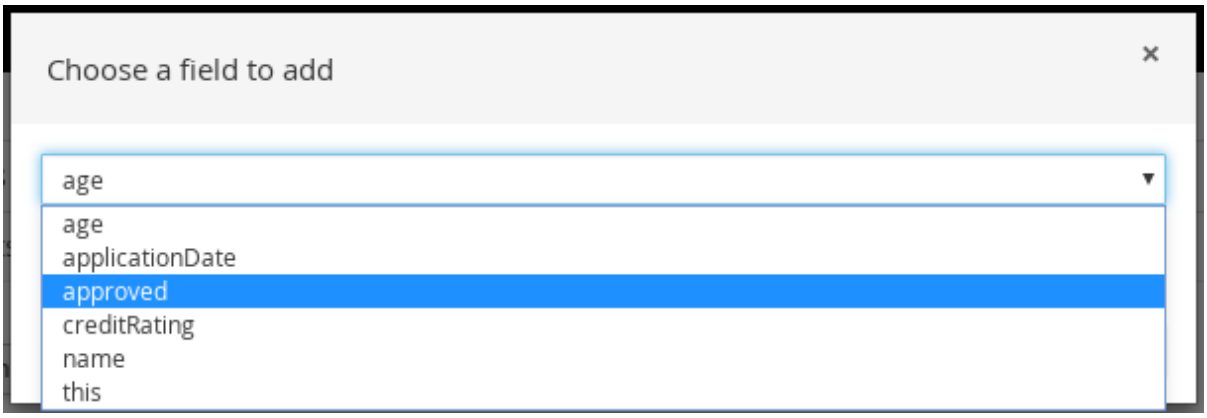
   The list includes the following options, depending on the data in the **GIVEN** section and the data objects available in the **Data Objects** tab of the test scenarios designer:

   - **Rule:** Use this to specify a particular rule in the project that is expected to be activated as a result of the **GIVEN** input. Type the name of a rule that is expected to be activated or select it from the list of rules, and then in the test scenarios designer, specify the number of times the rule should be activated.
   - **Fact value:** Use this to select a fact and define values for it that are expected to be valid as a result of the facts defined in the **GIVEN** section. The facts are listed by the **Fact name** previously defined for the **GIVEN** input.
   - **Any fact that matches:** Use this to validate that at least one fact with the specified values exists as a result of the **GIVEN** input.

2. Choose a fact for the desired expectation (such as **Fact value: application**) and click **Add** or **OK**.

3. Click the fact in the test scenarios designer and select the field to be added and modified.

   Figure 16.5. Modify a fact field

4. Set the field values to what is expected to be valid as a result of the **GIVEN** input (such as **approved** | **equals** | **false**).

NOTE

In the legacy test scenarios designer, you can use **["value1", "value2"]** string format in the **EXPECT** field to validate the list of strings.

5. Continue adding any other **EXPECT** input data for the scenario and click **Save** in the test scenarios designer to save your work.

6. After you have defined and saved all **GIVEN**, **EXPECT**, and other data for the scenario, click **Run scenario** in the upper-right corner to run this **.scenario** file, or click **Run all scenarios** to run all saved **.scenario** files in the project package (if there are multiple). Although the **Run scenario** option does not require the individual **.scenario** file to be saved, the **Run all scenarios** option does require all **.scenario** files to be saved.
   If the test fails, address any problems described in the **Alerts** message at the bottom of the window, review all components in the scenario, and try again to validate the scenario until the scenario passes.

7. Click **Save** in the test scenarios designer to save your work after all changes are complete.

CHAPTER 17. FEATURE COMPARISON OF LEGACY AND NEW TEST SCENARIO DESIGNER

Red Hat Process Automation Manager supports both the new test scenario designer and the former test scenario (Legacy) designer.

The default designer is the new test scenario designer, which supports testing of both rules and DMN models, and provides an enhanced overall user experience with test scenarios. You can continue to use the legacy test scenario designer, which only supports rule-based test scenarios.

IMPORTANT

The new test scenario designer has an improved layout and feature set and continues to be developed. However, the legacy test scenario designer is deprecated with Red Hat Process Automation Manager 7.3.0 and will be removed in a future Red Hat Process Automation Manager release.

The following table highlights the main features of legacy and new test scenario designer, which are supported in Red Hat Process Automation Manager to help you decide a suitable test scenario designer in your project.

- **+** indicates that the feature is present in the test scenario designer.
- **-** indicates that the feature is not present in the test scenario designer.

Table 17.1. Main features of legacy and new test scenario designer

Feature & highlights	New designer	Legacy designer	Documentation
Creating and running a test scenario • You can create test scenarios in Business Central to test the functionality of business rule data before deployment. • A basic test scenario must have at least a related data objects, **GIVEN** facts, and **EXPECT** results. • You can run the tests to validate your business rules and data.	+	+	• For more information about creating rule and DMN-based test scenarios, see Chapter 4, *Test scenario template*. • For more information about running the test scenarios, see Chapter 10, *Running the test scenarios*. • For more information about creating and running test scenarios (legacy), see Section 16.1, "Creating and running a test scenario (legacy)".

Feature & highlights	New designer	Legacy designer	Documentation
Adding GIVEN facts in test scenarios • You can insert and verify the **GIVEN** facts for the test.	+	+	• For more information about adding **GIVEN** facts in new test scenario designer, see Chapter 4, *Test scenario template*. • For more information about adding **GIVEN** facts in test scenarios (legacy), see Section 16.1.1, "Adding GIVEN facts in test scenarios (legacy)".
Adding EXPECT results in test scenarios • The **EXPECT** section defines the expected results based on the **GIVEN** input facts. • It represents the objects and their fields whose exact values are checked based on the provided information.	+	+	• For more information about adding **EXPECT** results in new test scenario designer, see Chapter 4, *Test scenario template*. • For more information about adding **EXPECT** results in test scenarios (legacy), see Section 16.1.2, "Adding EXPECT results in test scenarios (legacy)".
KIE session • You can set KIE session on test scenario level settings.	+	+	NA
KIE base on test scenario level • You can set KIE base on test scenario level settings.	-	+	NA
KIE base on project level • You can set KIE base on project level settings.	+	+	NA
Simulated date and time • You can set a simulated date and time for the legacy test scenario designer.	-	+	NA

Feature & highlights	New designer	Legacy designer	Documentation
Rule flow group • You can specify a rule flow group to be activated to test all the rules within that group.	+	+	• For more information about setting rule flow group in new test scenarios, see Section 3.9.1, "Configuring global settings for rule-based test scenarios". • For more information about setting rule flow group in test scenarios (legacy), Section 16.1.1, "Adding GIVEN facts in test scenarios (legacy)".
Global variables • Global variables are named objects that are visible to the decision engine but are different from the objects for facts. • Setting global variables for new test scenario is deprecated . • In case you want to reuse data sets for different scenarios, you can use the **Background** instance.	-	+	• For more information about **Background** instance in new test scenarios, see Chapter 7, *Background instance in test scenarios*. • For more information about global variables in test scenarios (legacy), see Section 16.1, "Creating and running a test scenario (legacy)".
Call method • You can use this to invoke a method from another fact when the rule execution is initiated. • You can invoke any Java class methods from the Java library or from a JAR that was imported for the project.	+	+	• For more information about calling a method in new test scenarios, see Chapter 9, *Expression syntax in test scenarios*. • For more information about calling a method in test scenarios (legacy), see Section 16.1, "Creating and running a test scenario (legacy)".
Modify an existing fact • You can modify a previously inserted fact in the decision engine between executions of the scenario.	-	+	For more information about modifying an existing fact in test scenarios (legacy), see Section 16.1.1, "Adding GIVEN facts in test scenarios (legacy)".

Feature & highlights	New designer	Legacy designer	Documentation
Bound variable • You can set the value of a field to the fact bound to a selected variable. • In the new test scenario designer, you can not define a variable inside a test scenario grid and reuse it inside **GIVEN** or **EXPECTED** cells.	-	+	For more information about how to set bound variables in test scenarios (legacy), see Section 16.1.1, "Adding GIVEN facts in test scenarios (legacy)".

CHAPTER 18. NEXT STEPS

Packaging and deploying a Red Hat Process Automation Manager project

APPENDIX A. VERSIONING INFORMATION

Documentation last updated on Tuesday, August 04, 2020.