Red Hat Process Automation Manager 7.7

テストシナリオを使用したデシジョンサービスのテスト

Last Updated: 2020-04-01

Red Hat Process Automation Manager 7.7 テストシナリオを使用したデシジョンサービスのテスト

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

法律上の通知

概要

本書は、Red Hat Process Automation Manager 7.7 で、テストシナリオを使用してデシジョンサービスをテストする方法を説明します。

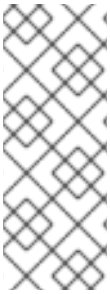
目次

前書き

ビジネス分析者またはビジネスルールの開発者は、Business Central でテストシナリオを使用して、プロジェクトをデプロイする前にデシジョンサービスをテストできます。DMN ベースおよびルールベースのデシジョンサービスをテストすると、プロジェクトのルールアセットが適切に、想定通りに機能することが確認できます。またデシジョンサービスは、プロジェクトの開発時にいつでもテストできます。

前提条件

- デシジョンサービスのスペースおよびプロジェクトが Business Central に作成されている。詳細は『**デシジョンサービスのスタートガイド**』を参照してください。
- ルールベースのデシジョンサービスに、ビジネスルールおよび関連するデータオブジェクトが定義されている。詳細は『**ガイド付きデシジョンテーブルを使用したデシジョンサービスの作成**』を参照してください。
- DMN ベースのデシジョンサービスに、DMN デシジョンロジックとその関連のカスタムデータタイプが定義されている。詳細は、『**DMN モデルを使用したデシジョンサービスの作成**』を参照してください。



注記

テストシナリオでは、ビジネスルールを構成するように定義したデータをテストできるため、ビジネスルールを先に定義しておくことは、テストシナリオにおける技術的な前提条件ではありません。ただし、先にルールを作成しておくと、テストシナリオでルール全体をテストでき、かつ意図するデシジョンサービスにシナリオがより近づくため便利です。DMN ベースの場合には、テストシナリオを使用することで、DMN デシジョンロジックとその関連のカスタムデータタイプがデシジョンサービス用に定義されます。

第1章 テストシナリオ

Red Hat Process Automation Manager のテストシナリオでは、ビジネスルールを実稼働環境にデプロイする前に、(ルールベースのテストシナリオの場合) ビジネスルールの機能とデータの妥当性、および (DMN ベースのテストシナリオの場合) DMN モデルを検証できます。このテストシナリオでは、プロジェクトのデータを使用して、指定した条件と、定義した 1 つ以上のビジネスルールで想定される結果を設定できます。シナリオを実行する際は、想定した結果と、ルールのインスタンスから実際に得られた結果を比較します。想定した結果が実際の結果に一致するとテストに成功し、一致しない場合はテストに失敗します。

Red Hat Process Automation Manager は現在、新規の **テストシナリオ** デザイナーと以前の **テストシナリオ (レガシー)** デザイナーの両方をサポートします。デフォルトのデザイナh、新規のテストシナリオデザイナーで、ルールと DMN モデルのテストをサポートし、テストシナリオの全体的な使用感が改善されています。必要に応じて、レガシーのテストシナリオをそのまま使用することができますが、ルールベースのテストシナリオしかサポートされません。

プロジェクトレベルや、特定のシナリオアセット内で利用可能なテストシナリオを実行するなど、複数の方法で定義済みのテストシナリオを実行できます。テストシナリオは独立しており、他のテストシナリオに影響を与えたり、テストシナリオを変更したりできません。Business Central でのプロジェクト開発中であればいつでも、テストシナリオを実行できます。テストシナリオの実行に、デシジョンサービスをコンパイルしたり、デプロイしたりする必要はありません。

別のパッケージからのデータオブジェクトは、テストシナリオと同じプロジェクトパッケージにインポートできます。同じパッケージに含まれるアセットはデフォルトでインポートされます。必要なデータオブジェクトとテストシナリオを作成したら、テストシナリオデザイナーの **Data Objects** タブを使用して、必要なデータオブジェクトがすべてリストされていることを検証するか、**アイテムを追加**して既存のデータオブジェクトをインポートします。

重要

テストシナリオのドキュメント全体で、**テストシナリオ** および **テストシナリオデザイナー** に関する言及はすべて、レガシーバージョンと明示的に記載がない限り、新規バージョンを対象としています。

第2章 データオブジェクト

データオブジェクトは、作成するルールアセットの構成要素です。データオブジェクトは、プロジェクトで指定したパッケージに Java オブジェクトとして実装されているカスタムのデータ型です。たとえば、データフィールド **Name**、**Address**、および **DateOfBirth** を使用して **Person** オブジェクトを作成し、ローン申し込みルールに詳細な個人情報を指定できます。このカスタムのデータ型は、アセットとデシジョンサービスがどのデータに基づいているかを指定します。

2.1. データオブジェクトの作成

以下の手順は、データオブジェクトを作成する際の一般的な概要で、特定のビジネスアセットに固有のものではありません。

手順

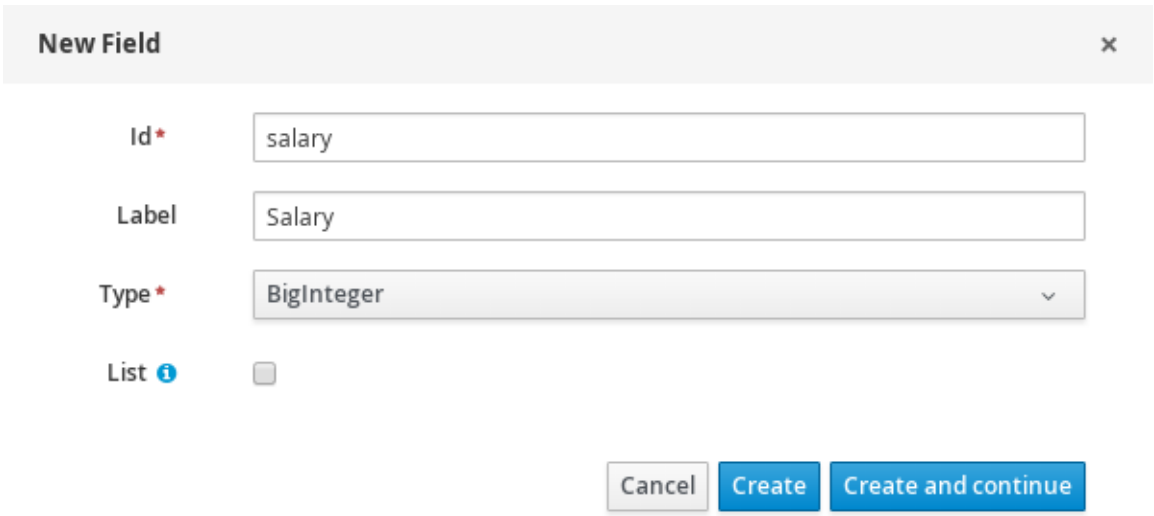
1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → Data Object** をクリックします。
3. 一意の **データオブジェクト** 名を入力し、**パッケージ** を選択します。これにより、その他のルールアセットでもデータオブジェクトを利用できるようになります。同じパッケージに、同じ名前のデータオブジェクトを複数作成することはできません。指定の DRL ファイルで、どのパッケージからでもデータオブジェクトをインポートできます。

別のパッケージからのデータオブジェクトのインポート

別のパッケージから直接、ガイド付きルールやガイド付きデシジョンテーブルデザイナーなどのアセットデザイナーに、既存のデータオブジェクトをインポートすることができます。プロジェクトで関連するルールアセットを選択し、アセットデザイナーで **Data Objects → New item**に移動して、インポートするオブジェクトを選択します。

4. データオブジェクトを永続化するには、**Persistable** チェックボックスを選択します。永続型データオブジェクトは、JPA 仕様に準じてデータベースに保存できます。デフォルトの JPA は Hibernate です。
5. **OK** をクリックします。
6. データオブジェクトデザイナーで **add field** をクリックして、**Id** 属性、**Label** 属性、**Type** 属性を使用するオブジェクトにフィールドを追加します。必須属性にはアスタリスク (*) マークが付いています。
   - **Id:** フィールドの一意の ID を入力します。
   - **Label:** (任意) フィールドのラベルを入力します。
   - **Type:** フィールドのデータタイプ＼を入力します。
   - **List:** (任意) このチェックボックスを選択すると、このフィールドで、指定したタイプのアイテムを複数保持できるようになります。

図2.1 データオブジェクトへのデータフィールドの追加



7. **Create** をクリックして新規フィールドを追加します。**Create and continue** をクリックすると、新しいフィールドが追加され、別のフィールドを引き続き追加できます。

注記

フィールドを編集するには、フィールド行を選択し、画面右側の **general properties** を使用します。

第3章 BUSINESS CENTRAL でのテストシナリオデザイナー

テストシナリオデザイナーは、テーブル形式のレイアウトを提供し、シナリオテンプレートと関連するテストケースすべてを定義できるようにします。デザイナーレイアウトはヘッダーと個別の行を持つテーブルで構成されています。ヘッダーは、**GIVEN** と **EXPECT** の行、インスタンスの行、対応のフィールドの行の 3 つで構成されます。ヘッダーは、テストシナリオテンプレートとしても知られており、個別の行はテストシナリオ定義と呼ばれます。

テストシナリオテンプレートまたはヘッダーは以下の 2 つの部分で構成されます。

- **GIVEN** データオブジェクトおよびそのフィールド: 入力情報を表現します。
- **EXPECT** データオブジェクトおよびそのフィールド: オブジェクトとフィールドを表現します。実際の値が指定の情報をもとにチェックされ、想定の結果を構成するのにも、この値を使用します。

テストシナリオの定義は、個別のテストケーステンプレートを表現します。

Project Explorer にはデザイナーの左パネルからアクセスできますが、右パネルからは **Settings**、**Test Tools**、**Scenario Cheatsheet**、**Test Report**、**Coverage Report** タブにアクセスできます。**Settings** タブにアクセスし、ルールベースおよび DMN ベースのテストシナリオのグローバル設定を表示し、編集できます。**Test Tools** を使用してデータオブジェクトマッピングを設定できます。**Scenario Cheatsheet** タブには参照として使用できるメモとチートシートが含まれています。**Test Report** タブにはテストの概要とシナリオステータスが表示されます。テストカバレッジ統計を表示するには、テストシナリオデザイナーの右側にある **Coverage Report** タブを使用します。

3.1. データオブジェクトのインポート

テストシナリオデザイナーは、テストシナリオと同じパッケージ内に配置されている全データオブジェクトを読み込みます。すべてのデータオブジェクトは、デザイナーの **Data Objects** タブから表示できます。読み込んだデータオブジェクトは、**Test Tools** パネルにも表示されます。

データオブジェクトが変更された場合に (新規データオブジェクトの作成時や、既存のデータオブジェクトの削除時など) デザイナーを終了して、開き直す必要があります。一覧からデータオブジェクトを選択して、フィールドとフィールドタイプを表示します。

テストシナリオとは異なるパッケージに配置されているデータオブジェクトを使用する場合には、先にそのデータオブジェクトをインポートする必要があります。以下の手順に従い、ルールベースのテストシナリオ用にデータオブジェクトをインポートしてください。

注記

DMN ベースのテストシナリオの作成中にデータオブジェクトをインポートできません。DMN ベーステストシナリオは、プロジェクトからのデータオブジェクトを使用せず、DMN ファイルで定義したカスタムのデータタイプを使用します。

手順

1. テストシナリオデザイナーの **Project Explorer** パネルに移動します。
2. **Test Scenario** からテストシナリオを選択します。
3. **Data Objects** タブを選択して、**New Item** をクリックします。
4. **Add import** ウィンドウで、ドロップダウンリストからデータオブジェクトを選択します。

5. **Ok** をクリックしてから **Save** をクリックします。
6. テストシナリオデザイナーを終了して再度開き、データオブジェクトリストから新しいデータオブジェクトを表示します。

3.2. テストシナリオのインポート

プロジェクトビューの **Asset** タブにある **Import Asset** ボタンを使用して、既存のテストシナリオをインポートできます。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. プロジェクトの **Asset** タブから **Import Asset** をクリックします。
3. **Create new Import Asset** ウィンドウで、以下を実行します。
   - インポートセットの名前を入力します。
   - **Package** ドロップダウンリストからパッケージを選択します。
   - **Please select a file to upload**から、**Choose File...** をクリックしてテストシナリオファイルを参照します。
4. ファイルを選択して **Open** をクリックします。
5. **Ok** をクリックすると、テストシナリオデザイナーでテストシナリオが開きます。

3.3. テストシナリオの保存

テストシナリオは、テストシナリオテンプレートの作成時や、テストシナリオ定義時にいつでも保存できます。

手順

1. 右上のテストシナリオデザイナーのツールバーから、**Save** をクリックします。
2. **Confirm Save** ウィンドウで、以下を実行します。
   a. テストシナリオに関するコメントを追加する場合には、**add a comment** をクリックします。
   b. もう一度 **Save** をクリックします。

テストシナリオが正常に保存されたことを示すメッセージが画面に表示されます。

3.4. テストシナリオのコピー

右上のツールバーの **Copy** ボタンを使用して、既存のテストシナリオを同じパッケージまたは別のパッケージにコピーできます。

手順

1. 右上のテストシナリオデザイナーのツールバーから、**Copy** をクリックします。
2. **Make a Copy** ウィンドウで、以下を実行します。
   a. **New Name** フィールドに名前を入力します。
   b. テストシナリオをコピーするパッケージを選択します。
   c. オプション: コメントを追加するには、**add a comment** をクリックします。
   d. **Make a Copy** をクリックします。

テストシナリオが正常にコピーされたことを示すメッセージが画面に表示されます。

3.5. テストシナリオのダウンロード

今後の参考に、またはバックアップとして、ローカルのマシンにテストシナリオのコピーをダウンロードできます。

手順

右上のテストシナリオデザイナーのツールバーで、**Download** をクリックします。

.scesim ファイルがローカルマシンにダウンロードされます。

3.6. テストシナリオのバージョン間の切り替え

Business Central には、テストシナリオのさまざまなバージョン間で切り替える機能があります。シナリオを保存するたびに、シナリオの新しいバージョンが **Latest Versions** に表示されます。この機能を使用するには、一度はテストシナリオファイルを保存しておく必要があります。

手順

1. 右上のテストシナリオデザイナーのツールバーから、**Latest Version** をクリックします。ファイルに複数バージョンがある場合には、ファイルの全バージョンが **Latest Version** に表示されます。
2. 作業するバージョンをクリックします。
   テストシナリオの選択したバージョンがテストシナリオデザイナーで開きます。
3. デザイナーツールバーから **Restore** をクリックします。
4. **Confirm Restore** で以下を実行します。
   a. コメントを追加するには、**add a comment** をクリックします。
   b. **Restore** をクリックして確定します。

選択したバージョンが正常にデザイナーに再読込されたことを示すメッセージが画面に表示されます。

3.7. アラートパネルの表示または非表示

Alerts パネルは、テストシナリオデザイナーまたはプロジェクトビューの下部に表示されます。実行したテストに失敗した場合には、ビルド情報とエラーメッセージが表示されます。

手順

右上のデザイナーツールバーで、**Hide Alerts/View Alerts** をクリックして、レポートパネルを有効または無効にします。

3.8. コンテキストメニューのオプション

テストシナリオデザイナーには、コンテキストメニューオプションがあり、行と列の追加、削除、複製など、テーブルでの基本操作を実行できます。コンテキストメニューを使用するには、テーブル要素を右クリックする必要があります。メニューオプションは、選択するテーブル要素により異なります。

表3.1 コンテキストメニューのオプション

テーブル要素	セルのラベル	利用可能なコンテキストメニューオプション
ヘッダー	# & Scenario description	下に行を挿入
	GIVEN & EXPECT	左端に列を挿入、右端に列を挿入、下に行を挿入
	INSTANCE 1, INSTANCE 2 & PROPERTY 1, PROPERTY 2	左に列を挿入、右に列を挿入、列を削除、インスタンスを複製、下に行を挿入
行	行番号、テストシナリオの説明、またはテストシナリオの定義を含むすべてのセル	上に行を挿入、下に行を挿入、行を複製、行を削除、シナリオを実行

表3.2 テーブルの対話の説明

テーブルの対話	説明
左端に列を挿入	(ユーザーの選択をもとにテーブルの GIVEN または EXPECT セクションで) 新たに左端に列を挿入します。
右端に列を挿入	(ユーザーの選択をもとにテーブルの GIVEN または EXPECT セクションで) 新たに右端に列を挿入します。
左に列を挿入	選択した列の左に新しい列を挿入します。新しい列は、(ユーザーの選択をもとにテーブルの GIVEN または EXPECT セクションに) 選択した列と同じタイプを挿入します。
右に行を挿入	選択した列の右に新しい列を挿入します。新しい列は、(ユーザーの選択をもとにテーブルの GIVEN または EXPECT セクションに) 選択した列と同じタイプを挿入します。
列を削除	選択した列を削除します。
上に行を挿入	選択した行の上に新しい行を挿入します。
下に行を挿入	選択した行の下に新しい行を挿入します。ヘッダーセルからの呼び出しの場合は、インデックス 1 の行を新たに挿入します。

テーブルの対話	説明
行を複製	選択した行を複製します。
インスタンスを複製	選択したインスタンスを複製します。
行を削除	選択した行を削除します。
シナリオを実行	単一のテストシナリオを実行します。

Insert column right または **Insert column left** コンテキストメニューオプションの動作はことなります。

- 選択した列にタイプが定義されていない場合には、タイプなしで新しい列が追加されます。
- 選択した列にタイプが定義されていない場合には、親のインスタンスタイプを引き継いだ列または新しい空の列が作成されます。
  - アクションがインスタンスヘッダーから実行された場合には、タイプなしの列が新たに作成されます。
  - アクションがプロパティーのヘッダーから実行される場合には、親のインスタンスタイプを引き継いだ列が新たに作成されます。

3.9. テストシナリオのグローバル設定

テストシナリオデザイナーの右側にあるグローバル **Settings** タブを使用して、アセットの追加プロパティーを設定および変更できます。

3.9.1. ルールベースのテストシナリオのグローバル設定の構成

以下の手順に従って、ルールベースのテストシナリオのグローバル設定を表示および編集します。

手順

1. テストシナリオデザイナーの右側にある **Settings** タブをクリックして、属性を表示します。
2. **Settings** パネルで次の属性を設定します。
   - **Name**: デザイナーの右上のツールバーから **Rename** オプションを使用して、既存のテストシナリオの名前を変更できます。
   - **Type**: この属性は、ルールベースのテストシナリオであり、読み取り専用であることを指定します。
   - **Stateless Session**: KieSession がステートレスかどうかを指定するには、このチェックボックスを選択するか、選択解除します。

注記

現在の KieSession がステートレスで、チェックボックスが選択されていない場合には、テストに失敗します。

- **KieSession**: (オプション) テストシナリオの KieSession を入力します。
- **RuleFlowGroup/AgendaGroup**: (オプション) テストシナリオの RuleFlowGroup または AgendaGroup を入力します。

3. オプション: テスト実行後にプロジェクトレベルからシミュレーション全体をスキップするには、チェックボックスを選択します。
4. **保存** をクリックします。

3.9.2. DMN ベースのテストシナリオのグローバル設定構成

以下の手順に従って、DMN ベースのテストシナリオのグローバル設定を表示および編集します。

手順

1. テストシナリオデザイナーの右側にある **Settings** タブをクリックして、属性を表示します。
2. **Settings** パネルで次の属性を設定します。
   - **Name**: デザイナーの右上のツールバーから **Rename** オプションを使用して、既存のテストシナリオの名前を変更できます。
   - **Type**: この属性は、DMN ベースのテストシナリオであり、読み取り専用であることを指定します。
   - **DMN model**: (オプション) テストシナリオ用の DMN モデルを入力します。
   - **DMN name**: これは DMN モデルの名前で、読み取り専用です。
   - **DMN namespace**: これは DMN モデルのデフォルトの名前空間で、読み取り専用です。
3. オプション: テスト実行後にプロジェクトレベルからシミュレーション全体をスキップするには、チェックボックスを選択します。
4. **保存** をクリックします。

第4章 テストシナリオテンプレート

テストシナリオの定義を指定する前に、テストシナリオテンプレートを作成する必要があります。テストシナリオテーブルのヘッダーにより、各シナリオのテンプレートが定義されます。GIVEN と EXPECT の両セクションに、インスタンスタイプとプロパティーヘッダーを設定する必要があります。インスタンスヘッダーは、特定のデータオブジェクト (ファクト) にマッピングし、プロパティーヘッダーは対応するデータオブジェクトの特定のフィールドにマッピングします。

テストシナリオデザイナーを使用すると、ルールベースと DMN ベースの両方のテストシナリオのテストシナリオテンプレートを作成できます。

4.1. ルールベースシナリオのテストシナリオテンプレートの作成

以下の手順に従い、ルールベースのテストシナリオのテストシナリオテンプレートを作成してルールとデータを検証します。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、テストシナリオを作成するプロジェクトをクリックします。
2. **Add Asset → Test Scenario** の順にクリックします。
3. **テストシナリオ** 名を入力し、適切な **パッケージ** を選択します。選択するパッケージは、必要なデータオブジェクトとルールアセットが割り当てられている、またはこれから割り当てるパッケージにする必要があります。
4. **Source type** として **RULE** を選択します。
5. **Ok** をクリックして、テストシナリオデザイナーでテストシナリオを作成して開きます。
6. **GIVEN** 列ヘッダーをデータオブジェクトにマッピングするには、以下を実行します。
    a. **GIVEN** セクションのインスタンスヘッダーセルを選択します。
    b. **Test Tools** タブからデータオブジェクトを選択します。
    c. **Insert Data Object** をクリックします。
7. **EXPECT** 列ヘッダーをデータオブジェクトにマッピングするには、以下を実行します。
    a. **EXPECT** セクションのインスタンスヘッダーセルをクリックします。
    b. **Test Tools** タブからデータオブジェクトを選択します。
    c. **Insert Data Object** をクリックします。
8. データオブジェクトをプロパティーセルにマッピングするには、以下を実行します。
    a. インスタンスヘッダーセルまたはプロパティーヘッダーセルを選択します。
    b. **Test Tools** タブからデータオブジェクトフィールドを選択します。
    c. **Insert Data Object** をクリックします。
9. データオブジェクトのプロパティーをさらに挿入するには、プロパティーヘッダーを右クリックして、必要に応じて、**Insert column right** または **Insert column left** を選択します。

10. テストシナリオの実行中に java メソッドをプロパティーセルに定義するには、以下を実行します。
    a. インスタンスヘッダーセルまたはプロパティーヘッダーセルを選択します。
    b. **Test Tools** タブからデータオブジェクトフィールドを選択します。
    c. **Insert Data Object** をクリックします。
    d. プリフィックスに **#** を指定した MVEL 式使用して、テストシナリオの実行に java メソッドを定義します。
    e. データオブジェクトのプロパティーをさらに挿入するには、プロパティーヘッダーセルを右クリックして、必要に応じて、**Insert column right** または **Insert column left** を選択します。
11. 必要に応じて、コンテキストメニューを使用して行と列を追加または削除します。

ルールベースのシナリオの式の構文に関する詳細は、「ルールベースのテストシナリオの式構文」を参照してください。

4.2. ルールベースのテストシナリオでのエイリアスの使用

テストシナリオデザイナーで、ヘッダーセルをデータオブジェクトにマッピングすると、データオブジェクトは **Test Tools** タブから削除されます。エイリアスを使用して、データオブジェクトを別のヘッダーセルに再マッピングできます。また、エイリアスを使用すると、テストシナリオで同じデータオブジェクトのインスタンスを複数指定できます。さらに、プロパティーエイリアスを作成して、使用するプロパティーの名前をテーブルで直接変更することもできます。

手順

Business Central のテストシナリオデザイナーで、ヘッダーセルをダブルクリックし、名前を手動で変更します。エイリアスの名前が一意であることを確認してください。

インスタンスが **Test Tools** タブのデータオブジェクトのリストに表示されます。

第5章 DMN ベーステストシナリオのテストテンプレート

Business Central は、DMN ベースのテストシナリオアセットすべてに対してテンプレートを生成し、この中に、関連の DMN モデルの指定のインプットやデシジョンがすべて含まれます。DMN モデルの入力ノードごとに、**GIVEN** 列が追加され、各デシジョンノードは **EXPECT** 列で表現されます。必要に応じて、デフォルトのテンプレートを変更できます。また、DMN モデルの一部のみをテストするには、生成したコラムを削除することも、EXPECT から GIVEN セクションのデシジョンノードを移動することが可能です。

5.1. DMN ベースのテストシナリオのテストシナリオテンプレート作成

以下の手順に従い、DMN ベースのシナリオのテストシナリオテンプレートを作成し、DMN モデルを検証します。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、テストシナリオを作成するプロジェクトをクリックします。
2. **Add Asset → Test Scenario** の順にクリックします。
3. **テストシナリオ** の名前を入力して、適切な **Package** を選択します。
4. **Source type** として **DMN** を選択します。
5. **Choose DMN asset**オプションを使用して、既存の DMN アセットを選択します。
6. **Ok** をクリックして、テストシナリオデザイナーでテストシナリオを作成して開きます。テンプレートは自動的に生成され、ニーズに合わせてこのテンプレートを変更できます。
7. テストシナリオの実行中に java メソッドをプロパティーセルに定義するには、以下を実行します。
    a. インスタンスヘッダーセルまたはプロパティーヘッダーセルをクリックします。
    b. **Test Tools** タブからデータオブジェクトフィールドを選択します。
    c. **Insert Data Object** をクリックします。
    d. 式を使用して、テストシナリオの実行に java メソッドを定義します。
    e. データオブジェクトのプロパティーをさらに追加するには、プロパティーヘッダーを右クリックして、必要に応じて、**Insert column right** または **Insert column left** を選択します。
8. 必要に応じて、コンテキストメニューを使用して行と列を追加または削除します。

DMN ベースのシナリオの式の構文に関する詳細は、「DMN ベースシナリオの式構文」を参照してください。

第6章 テストシナリオの定義

テストシナリオテンプレートの作成後、テストシナリオを定義する必要があります。テストシナリオテーブルの行では、個別のテストシナリオを定義します。テストシナリオには、一意のインデックス番号、説明、入力値セット (**Given** の値)、出力値セット (**Expect** の値) を定義します。

前提条件

- 選択したテストシナリオに、テストシナリオテンプレートが作成されている。

手順

1. テストシナリオデザイナーでテストシナリオを開きます。
2. テストシナリオの説明を入力して、行のセルに必要な値を入力します。
3. コンテキストメニューを使用して、必要に応じて行を追加または削除します。
   セルをダブルクリックしてインラインの編集を開始します。テスト評価で特定のセルを省略するには、そのセルを空白のままにします。

テストシナリオの定義後に、テストを実行できます。

第7章 テストシナリオでのバックグラウンドインスタンス

テストシナリオデザイナーでは、**Background** タブを使用して、ルールベースのテストシナリオと DMN ベースのテストシナリオのバックグラウンドデータを追加して設定できます。利用可能なデータオブジェクトに基づいて、テストシナリオ全体で共通となる GIVEN データを追加して定義できます。**Background** タブを使用して追加したデータは、**Model** タブデータでオーバーライドできません。

注記

Background タブで定義した GIVEN データは、同じ ***.scesim** ファイルのテストシナリオ間しか共有できず、別のテストシナリオには共有されません。

7.1. ルールベースのテストシナリオでバックグラウンドデータの追加

以下の手順に従って、ルールベースのテストシナリオでバックグラウンドデータを追加および設定します。

前提条件

- 選択したテストシナリオ用にルールベースのテストシナリオテンプレートが作成されている。ルールベースのテストシナリオの作成に関する詳細については、「ルールベースシナリオのテストシナリオテンプレートの作成」 を参照してください。
- 個々のテストシナリオが定義されている。テストシナリオの定義に関する詳細については、6章 *テストシナリオの定義* を参照してください。

手順

1. テストシナリオデザイナーでルールベースのテストシナリオを開きます。
2. テストシナリオデザイナーの **Background** タブをクリックします。
3. **GIVEN** セクションでインスタンスヘッダーセルを選択し、バックグラウンドのデータオブジェクトフィールドを追加します。
4. **Test Tools** パネルからデータオブジェクトを選択します。
5. **Insert Data Object** をクリックします。
6. バックグラウンドデータオブジェクトフィールドを追加するには、プロパティーヘッダーセルを選択します。
7. **Test Tools** パネルからデータオブジェクトを選択します。
8. **Insert Data Object** をクリックします。
9. データオブジェクトのプロパティーをさらに追加するには、プロパティーヘッダーを右クリックして、必要に応じて、**Insert column right** または **Insert column left** を選択します。
10. 必要に応じて、コンテキストメニューを使用して行と列を追加または削除します。
11. 定義済みのテストシナリオを実行します。

7.2. DMN ベースのテストシナリオでのバックグラウンドデータ追加

以下の手順に従って、DMN ベースのテストシナリオでバックグラウンドデータを追加して設定します。

前提条件

- 選択したテストシナリオ用に DMN ベースのテストシナリオテンプレートが作成されている。DMN ベースのテストシナリオの作成に関する詳細については、「DMN ベースのテストシナリオのテストシナリオテンプレート作成」 を参照してください。
- 個々のテストシナリオが定義されている。テストシナリオの定義に関する詳細については、6 章 *テストシナリオの定義* を参照してください。

手順

1. テストシナリオデザイナーでルールベースのテストシナリオを開きます。
2. テストシナリオデザイナーの **Background** タブをクリックします。
3. **GIVEN** セクションでインスタンスヘッダーセルを選択し、バックグラウンドのデータオブジェクトフィールドを追加します。
4. **Test Tools** パネルからデータオブジェクトを選択します。
5. **Insert Data Object** をクリックします。
6. バックグラウンドデータオブジェクトフィールドを追加するには、プロパティーヘッダーセルを選択します。
7. **Test Tools** パネルからデータオブジェクトを選択します。
8. **Insert Data Object** をクリックします。
9. データオブジェクトのプロパティーをさらに追加するには、プロパティーヘッダーを右クリックして、必要に応じて、**Insert column right** または **Insert column left** を選択します。
10. 必要に応じて、コンテキストメニューを使用して行と列を追加または削除します。
11. 定義済みのテストシナリオを実行します。

第8章 テストシナリオでのリストおよびマッピングコレクションの使用

テストシナリオデザイナーは、DMN ベースとルールベース両方のテストシナリオのリストおよびマッピングコレクションをサポートします。リストやマッピングのようなコレクションは、**GIVEN** および **EXPECT** の両コラムに、特定のセルの値として作成して定義できます。

注記

マップエントリーでは、エントリーキーのデータタイプは **String** である必要があります。

ルールベースのコレクションエディターの **EXPECT** コラムにパラメーターを指定するには、**actualValue** のキーワードを使用し、DMN ベースのテストシナリオでは **?** のキーワードを使用します。

手順

1. コラムタイプを先に設定します (タイプがリストまたはマッピングのフィールドを使用します)。
2. コラム内のセルをダブルクリックして、値を入力します。
3. コレクションエディターのポップアップでデータオブジェクトのリスト値を作成するには、以下を実行します。
   a. **Creae List** を選択します。
   b. **Add New Item** をクリックします。
   c. 必要な値を入力して、チェックアイコン  をクリックし、追加した各コレクションアイテムを保存します。
   d. **保存** をクリックします。
   e. コレクションからアイテムを編集するには、コレクションのポップアップエディターで鉛筆のアイコンをクリックします。
   f. **Save Changes** をクリックします。
   g. コレクションからアイテムを削除するには、コレクションのポップアップエディターでゴミ箱のアイコンをクリックします。
4. コレクションエディターのポップアップでデータオブジェクトのリスト値を定義するには、以下を実行します。
   a. **Define List** を選択します。
   b. MVEL または FEEL 式を使用して、テキストフィールドでリストの値を定義します。ルールベースのテストシナリオは MVEL 式言語を、DMN ベースのテストシナリオは FEEL 式言語を使用します。
   c. **保存** をクリックします。

5. コレクションエディターのポップアップでデータオブジェクトのマップの値を作成するには、以下を実行します。

   a. **Create Map** を選択します。

   b. **Add New Item** をクリックします。

   c. 必要な値を入力して、チェックアイコン  をクリックし、追加した各コレクションアイテムを保存します。

   d. **保存** をクリックします。

   e. コレクションからアイテムを編集するには、コレクションのポップアップエディターで鉛筆のアイコンをクリックします。

   f. **Save Changes** をクリックします。

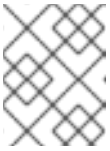
   g. コレクションからアイテムを削除するには、コレクションのポップアップエディターでゴミ箱のアイコンをクリックします。

6. コレクションエディターのポップアップでデータオブジェクトのマップの値を定義するには、以下を実行します。

   a. **Define Map** を選択します。

   b. MVEL または FEEL 式を使用して、テキストフィールドでマップの値を定義します。ルールベースのテストシナリオは MVEL 式言語を、DMN ベースのテストシナリオは FEEL 式言語を使用します。

   c. **保存** をクリックします。



注記

DMN ベースのテストシナリオ向けにマップ値を定義するには、コレクションエディターを使用せずにファクトを追加して FEEL 式を使用します。

7. **Remove** をクリックして全コレクションを削除します。

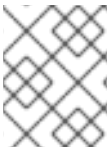
第9章 テストシナリオの式構文

テストシナリオデザイナーでは、ルールベースおよび DMN ベースのテストシナリオ両方似向け、異なる式言語をサポートします。ルールベースのテストシナリオは MVFLEX 式言語 (MVEL) を、DMN ベースのテストシナリオは FEEL (Friendly Enough Expression Language) 式言語をサポートします。

9.1. ルールベースのテストシナリオの式構文

ルールベースのテストシナリオは、以下の組み込みデータ型をサポートします。

- 文字列
- Boolean 型
- 整数
- Long
- Double
- 浮動
- 文字
- バイト
- Short
- LocalDate



注記

その他のデータ型については、プリフィックスとして **#** を指定した MVEL 式を使用してください。

テストシナリオデザイナーの BigDecimal の例に従って、プリフィックスに **#** を使用して java 式を設定します。

- **GIVEN** 列の値に、**# java.math.BigDecimal.valueOf(10)** と入力します。
- **EXPECT** 列の値に **# actualValue.intValue() == 10** と入力します。

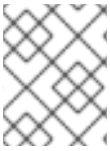
java 式の **EXPECT** 列の実際の値を参照して、条件を実行できます。

テストシナリオデザイナーでは、以下に示すルールベースのテストシナリオの定義式がサポートされます。

表9.1 式構文の説明

演算子	説明
=	値と同等であることを指定します。これは、全コラムでデフォルトとなっており、GIVEN コラムでサポートされる唯一の演算子です。

演算子	説明
=, =!, <>	値が同等でないことを指定します。この演算子は、他の演算子と組み合わることができます。
<, >, <=, >=	比較を指定します。未満、値よりも大きい、以下、以上
#	この演算子を使用して、プロパティーヘッダーセルに java 式の値を設定することで、java メソッドとして実行できます。
[value1, value2, value3]	値の一覧を指定します。1 つまたは複数の値が有効な場合には、シナリオの定義は True と評価されます。
expression1; expression2; expression3	式のリストを指定します。すべての式が有効な場合には、シナリオ定義は True と評価されます。



注記

セルがからの場合は、評価が省略されます。空の文字列を定義するには、**=**、**[]** または **;** を使用します。null 値を定義するには、**null** を使用します。

表9.2 式の例

式	説明
-1	実際の値は、-1 と同等です。
< 0	実際の値は、0 未満です。
! > 0	実際の値は、0 以下です。
[-1, 0, 1]	実際の値は、-1、0 または 1 です。
<> [1, -1]	実際の値は、1 でも -1 でもありません。
! 100; 0	実際の値は、100 ではなく、0 です。
!= < 0; <> > 1	実際の値は、0 未満でも、1 以上でもありません。
<> <= 0; >= 1	実際の値は、0 以下でなく、1 以上です。

注記

ルールベースのテストシナリオデザイナーの右側にある **Scenario Cheatsheet** タブで、サポートされているコマンドと構文を参照できます。

9.2. DMN ベースシナリオの式構文

以下のデータタイプは、テストシナリオデザイナーの DMN ベーステストシナリオでサポートされます。

表9.3 DMN ベースのシナリオがサポートするデータタイプ

サポートされるデータタイプ	説明
番号および文字列	文字列は、**"John Doe"**、**"Brno"** または **""** のように、引用符で囲む必要があります。
ブール値	**true**、**false** および **null**.
日時	例: **date("2019-05-13")** または **time("14:10:00+02:00")**
関数	**avg**、**max** などの組み込み計算機能をサポートします。
コンテキスト	例: **{x : 5, y : 3}**
範囲とリスト	例: **[1 .. 10]** または **[2, 3, 4, 5]**

注記

DMN ベースのテストシナリオデザイナーの右側にある Scenario Cheatsheet タブで、サポートされているコマンドと構文を参照できます。

第10章 テストシナリオの実行

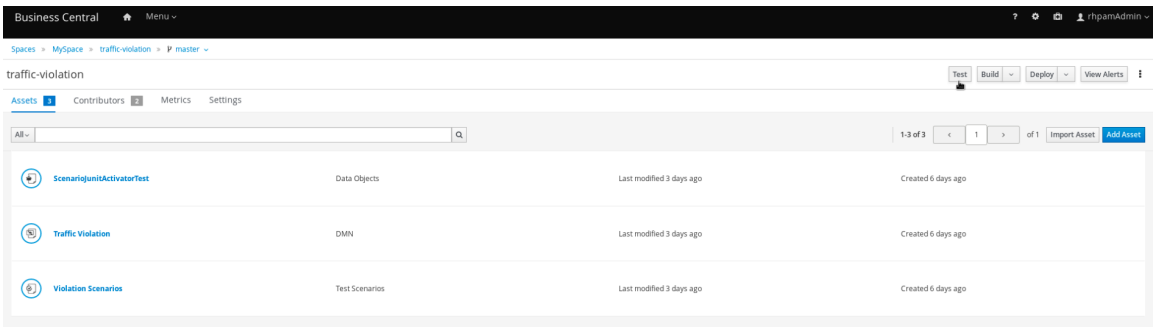
テストシナリオテンプレートを作成してテストシナリオを定義した後に、テストを実行してビジネスルールとデータを検証できます。

手順

1. 定義済みのテストシナリオを実行するには、次のタスクのいずれかを実行します。

   - プロジェクトで利用可能なすべてのテストシナリオを複数のアセット内で実行するには、プロジェクトページの右上隅で、**Test** をクリックします。

     プロジェクトビューからの全テストシナリオの実行

   - 単一の **.scesim** ファイルで定義された利用可能なすべてのテストシナリオを実行するには、テストシナリオデザイナーの上部の **Run Test**  アイコンをクリックします。

   - 単一の **.scesim** ファイルで定義されている単一のテストシナリオを実行するには、実行するテストシナリオの行を右クリックし、**Run scenario** を選択します。

2. **Test Report** パネルには、テストの概要とシナリオステータスが表示されます。
   テストの実行後に、テストシナリオテーブルに入力された値が期待値と一致しない場合には、対応するセルがハイライト表示されます。

3. テストが失敗した場合には、次のタスクを実行して、失敗内容に対してトラブルシューティングすることができます。

   - ポップアップウィンドウでエラーメッセージを確認するには、ハイライトされたセルにマウスのカーソルを合わせます。

   - デザイナーの下部にある **Alerts** パネルまたはプロジェクトビューを開いてエラーメッセージを確認するには、**View Alerts** をクリックします。

   - 必要な変更を加えて、シナリオが成功するまで再度テストを実行します。

第11章 ローカルでのテストシナリオの実行

Red Hat Process Automation Manager では、テストシナリオを Business Central で直接実行することも、コマンドラインを使用してローカルで実行することもできます。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. プロジェクトのホームページで **Settings** タブを選択します。
3. **git URL** を選択して、**Clipboard**  をクリックし、git url をコピーします。
4. コマンドターミナルを開いて、git プロジェクトのクローンを作成するディレクトリーに移動します。
5. 以下のコマンドを実行します。

```
git clone your_git_project_url
```

your_git_project_url は、**git://localhost:9418/MySpace/ProjectTestScenarios** などの適切なデータに置き換えます。

6. プロジェクトのクローンを正しく作成したら、git プロジェクトディレクトリーに移動して、以下のコマンドを実行します。

```
mvn clean test
```

プロジェクトのビルド情報およびテスト結果 (テスト実行回数およびテスト実行が成功したかどうかなど) は、コマンドターミナルに表示されます。失敗がある場合には、Business Central で必要な変更を行い、変更をプルし、もう一度コマンドを実行します。

第12章 テストシナリオスプレッドシートのエクスポートとインポート

これらのセクションでは、テストシナリオデザイナーでテストシナリオスプレッドシートをエクスポートおよびインポートする方法を説明します。Microsoft Excel や LibreOffice Calc などのソフトウェアを使用して、テストシナリオスプレッドシートを分析および管理できます。テストシナリオデザイナーは、**.CSV** ファイル形式をサポートします。コンマ区切り値 (CSV) 形式の RFC 仕様の詳細については、Common Format and MIME Type for Comma-Separated Values (CSV) Files を参照してください。

12.1. テストシナリオスプレッドシートのエクスポート

以下の手順に従って、テストシナリオデザイナーを使用してテストシナリオスプレッドシートをエクスポートします。

手順

1. 右上のテストシナリオデザイナーのツールバーで、**Export** 

ボタンをクリックします。
2. ローカルファイルディレクトリー内の宛先を選択し、**.CSV** ファイルの保存を確定します。

.CSV ファイルがローカルマシンにエクスポートされます。

12.2. テストシナリオスプレッドシートのインポート

以下の手順に従って、テストシナリオデザイナーを使用してテストシナリオスプレッドシートをインポートします。

手順

1. 右上のテストシナリオデザイナーのツールバーで、**Import** 

ボタンをクリックします。
2. **Select file to Import** プロンプトで、**Choose File...** をクリックし、ローカルファイルディレクトリーからインポートする **.CSV** ファイルを選択します。
3. **Import** をクリックします。

.CSV ファイルがテストシナリオデザイナーにインポートされます。

警告

選択した **.CSV** ファイルのヘッダーを変更しないでください。変更した場合には、スプレッドシートが正常にインポートされないことがあります。

第13章 テストシナリオのカバレッジレポート

テストシナリオデザイナーは、右側の **Coverage Report** タブを使用してテストカバレッジ統計を明確かつ一貫したかたちで表示します。また、カバレッジレポートをづアンロードして、テストカバレッジ統計の表示や分析も可能です。ダウンロードしたテストシナリオのカバレッジレポートは、**.CSV** ファイル形式をサポートします。CSV (Comma-Separated Values) 形式の RFC 仕様に関する詳細は、Common Format and MIME Type for Comma-Separated Values (CSV) Files を参照してください。

ルールベースおよび DMN ベースのテストシナリオのカバレッジレポートを表示できます。

13.1. ルールベースのテストシナリオのカバレッジレポート生成

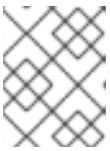
ルールベースのテストシナリオでは、**Coverage Report** タブに以下の詳細情報が含まれています。

- 利用可能なルールの数
- 実行するルールの数
- 実行するルールの割合
- 実行するルールの割合 (円グラフで表示)
- 各ルールを実行する回数
- 定義済みのテストシナリオごとに実行するルール

以下の手順に従って、ルールベースのテストシナリオのカバレッジレポートを生成します。

前提条件

- 選択したテストシナリオ用にルールベースのテストシナリオテンプレートが作成されている。ルールベースのテストシナリオの作成に関する詳細については、「ルールベースシナリオのテストシナリオテンプレートの作成」 を参照してください。
- 個々のテストシナリオが定義されている。テストシナリオの定義に関する詳細については、*6章 テストシナリオの定義* を参照してください。



注記

ルールベースのテストシナリオのカバレッジレポートを生成するには、ルールを最低でも 1 つ作成する必要があります。

手順

1. テストシナリオデザイナーでルールベースのテストシナリオを開きます。
2. 定義済みのテストシナリオを実行します。
3. テストシナリオデザイナーの右側にある **Coverage Report** をクリックして、テストカバレッジ統計を表示します。
4. オプション: テストシナリオのレポートをダウンロードするには、**Download report** をクリックします。

13.2. DMN ベースのテストシナリオのカバレッジレポート生成

DMN ベースのテストシナリオでは、**Coverage Report** タブに以下の詳細情報が含まれています。

- 利用可能なデシジョン数
- 実行するデシジョン数
- 実行するデシジョンの割合
- 実行するデシジョンの割合 (円グラフで表示)
- 各デシジョンを実行する回数
- 定義済みのテストシナリオごとに実行するデシジョン

以下の手順に従って、DMN ベースのテストシナリオのカバレッジレポートを生成します。

前提条件

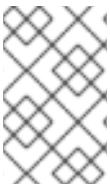
- 選択したテストシナリオ用に DMN ベースのテストシナリオテンプレートが作成されている。DMN ベースのテストシナリオの作成に関する詳細については、「DMN ベースのテストシナリオのテストシナリオテンプレート作成」 を参照してください。
- 個々のテストシナリオが定義されている。テストシナリオの定義に関する詳細については、6 章 *テストシナリオの定義* を参照してください。

手順

1. テストシナリオデザイナーで DMN ベースのテストシナリオを開きます。
2. 定義済みのテストシナリオを実行します。
3. テストシナリオデザイナーの右側にある **Coverage Report** をクリックして、テストカバレッジ統計を表示します。
4. オプション: テストシナリオのレポートをダウンロードするには、**Download report** をクリックします。

第14章 KIE SERVER REST API を使ったテストシナリオの実行

KIE Server の REST エンドポイントと直接対話すると、呼び出しコードとデシジョンロジックの定義を最も分離できます。KIE Server REST API を使用して、外部のテストシナリオを実行し、デプロイしたプロジェクトに対してテストシナリオを実行できます。



注記

この機能はデフォルトでは無効になっているので、**org.kie.scenariosimulation.server.ext.disabled** のシステムプロパティーを使用して有効化してください。

KIE Server REST API についての詳細は、『KIE API を使用した Red Hat Process Automation Manager の操作』を参照してください。

前提条件

- KIE Server がインストールされ、設定されている (**kie-server** ロールが割り当てられているユーザーの既知のユーザー名と認証情報を含む)。インストールオプションは、『Red Hat Process Automation Manager インストールの計画』を参照してください。
- KJAR アーティファクトとしてプロジェクトをビルドし、KIE Server にデプロイしている。
- KIE コンテナーの ID がある。

手順

1. KIE Server REST API エンドポイントにアクセスするためのベース URL を決定します。これには、以下の値が必要です (例ではローカルのデプロイメント値を使用しています)。
   - ホスト (**localhost**)
   - ポート (**8080**)
   - ルートコンテキスト (**kie-server**)
   - ベース REST パス (**services/rest/**)

   交通違反プロジェクトでのローカルデプロイメントにおけるベース URL の例:

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

2. ユーザー認証要件を決定します。
   ユーザーを KIE Server 設定に直接定義すると、ユーザー名およびパスワードを要求する HTTP Basic 認証が使用されます。要求を成功させるには、ユーザーに **kie-server** ルールが必要です。

   以下の例は、curl 要求に認証情報を追加する方法を示します。

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

   Red Hat シングルサインオンを使用して KIE Server を設定している場合は、要求にベアラートークンが必要です。

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

3. 要求と応答の形式を指定します。REST API エンドポイントには XML 書式が利用でき、このエンドポイントは要求ヘッダーを使用して設定されます。

   XML

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

4. テストシナリオを実行します。
   [POST] server/containers/{containerId}/scesim

   curl 要求例:

   ```
   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
   ```

   XML 要求例:

```
<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>
```

XML 応答例:

```
<?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>
```

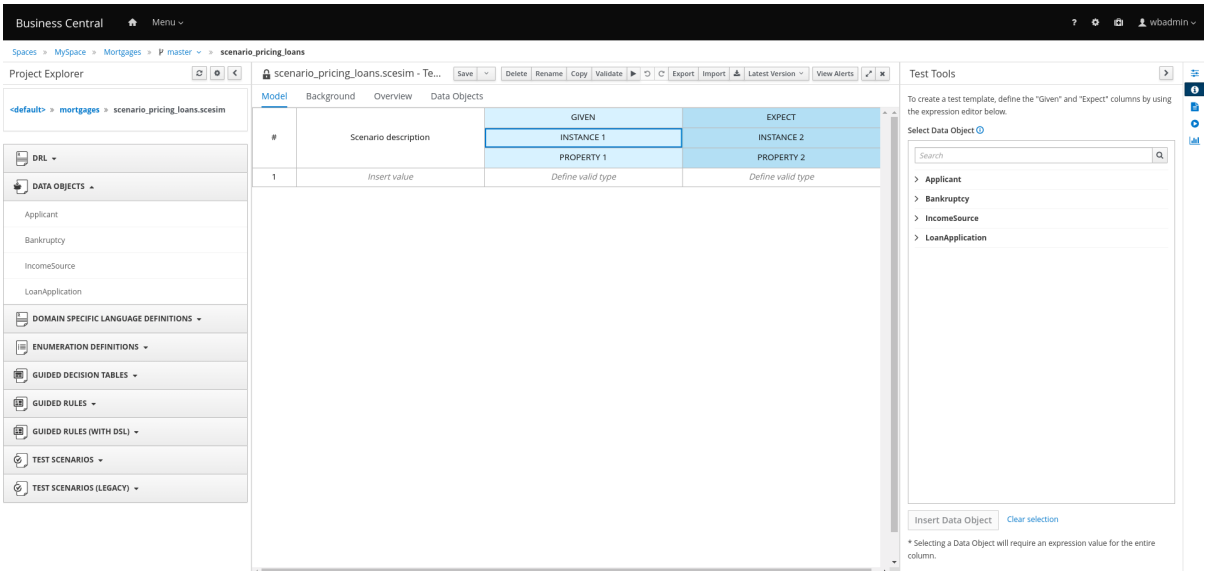
第15章 MORTGAGES サンプルプロジェクトを使用したテストシナリオの作成

本章では、テストシナリオデザイナーを使用して、Business Central に同梱されているサンプルの **Mortgages** プロジェクトからテストシナリオを作成して実行する方法を説明します。本章のテストシナリオの例は、**Mortgages** プロジェクトからの **Pricing loans** のガイド付きデシジョンテーブルに基づいています。

手順

1. Business Central にログインし、**Menu → Design → Projects** の順にクリックし、**Mortgages** をクリックします。
2. プロジェクトが **Projects** に表示されていない場合には、**MySpace** から **Try Samples → Mortgages → OK** の順にクリックします。
   アセットのウィンドウが表示されます。
3. **Add Asset → Test Scenario** の順にクリックします。
4. **Test Scenario** の名前として、**scenario_pricing_loans** を入力し、**Package** ドロップダウンリストから、デフォルトの **mortgages.mortgages** パッケージを選択します。
   選択するパッケージには、必要なルールアセットすべてが含まれている必要があります。
5. **Source type** として **RULE** を選択します。
6. **Ok** をクリックして、テストシナリオデザイナーでテストシナリオを作成して開きます。
7. **Project Explorer** を展開して以下を確認します。
   - **Applicant**、**Bankruptcy**、**IncomeSource** および **LoanApplication** データオブジェクトが存在する。
   - **Pricing loans** ガイド付きのデシジョンテーブルが存在する。
   - 新しいテストシナリオが **Test Scenario** に表示されていることを確認する。
8. データオブジェクトがすべて配置されていることを確認してから、テストシナリオデザイナーの **Model** タブに戻り、利用可能なデータオブジェクトのシナリオに、 **GIVEN** データと **EXPECT** データを定義します。

   空のテストシナリオデザイナー



9. **GIVEN** 列の詳細を定義します。

    a. **GIVEN** 列ヘッダーにある **INSTANCE 1** という名前のセルをクリックします。

    b. **Test Tools** パネルから、**LoanApplication** データオブジェクトを選択します。

    c. **Insert Data Object** をクリックします。

10. データオブジェクトのプロパティーを作成するには、必要に応じてプロパティーヘッダーセルを右クリックして、**Insert column right** または **Insert column left** を選択します。この例では、**GIVEN** 列にあと 2 つプロパティーセルを作成する必要があります。

11. 最初のプロパティーヘッダーセルを選択します。

    a. **Test Tools** パネルから、**LoanApplication** データオブジェクトを選択して展開します。

    b. Click **amount**.

    c. **Insert Data Object** をクリックして、データオブジェクトフィールドをプロパティーヘッダーセルにマッピングします。

12. 2 番目のプロパティーヘッダーセルを選択します。

    a. **Test Tools** パネルから、**LoanApplication** データオブジェクトを選択して展開します。

    b. **deposit** をクリックします。

    c. **Insert Data Object** をクリックします。

13. 3 番目のパーティーのプロパティーヘッダーセルを選択します。

    a. **Test Tools** パネルから、**LoanApplication** データオブジェクトを選択して展開します。

    b. **lengthYears** をクリックします。

    c. **Insert Data Object** をクリックします。

14. **LoanApplication** ヘッダーセルを右クリックして、**Insert column right** を選択すると、右側に、新しい **GIVEN** 列が作成されます。

15. 新しいヘッダーセルを選択します。

a. **Test Tools** パネルから、**IncomeSource** データオブジェクトを選択します。

b. **Insert Data Object** をクリックして、データオブジェクトフィールドをヘッダーセルにマッピングします。

16. **IncomeSource** の下のプロパティーヘッダーセルをクリックします。

a. **Test Tools** パネルから、**IncomeSource** データオブジェクトを選択して展開します。

b. **type** をクリックします。

c. **Insert Data Object** をクリックして、データオブジェクトフィールドをプロパティーヘッダーセルにマッピングします。
GIVEN 列セルがすべて定義されました。

17. 次に **EXPECT** 列の詳細を定義します。

a. **EXPECT** コラムヘッダーにある **INSTANCE 2** という名前のセルをクリックします。

b. **Test Tools** パネルから、**LoanApplication** データオブジェクトを選択します。

c. **Insert Data Object** をクリックします。

18. データオブジェクトのプロパティーを作成するには、必要に応じて **Insert column right** または **Insert column left** を選択します。**EXPECT** コラムの下に、さらに 2 つプロパティーヘッダーセルを作成します。

19. 最初のプロパティーヘッダーセルを選択します。

a. **Test Tools** パネルから、**LoanApplication** データオブジェクトを選択して展開します。

b. **Approved** をクリックします。

c. **Insert Data Object** をクリックして、データオブジェクトフィールドをプロパティーヘッダーセルにマッピングします。

20. 2 番目のプロパティーヘッダーセルを選択します。

a. **Test Tools** パネルから、**LoanApplication** データオブジェクトを選択して展開します。

b. **insuranceCost** をクリックします。

c. **Insert Data Object** をクリックして、データオブジェクトフィールドをプロパティーヘッダーセルにマッピングします。

21. 3 番目のパーティーのプロパティーヘッダーセルを選択します。

a. **Test Tools** パネルから、**LoanApplication** データオブジェクトを選択して展開します。

b. **approvedRate** をクリックします。

c. **Insert Data Object** をクリックして、データオブジェクトフィールドをプロパティーヘッダーセルにマッピングします。

22. テストシナリオを定義するには、1 行目に以下のデータを入力します。

- **GIVEN** 列の値として、Scenario Description には **Row 1 test scenario**、amount には **150000**、deposit には **19000**、lengthYears には **30**、type には **Asset** を入力します。

- EXPECT 列の値として、approved には **true**、insuranceCost には **0**、approvedRate には **2** を入力します。

23. 次に 2 番目の行に、以下のデータを入力します。

    - GIVEN 列の値として、Scenario Description には **Row 2 test scenario**、amount には **100002**、deposit には **2999**、lengthYears には **20**、type には **Job** を入力します。

    - EXPECT 列の値として、approved には **true**、insuranceCost には **10**、approvedRate には **6** を入力します。

24. シナリオに対して、GIVEN、EXPECT、その他のデータをすべて定義したら、テストシナリオデザイナーで **Save** をクリックして、設定した内容を保存します。

25. 右上隅の **Run Test** をクリックして **.scesim** ファイルを実行します。
テスト結果は、**Test Report** パネルに表示されます。**View Alerts** をクリックして、**Alerts** セクションからメッセージを表示します。テストに失敗した場合は、ウィンドウの下部にある **Alerts** セクションのメッセージを参照し、シナリオの全コンポーネントをレビューして修正してから、シナリオが合格するまでシナリオの検証を再度行います。

26. テストシナリオデザイナーで **Save** をクリックして、必要な変更をすべて加えてから、作業を保存します。

第16章 BUSINESS CENTRAL でのテストシナリオ (レガシー) デザイナー

Red Hat Process Automation Manager は現在、新規の **テストシナリオ** デザイナーと以前の **テストシナリオ (レガシー)** デザイナーの両方をサポートします。デフォルトのデザイナh、新規のテストシナリオデザイナーで、ルールと DMN モデルのテストをサポートし、テストシナリオの全体的な使用感が改善されています。必要に応じて、レガシーのテストシナリオをそのまま使用することができますが、ルールベースのテストシナリオしかサポートされません。

16.1. テストシナリオ (レガシー) の作成および実行

ビジネスルールデータをデプロイする前に、Business Central にテストシナリオを作成して、その機能をテストできます。基本的なテストシナリオには、少なくても以下のデータが必要です。

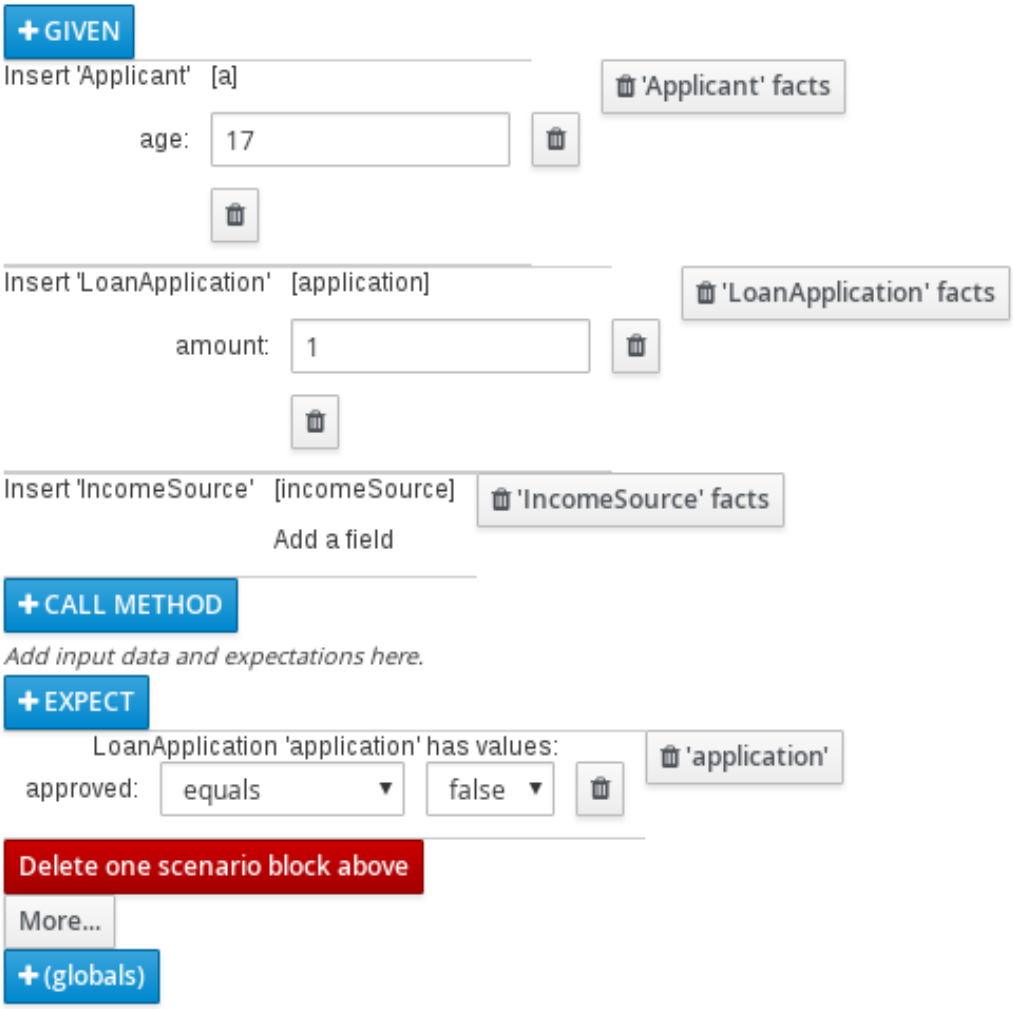
- 関連するデータオブジェクト
- **GIVEN (指定した)** ファクト
- **EXPECT (想定される)** 結果

テストシナリオでは、このデータを使用し、定義したファクトに基づいて、そのルールインスタンスに対して想定した結果と実際の結果の妥当性を検証できます。**CALL METHOD** と利用可能な **globals** をテストシナリオに追加することもできますが、これは必須ではありません。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. **Add Asset → テストシナリオ (レガシー)** の順にクリックします。
3. 分かりやすい **Test Scenario** 名を入力し、適切な **パッケージ** を選択します。指定するパッケージは、必要なルールアセットが割り当てられている、またはこれから割り当てるパッケージと同じにする必要があります。データオブジェクトは、パッケージからアセットのデザイナーにインポートできます。
4. **OK** をクリックして、テストシナリオを作成します。
   Project Explorer の **Test Scenarios** パネルに、新しいテストシナリオが追加されました。
5. **Data Objects** タブをクリックして、テストするルールに必要なデータオブジェクトがすべてリストされていることを検証します。追加されていない場合は、**New item** をクリックして別のパッケージから必要なデータオブジェクトをインポートするか、パッケージに データオブジェクトを作成 します。
6. データオブジェクトをすべて配置したら、テストシナリオデザイナーの **Model** タブに戻り、利用可能なデータオブジェクトのシナリオに、 **GIVEN** データと **EXPECT** データを定義します。

図16.1 テストシナリオデザイナー



GIVEN セクションには、テストする入力ファクトを定義します。たとえば、プロジェクトの **Underage** ルールで、ローン申請者の年齢が 21 歳未満であれば承認しない場合は、テストシナリオの GIVEN ファクトで、**Applicant** の **age** に、21 より小さい数字に設定する必要があります。

EXPECT セクションには、GIVEN に入力したファクトに基づいて想定される結果を定義します。つまり、入力ファクトを **GIVEN (指定)** すると、その他のファクトが有効であること、またはルール全体が有効であることを **EXPECT (想定)** します。たとえば、このシナリオで、申請者が 21 歳未満の場合に **想定される** 結果は、(申請者の年齢が基準を満たさないため) **LoanApplication** の **approved** が **false** になるか、**Underage** ルール全体が有効になります。

7. CALL METHOD と globals をテストシナリオに追加することもできます。

- CALL METHOD: ルールの実行開始時に、別のファクトからメソッドを呼び出します。CALL METHOD をクリックし、ファクトを選択し、 をクリックして呼び出すメソッドを選択します。(可能な場合は) プロジェクトに対してインポートした Java ライブラリー、または JAR から (ArrayList のメソッドなどの) Javaクラスメソッドを呼び出すことができます。

- globals: テストシナリオで妥当性を確認するプロジェクトにグローバル変数を追加します。globals をクリックして、妥当性を確認する変数を選択し、テストシナリオデザイナーでグローバル名をクリックして、グローバル変数に適用するフィールド値を定義します。グローバル変数が利用できない場合は、Business Central に新しいアセットとして作成する

必要があります。グローバル変数は、デシジョンエンジンに表示されますが、ファクトに対するオブジェクトとは異なるオブジェクトの名前です。グローバルのオブジェクトを変更しても、ルールの再評価は行われません。

8. 必要に応じて、テストシナリオデザイナーの下部で **More** をクリックし、同じシナリオファイルに別のデータブロックを追加します。

9. シナリオに対して、**GIVEN**、**EXPECT**、その他のデータをすべて定義したら、テストシナリオデザイナーで **Save** をクリックして、設定した内容を保存します。

10. 右上の **Run scenario** をクリックして、この **.scenario** ファイルを実行します。プロジェクトパッケージに **.scenario** ファイルを複数保存している場合は、**Run all scenarios** をクリックして、すべてのシナリオを実行します。**Run scenario** オプションでは、対象の **.scenario** ファイルを保存する必要はありませんが、**Run all scenarios** オプションを使用する場合は、すべての **.scenario** ファイルを保存する必要があります。
テストに失敗したら、ウィンドウ下部の **Alerts** メッセージに記載されている問題に対応し、シナリオの全コンポーネントを見直し、エラーが表示されなくなるまで妥当性確認を行います。

11. 変更がすべて終了したら、テストシナリオデザイナーで **Save** をクリックして、設定した内容を保存します。

16.1.1. テストシナリオ (レガシー) への GIVEN ファクトの追加

GIVEN セクションには、テストする入力ファクトを定義します。たとえば、プロジェクトの **Underage** ルールで、ローン申請者の年齢が 21 歳未満であれば承認しない場合は、テストシナリオの GIVEN ファクトで、**Applicant** の **age** に、21 より小さい数字を設定する必要があります。

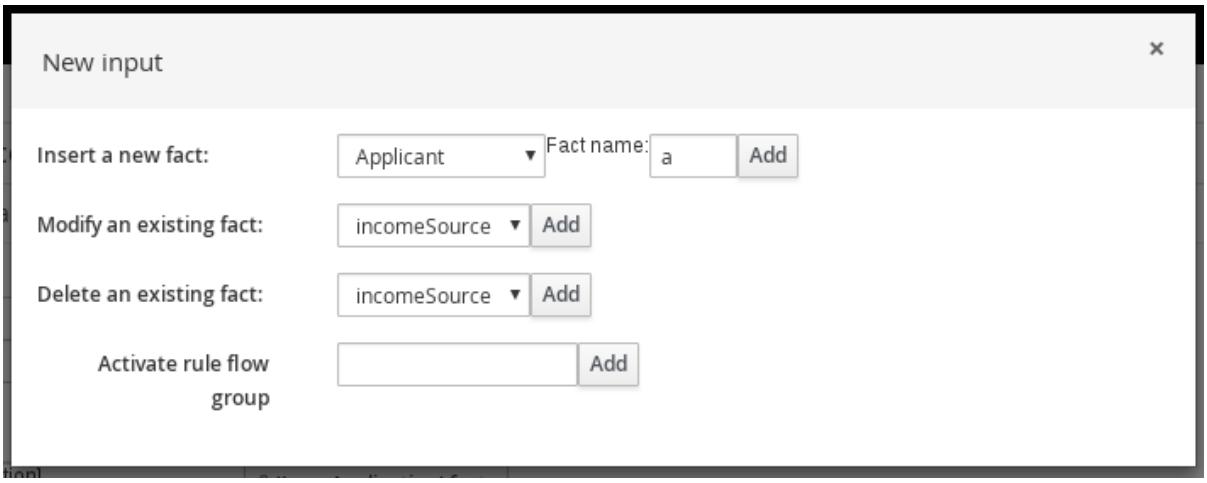
前提条件

- テストシナリオに必要なデータオブジェクトがすべて作成、またはインポートされていて、**テストシナリオ (レガシー)** デザイナーの **Data Objects** タブにリストされている。

手順

1. **テストシナリオ (レガシー)** デザイナーで、**GIVEN** をクリックして、利用可能なファクトが含まれる **New input** ウィンドウを開きます。

図16.2 テストシナリオへの GIVEN 情報の追加

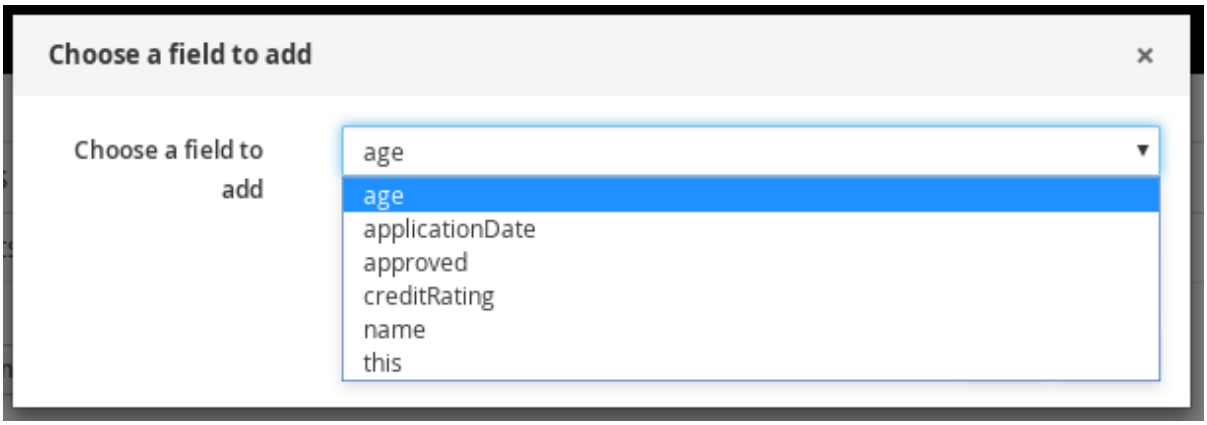


リストには以下のオプションが含まれます。表示されるオプションは、テストシナリオデザイナーの **Data Objects** タブで利用可能なデータオブジェクトによって異なります。

- **Insert a new fact:** ファクトを追加して、フィールド値を修正します。**Fact name** にファクトの変数を入力します。
- **Modify an existing fact:** (別のファクトが追加されている場合に限り表示) シナリオの実行と実行の間に、デシジョンエンジンにすでに挿入されていて、修正するファクトを指定します。
- **Delete an existing fact:** (別のファクトが追加されている場合に限り表示) シナリオの実行と実行の間に、デシジョンエンジンにすでに挿入されていて、削除するファクトを指定します。
- **Activate rule flow group:** そのグループ内にあるすべてのルールをテストできるように、有効にするルールフローグループを指定します。

2. 目的の入力オプションに対するファクトを選択し、**Add** をクリックします。たとえば、**Insert a new fact:** に **Applicant** を設定し、**Fact name** に対して **a** または **app**、もしくは別の変数を入力します。
3. テストシナリオデザイナーのファクトをクリックし、修正するフィールドを選択します。

図16.3 ファクトフィールドの修正



4. 編集アイコン ( ) をクリックし、以下のフィールド値を選択します。
   - **Literal value:** 特定のリテラル値を入力するオープンフィールドを作成します。
   - **Bound variable:** フィールドの値を、選択した変数にバインドするファクトに設定します。フィールドタイプが、バインドした変数型に一致する必要があります。
   - **Create new fact:** 新しいファクトを作成し、そのファクトを親ファクトのフィールド値として割り当てます。テストシナリオデザイナーで子ファクトをクリックし、同じようにフィールド値を割り当てるか、別のファクトをネストできます。
5. 続いて、シナリオに別の **GIVEN** 入力データを追加し、テストシナリオデザイナーで **Save** をクリックして、設定した内容を保存します。

16.1.2. テストシナリオ (レガシー) への EXPECT 結果の追加

EXPECT セクションには、**GIVEN** に入力したファクトに基づいて想定される結果を定義します。つまり、入力ファクトを **GIVEN (指定)** すると、その他のファクトが有効であること、またはルール全体が有効であることを **EXPECT (想定)** します。たとえば、シナリオで、申請者が 21 歳未満の場合に **想定される** 結果は、(申請者の年齢が基準を満たさないため) **LoanApplication** の **approved** が **false** になるか、**Underage** ルール全体が有効になります。

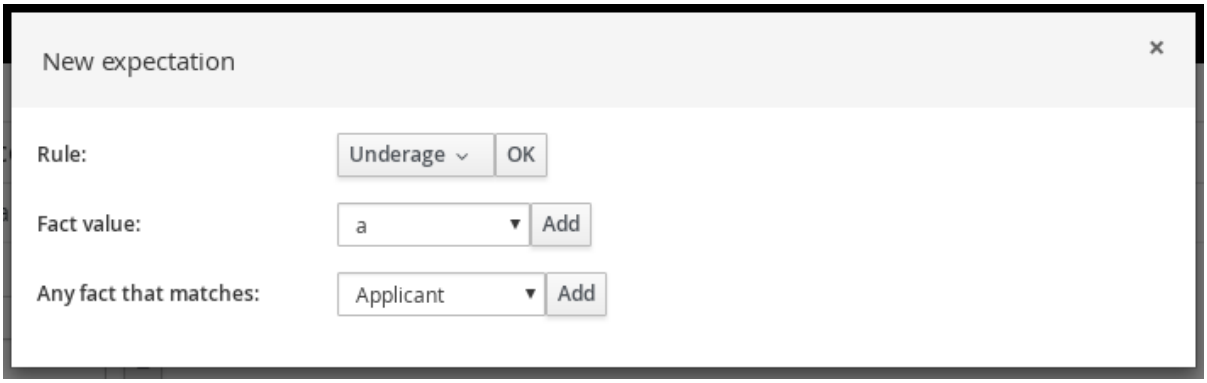
前提条件

- テストシナリオに必要なデータオブジェクトがすべて作成、またはインポートされていて、**テストシナリオ (レガシー)** デザイナーの **Data Objects** タブにリストされている。

手順

1. **テストシナリオ (レガシー)** デザイナーで、**EXPECT** をクリックして、利用可能なファクトが含まれる **New expectation** ウィンドウを開きます。

図16.4 テストシナリオへの EXPECT 結果の追加



リストには以下のオプションが含まれます。表示されるオプションは、**GIVEN** セクションのデータや、テストシナリオデザイナーの **Data Objects** タブで利用可能なデータオブジェクトによって異なります。

- **Rule:** プロジェクトに、**GIVEN** に指定した内容に対して有効になることが想定される特定のルールを指定します。ルールの名前を入力するか、ルールリストから選択します。次に、テストシナリオデザイナーで、ルールが有効になるべき回数を指定します。

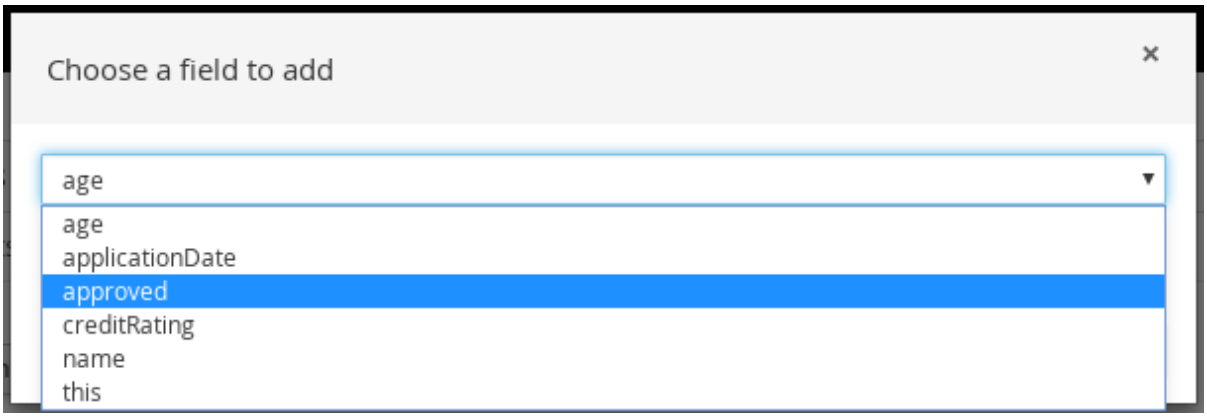
- **Fact value:** ファクトを選択し、**GIVEN** セクションに定義したファクトに対して有効になることが想定される値を定義します。ファクトは、**GIVEN** の入力に対して事前に定義した **Fact name** でリストされます。

- **Any fact that matches: GIVEN** に指定した内容に対して、指定した値を持つファクトが最低 1 つ存在するかどうかの妥当性を確認します。

2. (**Fact value: application** など) 期待される結果のファクトを選択し、**Add** または **OK** を選択します。

3. テストシナリオデザイナーでファクトをクリックし、追加または修正するフィールドを選択します。

図16.5 ファクトフィールドの修正



4. フィールド値に、GIVEN に指定した内容に対して、有効になると想定される値 (**approved** | **equals** | **false** など) を設定します。

5. 続いて、シナリオに別の EXPECT 入力データを追加し、テストシナリオデザイナーで Save をクリックして、設定内容を保存します。

6. シナリオに GIVEN、EXPECT、その他のデータを定義して保存したら、右上の Run scenario をクリックしてこの **.scenario** ファイルを実行します。プロジェクトパッケージに保存した **.scenario** が複数ある場合は、Run all scenarios をクリックして保存したすべてのシナリオを実行します。Run scenario オプションでは、対象の **.scenario** ファイルを保存する必要はありませんが、Run all scenarios オプションを使用する場合は、すべての **.scenario** ファイルを保存する必要があります。
   テストに失敗したら、ウィンドウ下部の Alerts メッセージに記載されている問題に対応し、シナリオの全コンポーネントを見直し、エラーが表示されなくなるまで妥当性確認を行います。

7. 変更がすべて終了したら、テストシナリオデザイナーで Save をクリックして、設定した内容を保存します。

第17章 次のステップ

『Red Hat Process Automation Manager プロジェクトのパッケージ化およびデプロイ』

付録A バージョン情報

本ドキュメントの最終更新日: 2020 年 3 月 18 日 (水)