Red Hat Process Automation Manager 7.6

スプレッドシート形式の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

Last Updated: 2020-02-24

Red Hat Process Automation Manager 7.6 スプレッドシート形式の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

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

法律上の通知

概要

本書では、Red Hat Process Automation Manager 7.6 で、スプレッドシート形式のデシジョンテーブルを使用してデシジョンサービスを設計する方法を説明します。

目次

前書き

ビジネスアナリストまたはビジネスルールの開発者は、スプレッドシートのデシジョンテーブル内に、テーブル形式でビジネスルールを定義し、Business Central のプロジェクトにスプレッドシートをアップロードできます。これらのルールは、Drools ルール言語 (DRL) にコンパイルされて、プロジェクトのデシジョンサービスの中心となります。

注記

ルールベースやテーブルベースのアセットではなく、Decision Model and Notation (DMN) モデルを使用してデシジョンサービスを設計することもできます。Red Hat Process Automation Manager 7.6 の DMN サポートに関する詳細は、以下の資料を参照してください。

- 『Getting started with decision services』(DMN デシジョンサービスの例を使用したステップバイステップのチュートリアル)
- 『Designing a decision service using DMN models』(Red Hat Process Automation Manager の DMN サポートおよび機能の概要)

前提条件

- デシジョンテーブルのスペースおよびプロジェクトが Business Central に作成されており、各アセットが、スペースに割り当てられたプロジェクトに関連付けられている。詳細は『Getting started with decision services』を参照してください。

第1章 RED HAT PROCESS AUTOMATION MANAGER におけるデシジョン作成アセット

Red Hat Process Automation Manager は、デシジョンサービスにビジネスデシジョンを定義するのに使用可能なアセットを複数サポートします。デシジョン作成アセットはそれぞれ長所が異なるため、ゴールおよびニーズに合わせて、アセットを 1 つ、または複数を組み合わせて使用できます。

以下の表では、デシジョンサービスでデシジョンを定義する最適な方法を選択できるように、Red Hat Process Automation Manager プロジェクトでサポートされている主要なデシジョン作成アセットを紹介します。

表1.1 Red Hat Process Automation Manager でサポートされるデシジョン作成アセット

アセット	主な特徴	オーサリングツール	ドキュメンテーション
DMN (Decision Model and Notation) モデル	• Object Management Group (OMG) が定義する標準記法をもとにしたデシジョンモデルである • 1 つまたは複数の意思決定要件グラフ (DRG: decision requirements graphs) を含むグラフィカルな意思決定要件ダイアグラム (DRD: decision requirements diagrams) を使用してビジネスデシジョンのフローを追跡する • DMN モデルが DMN 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する • DMN デシジョンテーブルおよび他の DMN ボックス式 (Boxed Expression) でデシジョンロジックを定義する Friendly Enough Expression Language (FEEL) をサポートする • Business Process Model and Notation (BPMN) プロセスモデルと効率的に統合できる • 包括性、具体性および安定性のあるデシジョンフローの作成に最適である	Business Central または DMN 準拠のエディター	『Designing a decision service using DMN models』

アセット	主な特徴	オーサリングツール	ドキュメンテーション
ガイド付きデシジョンテーブル	• Business Central の UI ベースのテーブルデザイナーで作成するルールのテーブル • デシジョンテーブルをスプレッドシートで対応する代わりにウィザードで対応する • 条件を満たした入力に、フィールドとオプションを提供する • ルールテンプレートを作成するテンプレートキーと値をサポートする • その他のアセットではサポートされていない、ヒットポリシー、リアルタイム検証などの追加機能をサポートする • コンパイルエラーを最小限に抑えるため、制限されているテーブル形式でルールを作成するのに最適	Business Central	『Designing a decision service using guided decision tables』
スプレッドシートのデシジョンテーブル	• Business Central にアップロード可能な XLS または XLSX スプレッドシート形式のデシジョンテーブルである • ルールテンプレートを作成するテンプレートキーと値をサポートする • Business Central 外で管理しているデシジョンテーブルでルールを作成するのに最適 • アップロード時に適切にルールをコンパイルするために厳密な構文要件がある	スプレッドシートエディター	『Designing a decision service using spreadsheet decision tables』

アセット	主な特徴	オーサリングツール	ドキュメンテーション
ガイド付きルール	• Business Central の UI ベースのルールデザイナーで作成する個々のルール • 条件を満たした入力に、フィールドとオプションを提供する • コンパイルエラーを最小限に抑えるため、制御されている形式で単独のルールを作成するのに最適	Business Central	『Designing a decision service using guided rules』
ガイド付きルールテンプレート	• Business Central の UI ベースのテンプレートデザイナーで作成する再利用可能なルール構造 • 条件を満たした入力に、フィールドとオプションを提供する • (このアセットの基本となる) ルールテンプレートを作成するテンプレートキーと値をサポートする • ルール構造が同じで、定義したフィールド値が異なるルールを多数作成するのに最適	Business Central	『Designing a decision service using guided rule templates』
DRL ルール	• **.drl** テキストファイルに直接定義する個々のルール • 最も柔軟性が高く、ルールと、ルール動作に関するその他の技術を定義できる • スタンドアロン環境で作成し、Red Hat Process Automation Manager に統合可能 • 詳細にわたる DRL オプションが必要なルールを作成するのに最適 • ルールを適切にコンパイルするために厳密な構文要件がある	Business Central または統合開発環境 (IDE)	『Designing a decision service using DRL rules』

アセット	主な特徴	オーサリングツール	ドキュメンテーション
予測モデルマークアップ言語 (PMML: Predictive Model Markup Language) モデル	• Data Mining Group (DMG) で定義した標準記法を元にした予測データ分析モデルである • PMML モデルが PMML 準拠プラットフォーム間で共有できるようにする XML スキーマを使用する • 回帰、スコアカード、ツリー、マイニングなどのモデルタイプをサポートする • スタンドアロンの Red Hat Process Automation Manager プロジェクトに追加したり、Business Central のプロジェクトにインポートしたりできる • Red Hat Process Automation Manager のデシジョンサービスに予測データを統合するのに最適である	PMML または XML エディター	『Designing a decision service using PMML models』

第2章 スプレッドシートのデシジョンテーブル

スプレッドシートのデシジョンテーブルは、表形式でビジネスルールを定義する XLS または XLSX 形式のスプレッドシートです。スプレッドシートのデシジョンテーブルは、スタンドアロンの Red Hat Process Automation Manager プロジェクトに追加したり、Business Central のプロジェクトにアップロードしたりできます。スプレッドシートの各行がルールになり、各列が条件、アクション、または別のルール属性になります。スプレッドシートのデシジョンテーブルを作成してアップロードした後に、その他のすべてのルールアセットと同じように、定義したルールを Drools Rule Language (DRL) ルールにコンパイルします。

スプレッドシートのデシジョンテーブルに関連するデータオブジェクトはすべて、スプレッドシートのデシジョンテーブルと同じプロジェクトパッケージに置く必要があります。同じパッケージのアセットはデフォルトでインポートされます。その他のパッケージの既存アセットは、デシジョンテーブルを使用してインポートできます。

第3章 データオブジェクト

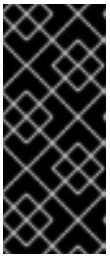
データオブジェクトは、作成するルールアセットの構成要素です。データオブジェクトは、プロジェクトで指定したパッケージに Java オブジェクトとして実装されているカスタムのデータ型です。たとえば、データフィールド **Name**、**Address**、および **DateOfBirth** を使用して **Person** オブジェクトを作成し、ローン申し込みルールに詳細な個人情報を指定できます。このカスタムのデータ型は、アセットとデシジョンサービスがどのデータに基づいているかを指定します。

3.1. 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。

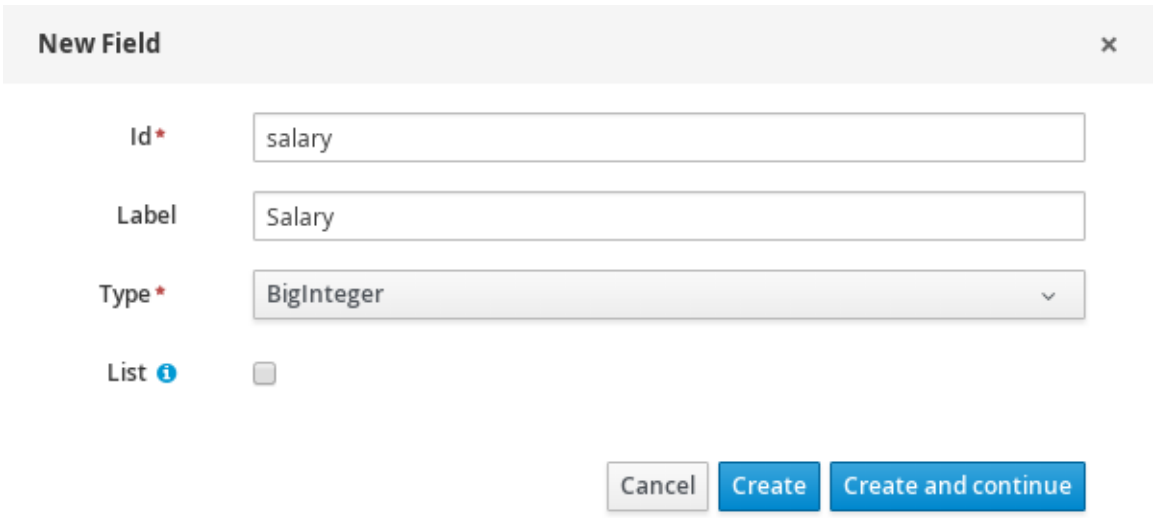


別のパッケージからのデータオブジェクトのインポート

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型を入力します。
    - **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3.1 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、新しい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作成できます。

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4章 デシジョンテーブルのユースケース

オンラインショッピングのサイトでは、注文したアイテムの配送料金が一覧表示されます。このサイトでは、以下の条件で配送料金が無料となります。

- 注文したアイテム数が 4 点以上、合計金額が 300 ドル以上。
- 配送の種類で「標準」が選択されている (購入日から 4 - 5 営業日に配送)。

この条件を適用した配送料金は以下のようになります。

表4.1 注文金額が 300 ドル未満の場合

アイテム数	配送日	配送料金 (米ドル) (N はアイテム数)
3 点以下	翌日	35
	2 日後	15
	標準	10
4 点以上	翌日	N*7.50
	2 日後	N*3.50
	標準	N*2.50

表4.2 注文金額が 300 ドルを超える場合

アイテム数	配送日	配送料金 (米ドル) (N はアイテム数)
3 点以下	翌日	25
	2 日後	10
	標準	N*1.50
4 点以上	翌日	N*5
	2 日後	N*2
	標準	無料

この条件と料金は、以下のサンプルスプレッドシートのデシジョンテーブルで紹介しています。

図4.1配送料金のデシジョンテーブル

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

この例が示すように、デシジョンテーブルを Business Central にアップロードするには、テーブルが XLS 形式または XLSX 形式のスプレッドシートで、構造と構文の要件に準拠する必要があります。詳細は「5章*スプレッドシートのデシジョンテーブルの定義*」を参照してください。

第5章 スプレッドシートのデシジョンテーブルの定義

スプレッドシート形式のデシジョンテーブル (XLS または XLSX) には、ルールデータを定義する 2 つの重要な領域、**RuleSet** 領域と **RuleTable** 領域が必要です。スプレッドシートの **RuleSet** 領域には、ルールセット名、ユニバーサルルール属性など、(このスプレッドシートだけでなく) すべてのルールをパッケージ全体に、グローバルに適用する要素を定義します。**RuleTable** 領域には、実際のルール (行) と、指定したルールセットのルールテーブルを構成する条件、アクション、その他のルール属性 (列) を定義します。スプレッドシート形式のデシジョンテーブルには **RuleTable** 領域を複数追加できますが、**RuleSet** 領域は 1 つだけとなります。

重要

通常は、デシジョンテーブルのスプレッドシートを 1 つだけアップロードする必要があります。これには、Business Central の 1 つのルールパッケージに必要なすべての **RuleTable** 定義が含まれます。異なるパッケージに複数のデシジョンテーブルのスプレッドシートをアップロードすることはできますが、同じパッケージに複数のスプレッドシートをアップロードすると、**RuleSet** 属性または **RuleTable** 属性が競合するコンパイルエラーが発生する可能性があるため、これは推奨されません。

デシジョンテーブルを定義する際は、以下のサンプルスプレッドシートを参照してください。

図5.1配送料金のスプレッドシートデシジョンテーブル例

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

手順

1. 新しいスプレッドシート (XLS または XLSX) の 2 列目または 3 列目 (サンプルの行 1) のセルに、**RuleSet** とラベルを付けます。左の列は、(任意で) 記述的メタデータに使用します。

2. 右隣のセルに、**RuleSet** の名前を入力します。このルールセットには、ルールパッケージに定義する **RuleTable** ルールがすべて含まれます。

3. **RuleSet** セルの下に、そのパッケージ内のすべてのルールテーブルにグローバルに適用するルール属性 (セルごとに 1 つ) を定義します。右のセルに属性値を指定します。たとえば、ラベルを **Import** にして、その右隣のセルに、その他のパッケージからデシジョンテーブルのパッケージにインポートするデータオブジェクトを指定します (形式は **package.name.object.name**)。サポートされるセルのラベルと値については、「RuleSet の定義」 を参照してください。

4. **RuleSet** セルと同じ列で、**RuleSet** 領域の何行か下の新しいセルに、ラベル **RuleTable** を入力し (サンプルの行 7)、テーブル名も同じセルに入力します。この名前は、区別のために追加したルールの番号とともに、このルールテーブルに指定した全ルールの名前の最初の部分として使用されます。この自動命名ルールは、**NAME** 属性列を追加すると上書きできます。

5. その下の 4 行には、必要に応じて以下の要素を定義します (サンプルの行 8-11)。

   - **ルール属性:** 条件、アクション、またはその他の属性。サポートされるセルのラベルと値については 「RuleTable の定義」 を参照してください。

   - **オブジェクトタイプ:** ルール属性が適用されるデータオブジェクト。同じオブジェクトタイプを複数の列に適用したい場合は、(サンプルのデシジョンテーブルに示されるように) 複数のオブジェクトセルを 1 つのセルに結合します。オブジェクトタイプを結合すると、結合範囲の下にあるすべての列が、1 つのパターンに指定する一連の制約になり、一度に 1 つのファクトに一致するようになります。同じオブジェクトを別の列で繰り返し使用すると、列ごとに別のパターンを作成できます。一致させるファクトは、同一にすることも別にすることもできます。

   - **制約:** オブジェクトタイプの制約。

   - **列ラベル:** (任意) 見やすくするために説明を入力する列のラベル。使用しない場合は空白にします。

   注記

   オブジェクトタイプと制約セルの両方を追加する代わりに、オブジェクトタイプのセルを空にし、対応する制約セルに完全式を追加します。たとえば、オブジェクトタイプに **Order**、制約に **itemsCount > $1** を (別々に) 追加する代わりに、オブジェクトタイプセルを空にして、制約セルに **Order(itemsCount > $1)** と入力できます。その他の制約セルでも同じです。

6. 必要なルール属性 (列) をすべて定義したら、必要に応じて各列の各行に値を入力してルールを生成します (サンプルの行 12-17)。データのないセルは無視されます (条件やアクションが適用されない場合など)。
   デシジョンテーブルのスプレッドシートにさらにルールテーブルを追加する場合がある場合は、前のテーブルの最後の行の後に 1 行空け、前のテーブルの **RuleTable** セルと **RuleSet** セルと同じ列のセルに、別の **RuleTable** のラベルを付け、このセクションの手順を繰り返して新しいテーブルを作成します (サンプルの行 19-29)。

7. XLS または XLSX のスプレッドシートを保存して終了します。

注記

Business Central にスプレッドシートをアップロードすると、スプレッドシートワークブックの最初のワークシートだけがデシジョンテーブルとして処理されます。**RuleTable** とともに使用する各 **RuleSet** の名前は、同じパッケージの全デシジョンテーブルファイルで一意にする必要があります。

Business Central でデシジョンテーブルをアップロードすると、以下の例のように、サンプルスプレッドシートのルールが DRL ルールとして表示されます。

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

セルの値に使用するホワイトスペースの有効化

デフォルトでは、デシジョンテーブルのセルの値の前後にあるホワイトスペースのは、デシジョンエンジンがデシジョンテーブルを処理する前に削除されます。セルの値の前後に意図的にホワイトスペースのを保持するには、Red Hat Process Automation Manager のディストリビューションで **drools.trimCellsInDTable** システムプロパティーを **false** に設定します。

たとえば、Red Hat Process Automation Manager と Red Hat JBoss EAP を併用するには、以下のシステムプロパティーを **$EAP_HOME/standalone/configuration/standalone-full.xml** ファイルに追加してください。

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

Java アプリケーションに埋め込まれたデシジョンエンジンを使用する場合は、以下のコマンドでシステムプロパティーを追加してください。

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

5.1. RULESET の定義

デシジョンテーブルの **RuleSet** 領域のエントリーは、(そのスプレッドシートだけでなく) パッケージのすべてのルールに適用される DRL 制約およびルール属性を定義します。 エントリーは、セルのペア (最初のセルにラベル、その右隣のセルに値) が縦方向に積み上げられます。デシジョンテーブルのスプレッドシートには、**RuleSet** 領域が 1 つだけあります。

以下の表は、**RuleSet** 定義でサポートされるラベルと値を示しています。

表5.1 サポートされる RuleSet の定義

ラベル	値	使用方法
RuleSet	生成した DRL ファイルのパッケージ名。任意。デフォルトは **rule_table**です。	最初のエントリーになります。
Sequential	**true** または **false**。**true** の場合は、ルールを上から適用する優先順位を使用します。	任意。1つまで指定可能。省略すると、適用順は指定されません。
SequentialMaxPriority	整数値	任意。1つまで指定可能。順次モードでこのオプションを使用して、優先順位の開始値を設定します。省略した場合のデフォルト値は 65535 です。
SequentialMinPriority	整数値	任意。1つまで指定可能。順次モードでこのオプションを使用して、優先順位の最低値に違反していないかどうかを確認します。省略した場合のデフォルト値は 0 です。
EscapeQuotes	**true** または **false**。**true** の場合は、引用符がエスケープされ、DRL にそのまま表示されます。	任意。1回まで指定可能。省略すると、引用符がエスケープされます。
Import	別のパッケージからインポートする、コンマ区切りの Java クラスのリスト。	任意。繰り返して使用可能。
Variables	DRL グローバルの宣言 (型に変数名が続く)。グローバル定義が複数になる場合は、コンマで区切る必要があります。	任意。繰り返して使用可能。
Functions	DRL 構文に準拠している1つまたは複数の関数定義。	任意。繰り返して使用可能。
Queries	DRL 構文に準拠している1つまたは複数のクエリー定義。	任意。繰り返して使用可能。
Declare	DRL 構文に準拠している1つまたは複数の宣言型。	任意。繰り返して使用可能。

警告

Microsoft Office、LibreOffice、および OpenOffice で二重引用符のエンコード方法が異なり、コンパイルエラーが発生する場合があります。たとえば、**“A”** は失敗しますが、**"A"** は成功します。

5.2. RULETABLE の定義

デシジョンテーブルの **RuleTable** 領域のエントリーは、そのルールテーブルのルールに対する条件、アクション、その他のルール属性を定義します。デシジョンテーブルのスプレッドシートには、**RuleTable** 領域を複数追加できます。

以下の表は、**RuleTable** 定義でサポートされるラベル (列ヘッダー) および値を示しています。列ヘッダーには、指定のラベル、またはこの表に記載されている文字で始まるカスタムラベルのいずれかを使用できます。

表5.2 サポートされる RuleTable の定義

ラベル	カスタムラベルの頭文字	値	使用方法
NAME	N	その行で生成したルールの名前を提供します。デフォルトは、**RuleTable** タグと行番号に続くテキストから作成されます。	最大1列。
DESCRIPTION	I	生成したルールのコメントになります。	最大1列。
CONDITION	C	条件内のパターンに制約を構築するコードスニペットおよび補間値。	ルールテーブルごとに最低1つ。
ACTION	A	ルールの結果に対するアクションを構築するコードスニペットおよび補間値。	ルールテーブルごとに最低1つ。
METADATA	@	ルールに対するメタデータエントリーを構築するコードスニペットおよび補間値。	任意。列の数。

以下のセクションでは、条件、アクション、メタデータのセルデータがどのように使用されるかについて説明します。

条件

CONDITION ヘッダーの列では、連続した行のセルが、条件要素になります。

- **1つ下のセル: CONDITION** のすぐ下のセルのテキストは、ルール条件のパターンを進化させ、次の行のスニペットを制約として使用します。そのセルを、隣接した1つ以上のセルと

結合した場合は、複数の制約を持った1つのパターンが作成されます。すべての制約が結合して括弧で囲まれ、このセルのテキストに追加されます。
このセルを空にすると、以下のセルのコードスニペットが自動的に有効な条件要素となります。たとえば、オブジェクトタイプに **Order**、制約に **itemsCount > $1** を (別々に) 追加する代わりに、オブジェクトタイプセルを空にして、制約セルに **Order(itemsCount > $1)** と入力できます。その他の制約セルでも同じです。

パターンのテキストの前に別のパターンを記述すれば、制約を使用しないパターンを追加できます。中が空の括弧は追加することも省くこともできます。パターンに **from** 句を追加することもできます。

パターンを **eval** で終了すると、コードスニペットが、**eval** の後の括弧の中にブール表現を生成します。

- **2つ下のセル: CONDITION** の2つ下のセルのテキストは、1つ下のセルのオブジェクト参照の制約として処理されます。このセルのコードスニペットは、その列のさらに下にあるセルから値が補間されます。下のセルからの値と **==** を使用した比較で構成される制約を作成する場合は、フィールドセレクターだけで十分です。その他の比較演算子は、スニペットの最後に指定する必要があり、値は下のセルから追加されます。その他のすべての制約形式については、**$param** シンボルを使用して、セルの内容を追加する場所を指定する必要があります。**$1**、**$2** などのシンボルを使用し、下のセルにコンマで区切った値を指定すれば、複数の値を挿入することもできます。ただし、**$1**、**$2** などをコンマで区切らないでください。テーブルの処理に失敗します。
パターン **forall($delimiter){$snippet}** に従ってテキストを展開する場合は、その下の各セルで、コンマ区切りの値に対して **$snippet** がそれぞれ1回ずつ使用されます。**$** シンボルの場所に値が挿入され、指定した **$delimiter** で結合します。**forall** 構文は、他のテキストで囲むことができることに注意してください。

  1つ下のセルにオブジェクトが含まれている場合は、そのセルから条件要素に完全なコードスニペットが追加されます。括弧のペアと、(結合したセルのパターンに複数の制約が追加されている場合は) 区切り文字のコンマが自動的に提供されます。1つ下のセルが空の場合は、このセルのコードスニペットが自動的に有効な条件要素となります。たとえば、オブジェクトタイプに **Order**、制約に **itemsCount > $1** を (別々に) 追加する代わりに、オブジェクトタイプのセルを空にして、制約セルに **Order(itemsCount > $1)** と入力できます。その他の制約セルでも同じです。

- **3つ下のセル: CONDITION** の3つ下のセルのテキストは、見やすくするためにその列の説明を入力するラベルです。

- **4つ下のセル:** 4行目以降の、空セル以外のエントリーは補間データとして提供されます。セルが空の場合は、このルールで制約や条件が省略されます。

アクション

ACTION ヘッダーの列では、連続した行のセルが、アクション命令文になります。

- **1つ下のセル: ACTION** ヘッダーの1つ下のセルのテキストは任意です。テキストがある場合は、オブジェクト参照として解釈されます。

- **2つ下のセル: ACTION** の2つ下のセルのテキストは、その列のさらに下にあるセルの値が補完されるコードスニペットです。挿入が1つの場合は、**$param** シンボルを使用して、セルの内容を追加する場所を指定します。**$1**、**$2** などのシンボルを使用し、下のセルにコンマで区切った値を指定すれば、複数の値を挿入することもできます。**$1**、**$2** などをコンマで区切らないでください。テーブルの処理に失敗します。
テキストにマーカーシンボルがない場合は、補完なしでメソッドが呼び出されます。このとき、下の行で空セル以外のエントリーを使用して、命令文を追加します。**forall** 構文がサポートされます。

1つ下のセルにオブジェクトが含まれている場合は、そのセルのテキスト、ピリオド、2つ下のセルのテキスト、終わりを示すセミコロンが1列に並べられ、アクション命令文として追加されるメソッドコールとなります。1つ下のセルが空の場合は、このセルのコードスニペットが自動的に有効なアクション要素となります。

- **3つ下のセル: ACTION** の3つ下のセルのテキストは、見やすくするためにその列の説明を入力するラベルです。
- **4つ下のセル:** 4行目以降の、空セル以外のエントリーは補間データとして提供されます。セルが空の場合は、このルールで制約や条件が省略されます。

メタデータ

ヘッダーが **METADATA** の列では、連続した行のセルが、生成されるルールのメタデータアノテーションになります。

- **1つ下のセル: METADATA** の1つ下のセルのテキストは無視されます。
- **2つ下のセル: METADATA** の2つ下のセルのテキストは、ルール行のセルの値を使用して補完されます。メタデータのマーカー文字 @ が接頭辞として自動的に追加されるため、このセルのテキストに追加する必要はありません。
- **3つ下のセル: METADATA** の3つ下のセルのテキストは、見やすくするためにその列の説明を入力するラベルです。
- **4つ下のセル:** 4行目以降の、空セル以外のエントリーは補完データとして提供されます。セルが空の場合は、このルールでメタデータアノテーションが省略されます。

5.3. RULESET 定義または RULETABLE 定義におけるその他のルール属性

RuleSet 領域および **RuleTable** 領域は、**PRIORITY**、**NO-LOOP** などのラベルおよび値もサポートします。**RuleSet** 領域に指定したルール属性は、(そのスプレッドシートだけでなく) 同じパッケージにあるすべてのルールアセットに影響します。**RuleTable** 領域に指定したルール属性は、そのルールテーブル内のルールにのみ影響します。各ルール属性は、**RuleSet** 領域で一度、**RuleTable** 領域で一度使用できます。同じ属性を、スプレッドシートの **RuleSet** 領域および **RuleTable** 領域の両方で使用した場合は、**RuleTable** が優先され、**RuleSet** 領域の属性が上書きされます。

以下の表は、その他の **RuleSet** 定義または **RuleTable** 定義でサポートされるラベル (列ヘッダー) および値を示します。列ヘッダーには、指定のラベル、またはこの表に記載されている文字で始まるカスタムラベルのいずれかを使用できます。

表5.3 RuleSet 定義または RuleTable 定義におけるその他のルール属性

ラベル	カスタムラベルの頭文字	値
PRIORITY	P	ルールの優先順位 (**salience**) の値を定義する整数。優先順位の値が高くなると、アクティベーションキューに追加された時の優先順位が高くなります。これは、**Sequential** フラグで上書きされます。 例: **PRIORITY 10**

ラベル	カスタムラベルの頭文字	値
DATE-EFFECTIVE	V	日付定義および時間定義を含む文字列。現在の日時が **DATE-EFFECTIVE** 属性よりも後の場合は、このルールがアクティブになります。 例: **DATE-EFFECTIVE "4-Sep-2018"**
DATE-EXPIRES	Z	日時定義を含む文字列。現在日時が **DATE-EXPIRES** 属性よりも後になると、このルールをアクティブにすることはできません。 例: **DATE-EXPIRES "4-Oct-2018"**
NO-LOOP	U	ブール値。このオプションを **true** に設定すると、以前一致した条件がこのルールにより再トリガーとなる場合に、このルールを再度アクティブにする (ループする) ことができません 。 例: **NO-LOOP true**
AGENDA-GROUP	G	ルールを割り当てるアジェンダグループを指定する文字列。アジェンダグループを使用すると、アジェンダをパーティションで区切り、ルールのグループに対する実行をさらに制御できます。フォーカスを取得したアジェンダグループのルールだけがアクティブになります。 例: **AGENDA-GROUP "GroupName"**
ACTIVATION-GROUP	X	ルールを割り当てるアクティベーション (または XOR) グループを指定する文字列。アクティベーショングループでアクティブにできるルールは 1 つだけです。最初のルールが実行されると、アクティベーショングループの中で、アクティベーションが保留中のルールはすべてキャンセルされます。 例: **ACTIVATION-GROUP "GroupName"**
DURATION	D	ルールの条件が一致している場合に、ルールがアクティブになってからの時間をミリ秒で定義する長整数値。 例: **DURATION 10000**
TIMER	T	ルールのスケジュールに対する **int** (間隔) または **cron** タイマー定義を指定する文字列。 例: **TIMER "*/5 * * * *"** (5 分ごと)

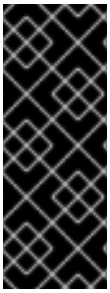
ラベル	カスタムラベルの頭文字	値
CALENDAR	E	ルールのスケジュールを指定する Quartz カレンダーの定義。 例: **CALENDAR "* * 0-7,18-23 ? * *"** (営業時間外を除く)
AUTO-FOCUS	F	アジェンダグループ内のルールにのみ適用可能なブール値。このオプションが **true** に設定されている場合は、次にルールがアクティブになったときに、そのルールが割り当てられたアジェンダグループにフォーカスが自動的に指定されます。 例: **AUTO-FOCUS true**
LOCK-ON-ACTIVE	L	ルールフローグループまたはアジェンダグループ内のルールにのみ適用可能なブール値。このオプションを **true** に設定すると、次回、ルールのルールフローグループがアクティブになるか、ルールのアジェンダグループがフォーカスを受けると、(ルールフローグループがアクティブでなくなるか、アジェンダグループがフォーカスを失うまで) ルールをアクティブにすることができません。これは、**no-loop** 属性を強力にしたものです。なぜなら、一致するルールのアクティベーションが、(ルールそのものによるものだけでなく) アップデート元にかかわらず破棄されるためです。この属性は、ファクトを修正するルールが多数あり、ルールの再一致と再発行を希望しない計算ルールに適しています。 例: **LOCK-ON-ACTIVE true**
RULEFLOW-GROUP	R	ルールフローグループを指定する文字列。ルールフローグループで、関連するルールフローによってそのグループがアクティブになった場合に限りルールを発行できます。 例: **RULEFLOW-GROUP "GroupName"**

図5.2 属性列が含まれるデシジョンテーブルのサンプルスプレッドシート

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

第6章 スプレッドシート形式のデシジョンテーブルの BUSINESS CENTRAL へのアップロード

外部の XLS ファイルまたは XLSX スプレッドシート形式のデシジョンテーブルを定義した後に、Business Central のプロジェクトに、このスプレッドシートファイルをアップロードできます。



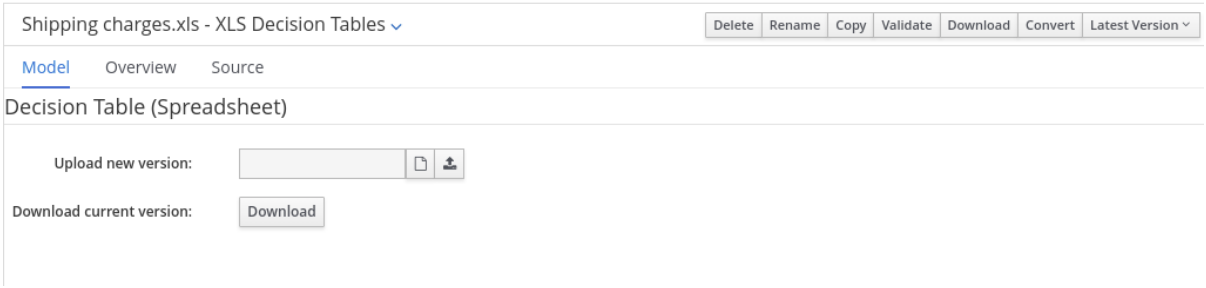
重要

通常は、デシジョンテーブルのスプレッドシートを 1 つだけアップロードする必要があります。これには、Business Central の 1 つのルールパッケージに必要なすべての **RuleTable** 定義が含まれます。異なるパッケージに複数のデシジョンテーブルのスプレッドシートをアップロードすることはできますが、同じパッケージに複数のスプレッドシートをアップロードすると、**RuleSet** 属性または **RuleTable** 属性が競合するコンパイルエラーが発生する可能性があるため、これは推奨されません。

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 → Decision Table (Spreadsheet)** をクリックします。
3. 参考となる **デシジョンテーブル** 名を入力し、適切な **パッケージ** を選択します。
4. ファイルタイプ (**xls** または **xlsx**) を選択し、**ファイルの選択** アイコンをクリックし、スプレッドシートを選択します。**OK** をクリックしてアップロードします。
5. デシジョンテーブルデザイナーの右上のツールバーで **Validate** をクリックして、テーブルを検証します。テーブル検証に失敗した場合は、XLS ファイルまたは XLSX ファイルを開いて、構文エラーに対処します。構文のヘルプは「5章*スプレッドシートのデシジョンテーブルの定義*」を参照してください。
   デシジョンテーブルの新しいバージョンのアップロード、または現行バージョンのダウンロードが可能です。

図6.1 アップロードしたデシジョンテーブルのオプション



第7章 BUSINESS CENTRAL にアップロードしたスプレッドシート形式のデシジョンテーブルの、ガイド付きデシジョンテーブルへの変換

XLS または XLSX のスプレッドシート形式のデシジョンテーブルファイルを Business Central のプロジェクトへアップロードした後、デシジョンテーブルを Business Central で直接変更できるガイド付きデシジョンテーブルに変換することができます。

ガイド付きデシジョンテーブルの詳細は、『Designing a decision service using guided decision tables』を参照してください。

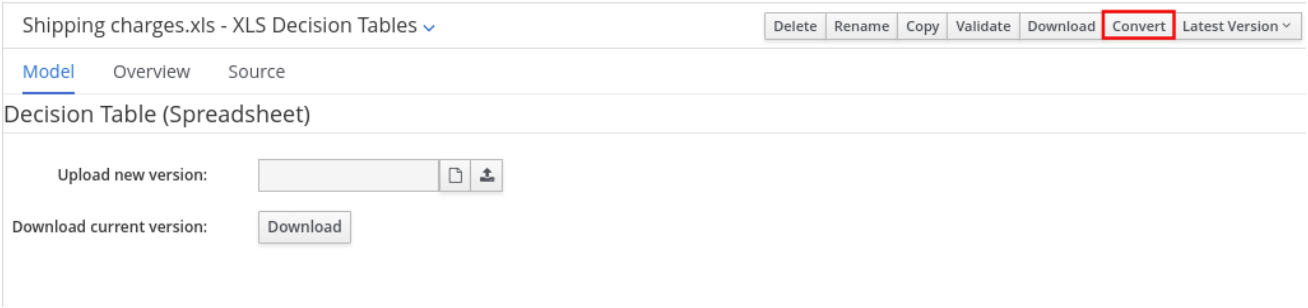
警告

ガイド付きデシジョンテーブルとスプレッドシートのデシジョンテーブルは、デシジョンテーブル形式が異なり、サポート対象機能も異なります。別のデシジョンテーブル形式に変換する場合には、サポート対象機能が両者で異なる分は変更されるか、失われることになります。

手順

Business Central で、変換するアップロードされたデシジョンテーブルアセットに移動して、デシジョンテーブルデザイナーの右上にあるツールバーで **Convert** をクリックします。

図7.1 アップロードしたデシジョンテーブルの変換



変換後のデシジョンテーブルは、Business Central で直接変更可能なプロジェクト内のガイド付きデシジョンテーブルアセットとして利用できるようになります。

第8章 ルールの実行

ルールの例を特定するか、Business Central でルールを作成したら、関連のプロジェクトをビルドしてデプロイし、ローカルまたは Process Server でルールを実行してルールをテストできます。

前提条件

- Business Central および Process Server がインストールされ、実行されている。インストールオプションは、『Planning a Red Hat Process Automation Manager installation』を参照してください。

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 **Assets** ページの右上にある **Deploy** をクリックして、プロジェクトをビルドして Process Server にデプロイします。ビルドに失敗したら、画面下部の **Alerts** パネルに記載されている問題に対処します。
   プロジェクトデプロイメントオプションに関する詳細は、『Packaging and deploying a Red Hat Process Automation Manager project』を参照してください。
3. ローカルでのルール実行に使用するか、Business Server でルールを実行するクライアントアプリケーションとして使用できるように、まだ作成されていない場合には、Process Central 外に Maven または Java プロジェクトを作成します。プロジェクトには、**pom.xml** ファイルと、プロジェクトリソースの実行に必要なその他のコンポーネントを含める必要があります。
   テストプロジェクトの例については、「"Other methods for creating and executing DRL rules"」を参照してください。
4. テストプロジェクトまたはクライアントアプリケーションの **pom.xml** ファイルを開き、以下の依存関係が追加されていない場合は追加します。
   - **kie-ci**: クライアントアプリケーションで、**ReleaseId** を使用して、Business Central プロジェクトデータをローカルにロードします。
   - **kie-server-client**: クライアントアプリケーションで、Process Server のアセットを使用してリモートに接続します。
   - **slf4j**: (オプション) クライアントアプリケーションで、Process Server に接続したあと、SLF4J (Simple Logging Facade for Java) を使用して、デバッグのログ情報を返します。

   クライアントアプリケーションの **pom.xml** ファイルにおける、Red Hat Process Automation Manager 7.6 の依存関係の例

```
<!-- For local execution -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>7.30.0.Final-redhat-00002</version>
</dependency>

<!-- For remote execution on Process Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>7.30.0.Final-redhat-00002</version>
```

```
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>org.slf4j</groupId>
  <artifactId>slf4j-simple</artifactId>
  <version>1.7.25</version>
</dependency>
```

このアーティファクトで利用可能なバージョンについては、オンラインの Nexus Repository Manager でグループ ID とアーティファクト ID を検索してください。

注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation 部品表 (BOM) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Process Decision Manager と Red Hat Process Automation Manager の両方に適用します。BOM ファイルを追加すると、指定の Maven リポジトリーからの一時的な依存関係の内、正しいバージョンが、このプロジェクトに追加されます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.6.0.redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) についての詳細情報は、「What is the mapping between Red Hat Process Automation Manager and the Maven library version?」を参照してください。

5. モジュールクラスを含むアーティファクトの依存関係が、クライアントアプリケーションの **pom.xml** ファイルに定義されていて、デプロイしたプロジェクトの **pom.xml** ファイルに記載されているのと同じであることを確認します。モデルクラスの依存関係が、クライアントアプリケーションとプロジェクトで異なると、実行エラーが発生します。
   Business Central でプロジェクトの **pom.xml** ファイルを利用するには、プロジェクトで既存のアセットを選択し、画面左側の **Project Explorer** メニューで **Customize View** ギアアイコンをクリックし、**Repository View → pom.xml** を選択します。

   たとえば、以下の **Person** クラスの依存関係は、クライアントと、デプロイしたプロジェクトの **pom.xml** ファイル両方に表示されます。

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

6. デバッグ向けロギングを行うために、**slf4j** 依存関係を、クライアントアプリケーションの

pom.xml ファイルに追加した場合は、関連するクラスパス (Maven の **src/main/resources/META-INF** 内など) に **simplelogger.properties** ファイルを作成し、以下の内容を記載します。

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

7. クライアントアプリケーションに、必要なインポートを含む **.java** メインクラスと、KIE ベースをロードする **main()** メソッドを作成し、ファクトを挿入し、ルールを実行します。
たとえば、プロジェクトの **Person** オブジェクトには、名前、苗字、時給、賃金を設定および取得するゲッターメソッドおよびセッターメソッドが含まれます。プロジェクトにある以下の **Wage** ルールでは、賃金と時給を計算し、その結果に基づいてメッセージを表示します。

```
package com.sample;

import com.sample.Person;

dialect "java"

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

(必要に応じて) Process Server の外でローカルにこのルールをテストするには、**.java** クラスで、KIE サービス、KIE コンテナー、および KIE セッションをインポートするように設定し、その後、**main()** メソッドを使用して、定義したファクトモデルに対してすべてのルールを実行します。

ローカルでルールの実行

```
import org.kie.api.KieServices;
import org.kie.api.builder.ReleaseId;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.drools.compiler.kproject.ReleaseIdImpl;

public class RulesTest {

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

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

      // Set up the fact model:
      Person p = new Person();
      p.setWage(12);
```

```
      p.setFirstName("Tom");
      p.setLastName("Summers");
      p.setHourlyRate(10);

      // Insert the person into the session:
      kSession.insert(p);

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

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

Process Server でこのルールをテストするには、ローカル例と同じように、インポートとルール実行情報で **.java** クラスを設定し、KIE サービス設定および KIE サービスクライアントの詳細を指定します。

Process Server でルールの実行

```
package com.sample;

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

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

import com.sample.Person;

public class RulesTest {

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

  public static final void main(String[] args) {
    try {
      // Define KIE services configuration and client:
      Set<Class<?>> allClasses = new HashSet<Class<?>>();
      allClasses.add(Person.class);
```

```
        String serverUrl = "http://$HOST:$PORT/kie-server/services/rest/server";
        String username = "$USERNAME";
        String password = "$PASSWORD";
        KieServicesConfiguration config =
          KieServicesFactory.newRestConfiguration(serverUrl,
                                          username,
                                          password);
        config.setMarshallingFormat(MarshallingFormat.JAXB);
        config.addExtraClasses(allClasses);
        KieServicesClient kieServicesClient =
          KieServicesFactory.newKieServicesClient(config);

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

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

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

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

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

8. 設定した **.java** クラスをプロジェクトディレクトリーから実行します。(Red Hat CodeReady Studio などの) 開発プラットフォーム、またはコマンドラインでファイルを実行できます。(プロジェクトディレクトリーにおける) Maven の実行例:

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

(プロジェクトディレクトリーにおける) Java の実行例

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

9. コマンドラインおよびサーバーログで、ルール実行のステータスを確認します。ルールが想定通りに実行されない場合は、プロジェクトに設定したルールと、メインのクラス設定を確認して、指定したデータの妥当性を確認します。

8.1. 実行可能ルールモデル

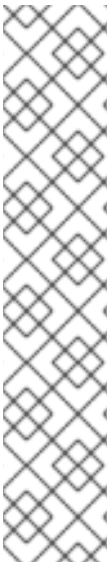
実行可能ルールモデルは埋め込み可能なモデルで、ビルド時に実行するルールセットの Java ベース表記を提供します。実行可能モデルは Red Hat Process Automation Manager の標準アセットパッケージングの代わりとなるもので、より効率的です。KIE コンテナーと KIE ベースの作成がより迅速にでき、DRL (Drools Rule Language) ファイルリストや他の Red Hat Process Automation Manager アセットが多い場合は、特に有効です。このモデルは詳細レベルにわたり、インデックス評価の lambda 表記など、必要な実行情報すべてを提供できます。

実行可能なルールモデルでは、プロジェクトにとって具体的に以下のような利点があります。

- **コンパイル時間:** 従来のパッケージ化された Red Hat Process Automation Manager プロジェクト (KJAR) には、制限や結果を実装する事前生成済みのクラスと合わせて、ルールベースを定義する DRL ファイルのリストやその他の Red Hat Process Automation Manager アーティファクトが含まれています。これらの DRL ファイルは、KJAR が Maven リポジトリーからダウンロードされて、KIE コンテナーにインストールされた時点で、解析してコンパイルする必要があります。特に大規模なルールセットの場合など、このプロセスは時間がかかる可能性があります。実行可能なモデルでは、プロジェクト KJAR 内で、Java クラスをパッケージして、プロジェクトルールベースの実行可能なモデルを実装し、はるかに早い方法で KIE コンテナーと KIE ベースをを再作成することができます。Maven プロジェクトでは、**kie-maven-plugin** を使用してコンパイルプロセス中に DRL ファイルから 実行可能なモデルソースを自動的に生成します。

- **ランタイム:** 実行可能なモデルでは、制約はすべて、Java lambda 式で定義されます。同じ lambda 式も制約評価に使用するので、**mvel** ベースの制約をバイトコードに変換するのに、解釈評価用の **mvel** 式も、Just-In-Time (JIT) プロセスも使用しません。これにより、さらに迅速で効率的なランタイムを構築できます。

- **開発時間:** 実行可能なモデルでは、DRL 形式で直接要素をエンコードしたり、DRL パーサーを対応するように変更したりする必要なく、デシジョンエンジンの新機能で開発および試行することができます。



注記

実行可能なルールモデルのクエリー定義に使用できるのは、引数最大 10 個のみです。

実行可能なルールモデルのルール結果内にある変数については、使用できるバインド変数は、最大 24 個のみとなっています (同梱されている **drools** 変数を含む)。たとえば、以下のルールの結果では、バインド変数を 25 個以上使用しているので、コンパイルエラーが発生します。

```
...
then
  $input.setNo25Count(functions.sumOf(new Object[]{$no1Count_1, $no2Count_1,
$no3Count_1, ..., $no25Count_1}).intValue());
  $input.getFirings().add("fired");
  update($input);
```

8.1.1. Maven プロジェクトへの実行可能なルールモデルの埋め込み

Maven プロジェクトに実行可能ルールモデルを埋め込み、ビルド時にルールアセットをより効率的にコンパイルすることができます。

前提条件

- Red Hat Process Automation Manager ビジネスアセットを含む Maven 化したプロジェクトがある。

手順

1. Maven プロジェクトの **pom.xml** ファイルで、パッケージタイプを **kjar** に設定し、**kie-maven-plugin** ビルドコンポーネントを追加します。

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

kjar パッケージングタイプは、**kie-maven-plugin** コンポーネントをアクティブにして、アーティファクトリソースを検証してプリコンパイルします。**<version>** は、プロジェクトで現在使用される Red Hat Process Automation Manager の Maven アーティファクトのバージョン (例: 7.30.0.Final-redhat-00002) で、これらの設定は Maven プロジェクトを適切にパッケージするために必要です。

注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation 部品表 (BOM) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Process Decision Manager と Red Hat Process Automation Manager の両方に適用します。BOM ファイルを追加すると、指定の Maven リポジトリーからの一時的な依存関係の内、正しいバージョンが、このプロジェクトに追加されます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.6.0.redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) についての詳細情報は、What is the mapping between RHPAM product and maven library version? を参照してください。

2. 以下の依存関係を **pom.xml** ファイルに追加して、ルールアセットが実行可能なモデルからビルドできるようにします。

   - **drools-canonical-model**: Red Hat Process Automation Manager から独立するルールセットモデルの実行可能な正規表現を有効にします。
   - **drools-model-compiler**: デシジョンエンジンで実行できるように Red Hat Process Automation Manager の内部データ構造に実行可能なモデルをコンパイルします。

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

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

3. コマンドターミナルで Maven 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、実行可能なモデルからプロジェクトをビルドします。

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

   -DgenerateModel=<VALUE> プロパティーで、プロジェクトが DRL ベースの KJAR ではなく、モデルベースの KJAR としてビルドできるようにします。

   <VALUE> は、3 つの値のいずれかに置き換えます。

   - **YES**: オリジナルプロジェクトの DRL ファイルに対応する実行可能なモデルを生成し、生成した KJAR から DRL ファイルを除外します。
   - **WITHDRL**: オリジナルプロジェクトの DRL ファイルに対応する実行可能なモデルを生成し、文書化の目的で、生成した KJAR に DRL ファイルを追加します (KIE ベースはいずれの場合でも実行可能なモデルからビルドされます)。
   - **NO**: 実行可能なモデルは生成されません。

   ビルドコマンドの例:

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

Maven プロジェクトのパッケージ化に関する詳細は、「Packaging and deploying a Red Hat Process Automation Manager project」を参照してください。

8.1.2. Java アプリケーションページへの実行可能なルールモデルの埋め込み

Java アプリケーションに実行可能ルールモデルをプログラミングを使用して埋め込み、ビルド時にルールアセットをより効率的にコンパイルすることができます。

前提条件

- Red Hat Process Automation Manager ビジネスアセットを含む Java アプリケーションがある。

手順

1. Java プロジェクトの適切なクラスパスに、以下の依存関係を追加します。

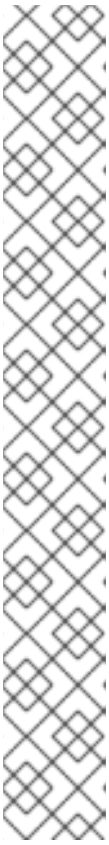
   - **drools-canonical-model**: Red Hat Process Automation Manager から独立するルールセットモデルの実行可能な正規表現を有効にします。

   - **drools-model-compiler**: デシジョンエンジンで実行できるように Red Hat Process Automation Manager の内部データ構造に実行可能なモデルをコンパイルします。

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

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

<version> は、プロジェクトで現在使用する Red Hat Process Automation Manager の Maven アーティファクトのバージョンです (例: 7.30.0.Final-redhat-00002)。



注記

個別の依存関係に対して Red Hat Process Automation Manager **<version>** を指定するのではなく、Red Hat Business Automation 部品表 (BOM) の依存関係をプロジェクトの **pom.xml** ファイルに追加することを検討してください。Red Hat Business Automation BOM は、Red Hat Process Decision Manager と Red Hat Process Automation Manager の両方に適用します。BOM ファイルを追加すると、指定の Maven リポジトリーからの一時的な依存関係の内、正しいバージョンが、このプロジェクトに追加されます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.6.0.redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) についての詳細情報は、What is the mapping between RHPAM product and maven library version? を参照してください。

2. ルールアセットを KIE 仮想ファイルシステム **KieFileSystem** に追加して、**KieBuilder** に **buildAll(ExecutableModelProject.class)** を指定して使用し、実行可能なモデルからアセットをビルドします。

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

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

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

実行可能なモデルから **KieFileSystem** をビルドした後に、作成された **KieSession** は効率のあまりよくない **mvel** 式ではなく、lambda 式をもとにした制約を使用します。**buildAll()** に引数が含まれていない場合には、プロジェクトは実行可能なモデルのない標準の手法でビルドされます。

KieFileSystem を使用する代わりに、実行可能なモデルを作成する手作業の多い別の方法として、Fluent API で **Model** を定義し、そこから **KieBase** を作成することができます。

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

Java アプリケーション内でプロジェクトをプログラミングを使用してパッケージ化する方法の詳細は、『Packaging and deploying a Red Hat Process Automation Manager project』を参照してください。

第9章 次のステップ

- 『Testing a decision service using test scenarios』
- 『Packaging and deploying a Red Hat Process Automation Manager project』

付録A バージョン情報

本書の最終更新日: 2019 年 10 月 31 日 (木)