Red Hat Process Automation Manager 7.7

ガイド付きルールを使用したデシジョンサービスの作成

Last Updated: 2020-03-31

Red Hat Process Automation Manager 7.7 ガイド付きルー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

Copyright © 2020 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at
http://creativecommons.org/licenses/by-sa/3.0/
. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux ® is the registered trademark of Linus Torvalds in the United States and other countries.

Java ® is a registered trademark of Oracle and/or its affiliates.

XFS ® is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL ® is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js ® is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack ® Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

概要

本書は、Red Hat Process Automation Manager 7.7 で、ガイド付きルールを使用してデシジョンサービスを作成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 でガイド付きルールデザイナーを使用してビジネスルールを定義できます。このガイド付きルールは Drools Rule Language (DRL) に組み込まれ、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 7.7 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。

- 『**デシジョンサービスのスタートガイド**』(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)
- 『**DMN モデルを使用したデシジョンサービスの作成**』(Red Hat Process Automation Manager の DMN サポートおよび機能の概要)

前提条件

- ガイド付きルールのチームおよびプロジェクトが Business Central に作成されていて、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ます。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。

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に基づく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の意思決定フローを追跡する • 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	『DMN モデルを使用したデシジョンサービスの作成』

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に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指定する • ルールテンプレートを作成するための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	『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』
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ための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	『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』
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別ルール • 条件を満たした入力に、フィールドとオプションを指定する • 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	『ガイド付きルールを使用したデシジョンサービスの作成』

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指定する • (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ための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	『**ガイド付きルールテンプレートを使用したデシジョンサービスの作成**』
DRL ルール	• **.drl** テキストファイルに直接定義する個別ルール • 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)	『**DRL ルールを使用したデシジョンサービスの作成**』

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) が定義する標準記法に基づく予測データ分析モデルである • PMML モデルを 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	『PMML モデルでのデシジョンサービスの作成』

第2章 ガイド付きルール

ガイド付きルールは、ルール作成のプロセスを提供する、Business Central の UI ベースのガイド付きルールデザイナーで作成するビジネスルールです。ガイド付きルールデザイナーを使用すると、ルールを定義するデータオブジェクトに基づいて、可能なインプットにフィールドおよびオプションを提供します。定義したガイド付きルールは、その他のすべてのルールアセットとともに Drools Rule Language (DRL) ルールにコンパイルされます。

ガイド付きルールに関連するすべてのデータオブジェクトは、ガイド付きルー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ガイド付きルールを作成したら、ガイド付きルールデザイナーの **Data Objects** タブから、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たり、 **新規アイテム** を追加してその他の既存データオブジェクトをインポートしたりできます。

第3章 データオブジェクト

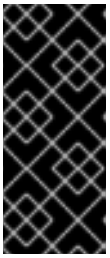
データオブジェクトは、作成するルールアセットの構成要素です。データオブジェクトは、プロジェクトで指定したパッケージに Java オブジェクトとして実装されているカスタムのデータ型です。たとえば、データフィールド **Name**、**Address**、および **DateOfBirth** を使用して **Person** オブジェクトを作成し、ローン申し込みルールに詳細な個人情報を指定できます。このカスタムのデータ型は、アセットとデシジョンサービスがどのデータに基づいているかを指定します。

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。

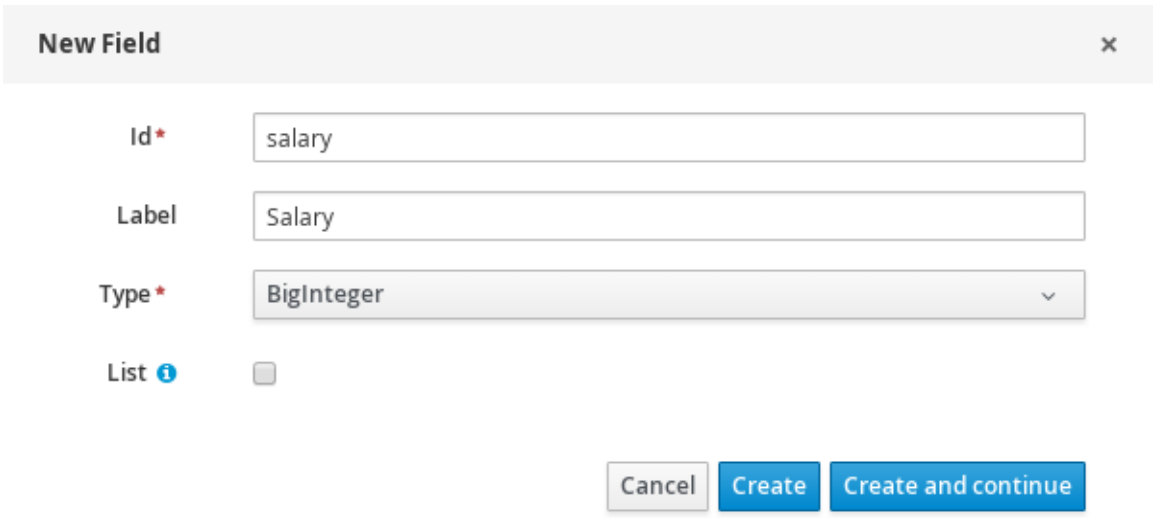


別のパッケージからのデータオブジェクトのインポート

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。

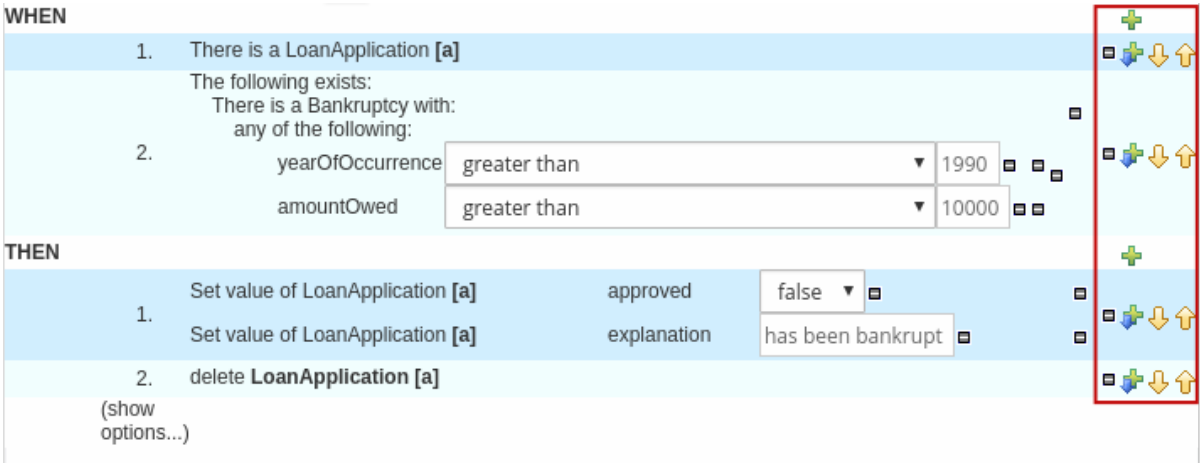
第4章 ガイド付きルールの作成

ガイド付きルールを使用すると、そのルールに関連するデータオブジェクトに基づいて、構造化フォーマットでビジネスルールを定義できるようになります。

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. **Add Asset → Guided Rule** をクリックします。
3. 参考となる **ガイド付きルール** 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、必要なデータオブジェク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。
   ドメイン固有言語 (DSL) アセットがプロジェクトに定義されている場合は、**Show declared DSL sentences** を選択することもできます。この DSL アセットは、ガイド付きルールデザイナーで定義する条件およびアクションに使用できるオブジェクトです。
4. **OK** をクリックして、ルールアセットを作成します。
   新しいガイド付きルールが、**Project Explorer** の **Guided Rules** パネルに追加されます。**Show declared DSL sentences** オプションを選択している場合は **Guided Rules (with DSL)** パネルに追加されます。
5. **Data Objects** タブをクリックして、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確認します。リストされていない場合は、**New item** をクリックして、他のパッケージから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、ガイド付きルールデザイナーの **Model** タブに戻り、ウィンドウの右側のボタンから、利用可能なデータオブジェクトに、ルールの **WHEN** (条件) セクションおよび **THEN** (アクション) セクションを追加して定義します。

図4.1 ガイド付きルールデザイナー



ルールの **WHEN** 部分には、アクションを実行するのに必要な条件が含まれます。たとえば、銀行のローン申し込みに年齢制限 (21 歳以上) が必要な場合、**Underage** ルールの **WHEN** 条件は **Age | less than | 21** となります。

ルールの **THEN** 部分には、ルールの条件部分に一致したときに実行するアクションが含まれます。たとえば、ローンの申込者が 21 歳に満たない場合は、**THEN** アクションの **approved** が **false** になり、年齢が基準に達していないためローンの申し込みが承認されません。

例外を設定して、より複雑なルールを指定することもできます。たとえば、申込者の年齢が達していなくても、保護者が関われば (保護者の同意があれば) 承認されるようにすることもできます。この場合には、**guarantor** データオブジェクトを作成またはインポートして、そのフィールドをガイド付きルールに追加します。

7. ルールのコンポーネントをすべて定義したら、ガイド付きルールデザイナーの右上のツールバーで **Validate** をクリックして、ガイド付きルールの妥当性を確認します。ルールの妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、ルール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ルールの妥当性確認を行います。

8. ガイド付きルールデザイナーで **Save** をクリックして、設定した内容を保存します。

4.1. ガイド付きルールへの WHEN 条件の追加

ルールの **WHEN** 部分には、アクションを実行するのに必要な条件が含まれます。たとえば、銀行のローン申し込みに年齢制限 (21 歳以上) が必要な場合、**Underage** ルールの **WHEN** 条件は **Age | less than | 21** となります。ルールをいつ、どのように適用するかを決定するために、単純または複雑な条件を設定できます。

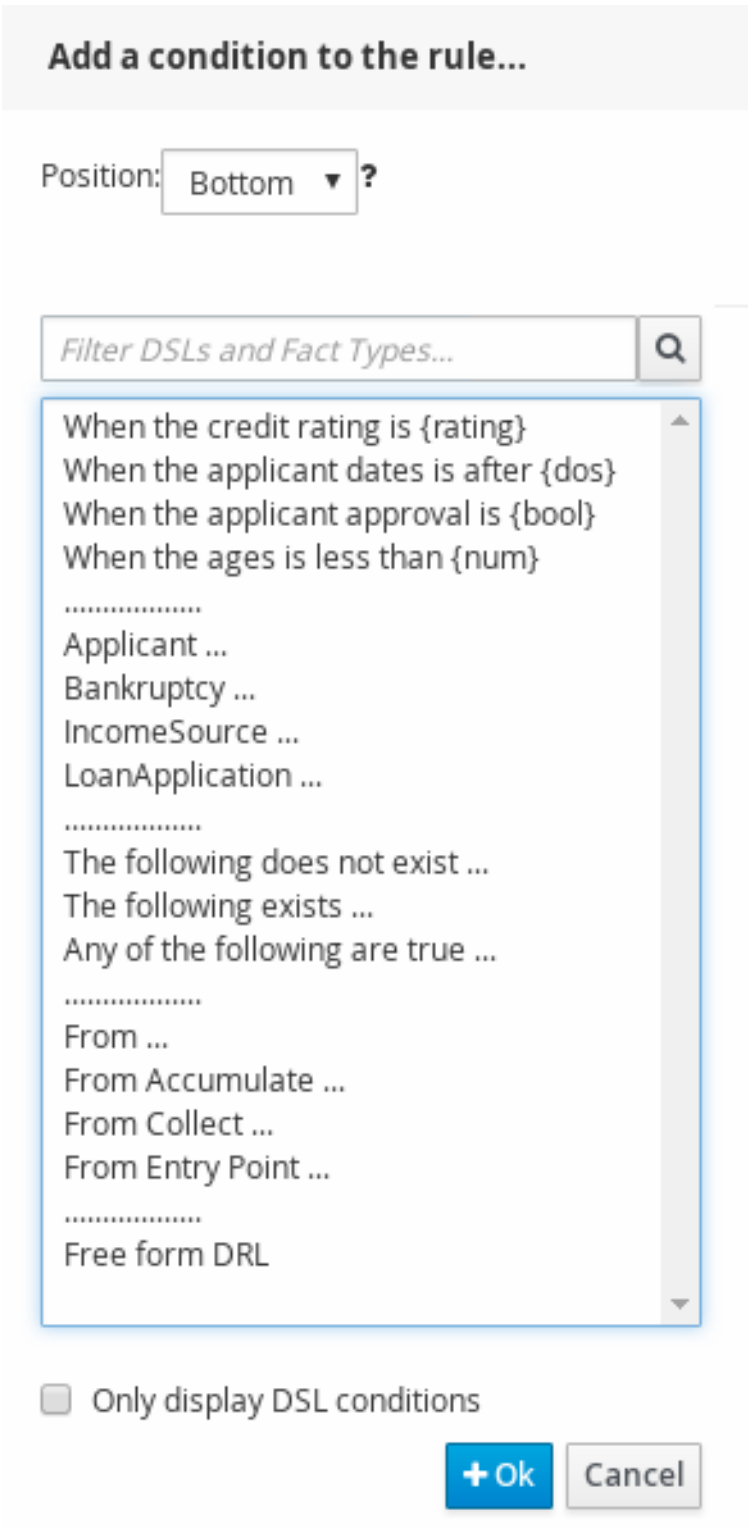
前提条件

- ルールに必要なデータオブジェクトはすべて作成、またはインポートされており、ガイド付きルールデザイナーの **Data Objects** タブにリストされている。

手順

1. ガイド付きルールデザイナーで、**WHEN** セクションの右側のプラスアイコン () をクリックします。
利用可能な条件要素が追加された **Add a condition to the rule**ウィンドウが開きます。

図4.2 ルールへの条件の追加



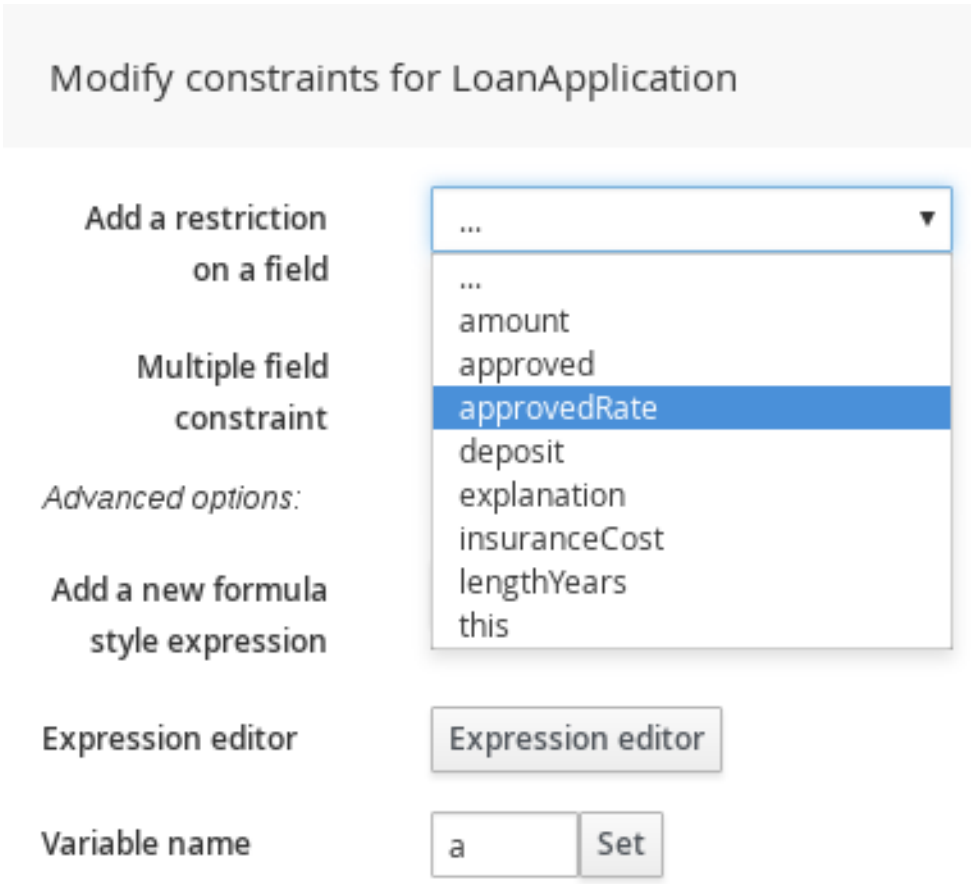
このリストには、ガイド付きルールデザイナーの **Data Objects** タブのデータオブジェクトと、パッケージに定義した DSL オブジェクト (このガイド付きルールを作成したときに **Show declared DSL sentences** を選択した場合) と、以下の標準オプションが含まれます。

- **The following does not exist:** 存在すべきでないファクトと制約を指定します。
- **The following exists:** 存在すべきファクトと制約を指定します。このオプションは、最初に一致したものだけが適用され、その後一致するものは無視されます。
- **Any of the following are true:** true であるファクトと制約をリストします。

- **From:** ルールに **From** 条件要素を定義します。
- **From Accumulate:** ルールの **Accumulate** 条件要素を定義します。
- **From Collect:** ルールの **Collect** 条件要素を定義します。
- **From Entry Point:** パターンの **Entry Point** を定義します。
- **Free form DRL:** Free form DRL フィールドを挿入します。このフィールドには、ガイド付きルールデザイナーを使用せずに、自由に条件要素を定義できます。

2. 条件要素 (**LoanApplication** など) を選択し、**OK** をクリックします。
3. ガイド付きルールデザイナーで条件要素をクリックし、**Modify constraints for LoanApplication** ウィンドウで、フィールドへの制限の追加、複数のフィールド制約の適用、新しい数式表現の追加、式エディターの適用、または変数名の設定を行います。

図4.3 条件の変更



注記

変数名を使用すると、ガイド付きルールの別の構成でファクトまたはフィールドを指定できます。たとえば、**LoanApplication** の変数を **a** とし、倒産の根拠になっている申し込みを指定する **Bankruptcy** 制約で **a** を参照します。

```
a : LoanApplication()
Bankruptcy( application == a ).
```

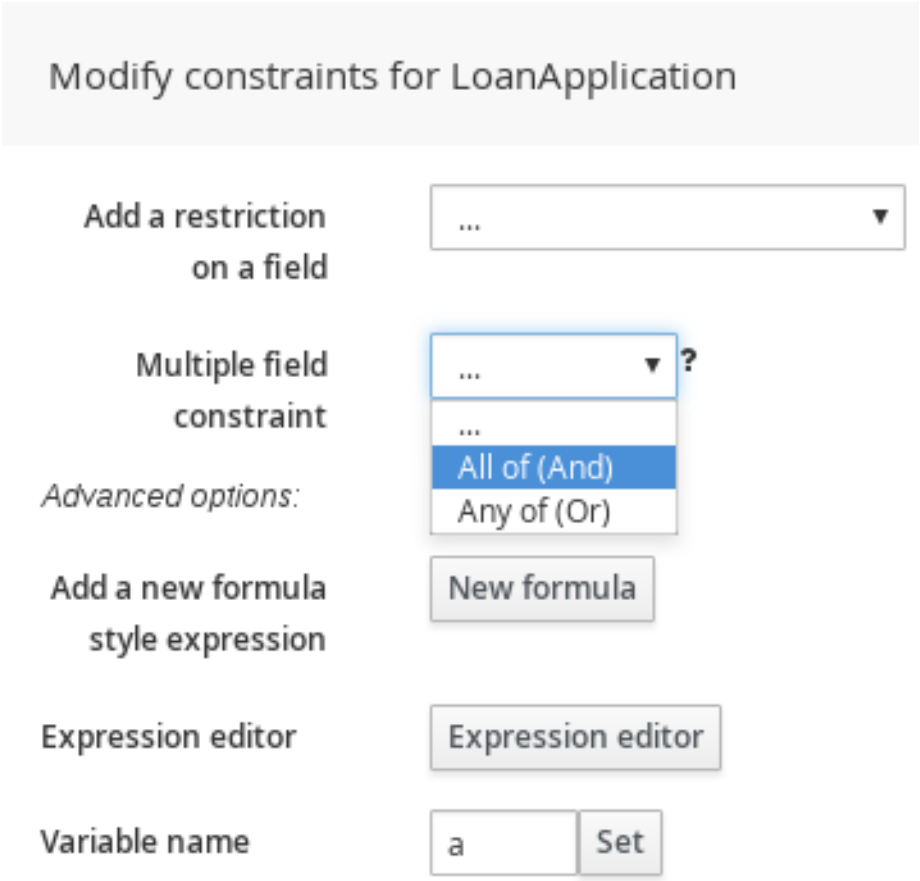
制約を選択したら、ウィンドウが自動的に閉じます。

4. 追加した制約の隣にあるドロップダウンメニューから、制限の演算子 (**greater than** など) を選択します。

5. 編集アイコン ( ) をクリックして、フィールド値を定義します。フィールド値はリテラル値、式、または完全な MVEL 表現にすることができます。

6. フィールド制約を複数適用するには、条件をクリックし、**Modify constraints for LoanApplication** ウィンドウで、**Multiple field constraint** ドロップダウンメニューから **All of(And)** または **Any of(Or)** を選択します。

図4.4 複数のフィールド制約の追加



7. ガイド付きルールデザイナーで制約をクリックして、フィールド値をさらに定義します。

8. ルールの条件コンポーネントをすべて定義したら、ガイド付きルールデザイナーの右上のツールバーで **Validate** をクリックして、ガイド付きルール条件の妥当性を確認します。ルールの妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、ルール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ルールの妥当性確認を行います。

9. ガイド付きルールデザイナーで **Save** をクリックして、設定した内容を保存します。

4.2. ガイド付きルールに THEN アクションの追加

ルールの **WHEN** 条件が一致した場合に実行するアクションがルールの **THEN** 部分に含まれます。たとえば、ローンの申込者が 21 歳に満たない場合は、**THEN** アクションにより **approved** が **false** になり、年齢が達していないためローンの申し込みが承認されません。ルールをいつ、どのように適用するかを決定するために、単純または複雑な条件を設定できます。

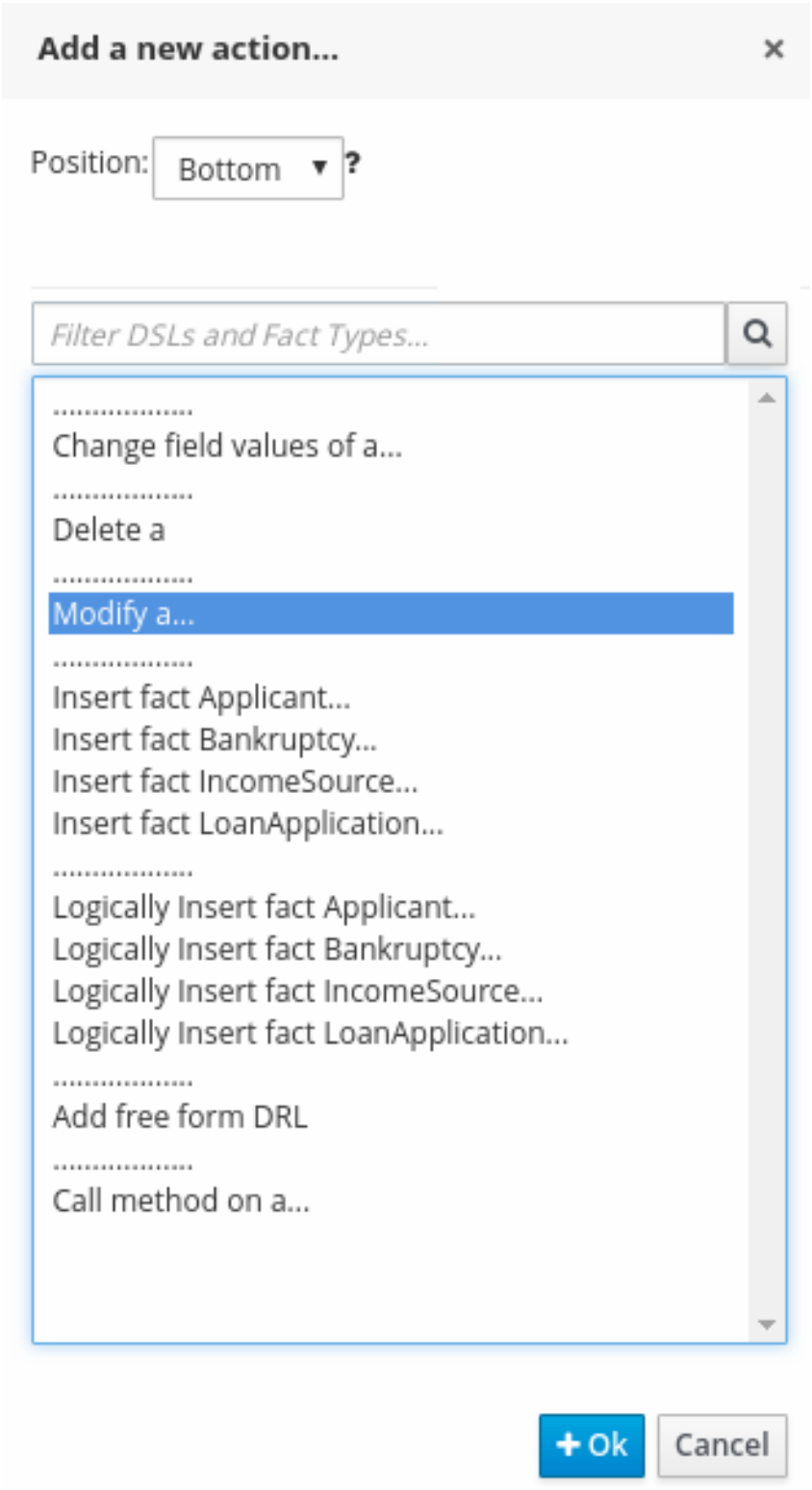
前提条件

- ルールに必要なデータオブジェクトはすべて作成、またはインポートされており、ガイド付きルールデザイナーの **Data Objects** タブにリストされている。

手順

1. ガイド付きルールデザイナーで ( ) をクリックします。
利用可能なアクション要素が追加された **Add a new action** ウィンドウが開きます。

図4.5 ルールへのアクションの追加

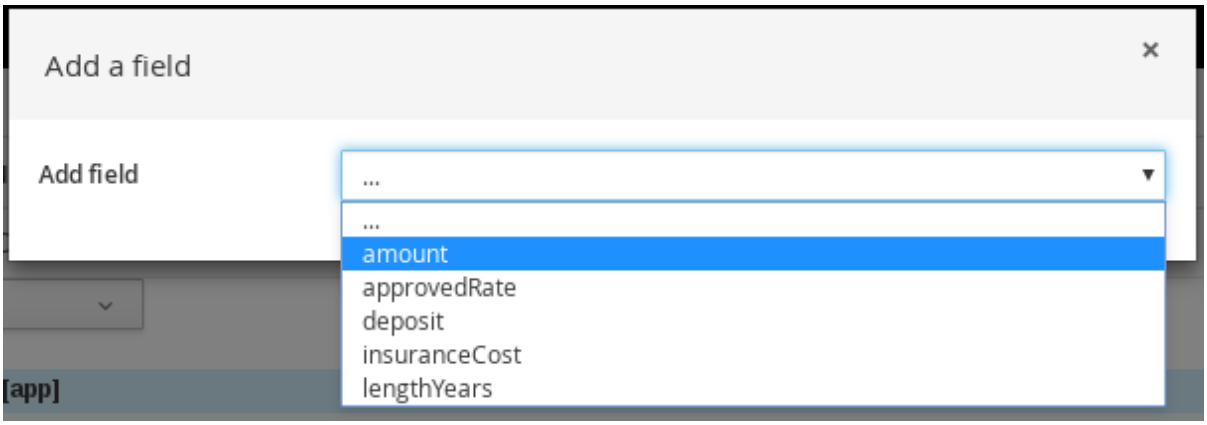


このリストには、ガイド付きルールデザイナーの **Data Objects** タブのデータオブジェクトと、パッケージに定義した DSL オブジェクト (ガイド付きルールを作成したときに **Show declared DSL sentences** を選択した場合) に基づいた挿入と修正のオプションが含まれます。

- **Change field values of:** (**LoanApplication** などの) ファクトにフィールドの値を設定します。この変更はデシジョンエンジンには通知されません。
- **Delete:** ファクトを削除します。
- **Modify:** ファクトに対して修正するフィールドを指定します。この変更はデシジョンエンジンには通知されません。
- **Insert fact (ファクトの挿入):** ファクトを挿入し、ファクトの結果フィールドと値を定義します。
- **Logically Insert fact (ファクトの論理的な挿入):** ファクトをデシジョンエンジンに論理的に挿入し、ファクトに対してフィールドと値を定義します。デシジョンエンジンは、ファクトの挿入および取り消しに対して論理的な決断を行います。定期的な挿入、または指定した挿入の後に、ファクトを明示的に取り消す必要があります。論理挿入の後に、ファクトをアサートした条件が TRUE ではなくなると、ファクトは自動的に取り消されます。
- **Add free form DRL:**Free form DRL フィールドを挿入します。このフィールドには、ガイド付きルールデザイナーを使用せずに、自由に条件要素を定義できます。
- **Call method on:** 別のファクトからメソッドを呼び出します。

2. アクション要素 (**Modify** など) を選択し、**OK** をクリックします。
3. ガイド付きルールデザイナーでアクション要素をクリックし、**Add a field** ウィンドウを使用してフィールドを選択します。

図4.6 フィールドの追加



フィールドを選択したら、ウィンドウが自動的に閉じます。

4. 編集アイコン ( ) をクリックして、フィールド値を定義します。このフィールド値は、リテラル値または式にすることができます。
5. ルールのアクションコンポーネントをすべて定義したら、ガイド付きルールデザイナーの右上のツールバーで **Validate** をクリックして、ルールのアクションの妥当性を確認します。ルールの妥当性確認に失敗したら、エラーメッセージに記載された問題に対応し、ルール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ルールの妥当性確認を行います。
6. ガイド付きルールデザイナーで **Save** をクリックして、設定した内容を保存します。

4.3. ルールアセットのドロップダウンリストの列挙定義

Business Central での列挙定義では、ガイド付きルール、ガイド付きルールテンプレート、ガイド付きデシジョンテーブルの条件やアクションのフィールドで使用可能な値を指定します。列挙定義には、ルールアセットで該当するフィールドのドロップダウンリストとして表示される対応値一覧に対する **fact.field** マッピングが含まれています。列挙定義と同じファクトとフィールドをベースにしたフィールドを選択すると、定義した値のドロップダウンリストが表示されます。

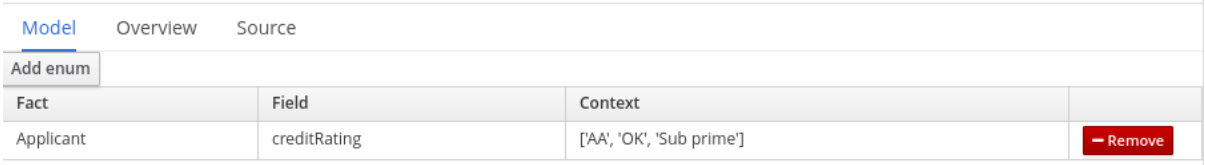
列挙は、Business Central または Red Hat Process Automation Manager プロジェクトの DRL ソースで定義できます。

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. **Add Asset → Enumerataion** をクリックします。
3. 分かりやすい **Enumeration** 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、必要なデータオブジェクトと適切なルールアセットが割り当てられているか、これから割り当てるパッケージと同じでなければなりません。
4. **Ok** をクリックして列挙を作成します。
Project Explorer の **Enumeration Definitions** パネルに、新しい列挙が追加されました。
5. 列挙デザイナーの **Model** タブで、**Add enum** をクリックし、以下の列挙値を定義します。
    - **Fact**: この列挙を関連付けるプロジェクトの同じパッケージ内に、既存のデータオブジェクトを指定します。**Project Explorer** で **Data Objects** パネルを開き、利用可能なデータオブジェクトを表示するか、必要に応じて新規アセットとして適切なデータオブジェクトを作成します。
    - **Field**: **Fact** 用に選択したデータオブジェクトの一部として定義した既存のフィールド ID を指定します。**Project Explorer** で **Data Objects** パネルを開き、適切なデータオブジェクトを選択して、利用可能な **Identifier** オプションの一覧を表示します。必要に応じて、データオブジェクトに関連する ID を作成してください。
    - **Context**: **Fact** と **Field** の定義にマッピングする **['string1','string2','string3']** または **[integer1,integer2,integer3]** 形式の値一覧を定義します。これらの値は、ルールアセットの適切なフィールドに、ドロップダウンリストとして表示されます。

    たとえば、以下の列挙は、ローン申請ディジョンサービスの申請者でクレジットスコアに使用するドロップダウンの値を定義します。

図4.7 Business Central での申請者のクレジットスコアの列挙例



DRL ソースの申請者のクレジットスコアの列挙例

```
'Applicant.creditRating' : ['AA', 'OK', 'Sub prime']
```

以下の例では、プロジェクトと同じパッケージ内にあり、**Applicant** データオブジェクトと **creditRating** フィールドを使用するガイド付きルール、ガイド付きルールテンプレートまたはガイド付きデシジョンテーブルであれば、設定値がドロップダウンオプションとして利用できます。

図4.8 ガイド付きルールまたはガイド付きルールテンプレートでの列挙ドロップダウンオプション例

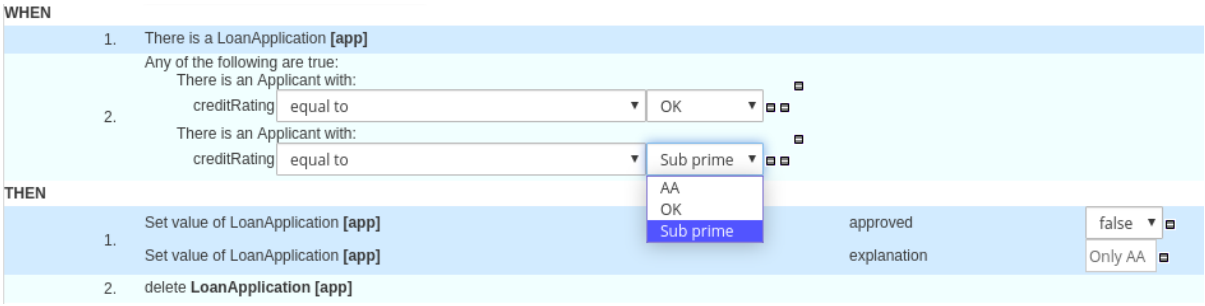
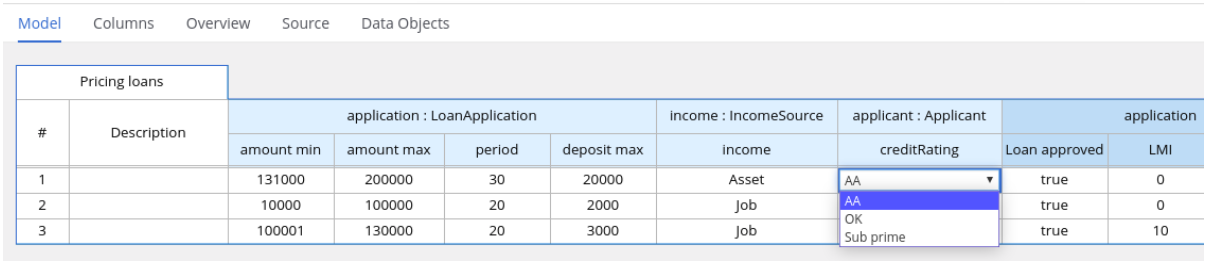


図4.9 ガイド付きデシジョンテーブルでの列挙ドロップダウンオプション例



4.3.1. ルールアセットの詳細列挙オプション

Red Hat Process Automation Manager プロジェクトの列挙定義を使用した詳細ユースケースについては、列挙を定義する時に、以下の拡張オプションの使用を検討してください。

Business Central の値との DRL の値におけるマッピング

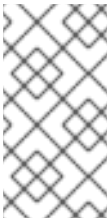
DRL ソースと Business Central インターフェースで、異なる列挙値またはより複雑な列挙値を表示するには、列挙の定義値に **'fact.field' : ['sourceValue1=UIValue1','sourceValue2=UIValue2', ...]** 形式のマッピングを使用します。
たとえば、ローンの状態に関する以下の列挙定義では、**A** または **D** のオプションを DRL ファイルで使用しますが、Business Central では **Approved** または **Declined** のオプションが表示されます。

```
'Loan.status' : ['A=Approved','D=Declined']
```

列挙値の依存関係

選択した値を 1 つのドロップダウンリストにまとめて、後続のドロップダウンリストで利用可能なオプションを判断する場合は、列挙定義で **'fact.fieldB[fieldA=value1]' : ['value2', 'value3', ...]** の形式を使用します。
たとえば、保険契約に関する以下の列挙定義では、**policyType** フィールドに **Home** または **Car** の値を使用できます。ユーザーが選択する保険契約タイプにより、利用できる契約 **coverage** のフィールドオプションが決まります。

```
'Insurance.policyType' : ['Home', 'Car']'Insurance.coverage[policyType=Home]' : ['property', 'liability']'Insurance.coverage[policyType=Car]' : ['collision', 'fullCoverage']
```



注記

列挙依存関係は、ルールの条件およびアクションをまたいで適用されません。たとえば、保険契約のユースケースでは、ルール条件で選択した契約をもとに、ルールアクションで利用可能な補償オプションが決定されるわけではありません (該当する場合)。

列挙の外部データソース

列挙定義で直接、値を定義するのではなく、外部のデータソースから列挙値の一覧を取得する場合には、プロジェクトのクラスパスで、文字列の **java.util.List** の一覧を返すヘルパークラスを追加します。列挙定義で、値を指定する代わりに、外部の値を取得するように設定したヘルパークラスを特定します。
たとえば、ローン申請者の地域に関する以下の列挙定義では、**'Applicant.region' : ['country1', 'country2', ...]** の形式で明示的に申請者の地域を定義するのではなく、外部で定義した値の一覧を返すヘルパークラスを使用します。

```
'Applicant.region' : (new com.mycompany.DataHelper()).getListOfRegions()
```

この例では、**DataHelper** クラスに、文字列の一覧を返す **getListOfRegions()**メソッドが含まれます。列挙は、ルールアセットの関連フィールドのドロップダウンリストに、読み込まれます。

また、通常通り従属フィールドを特定して、引用符でヘルパーへの呼び出しを括ることで、ヘルパークラスから動的に、従属の列挙定義を読み込むこともできます。

```
'Applicant.region[countryCode]' : '(new
com.mycompany.DataHelper()).getListOfRegions("@{countryCode}")'
```

リレーショナルデータベースなど、外部データソースから列挙データすべてを読み込む場合には、**Map<String, List<String>>** マッピングを返す、Java クラスを実装できます。マップのキーは、**fact.field** マッピングで、値は、値の **java.util.List<String>** 一覧です。

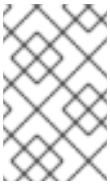
たとえば、以下の Java クラスでは、関連する列挙のローン申請者の地域を定義します。

```
public class SampleDataSource {

  public Map<String, List<String>> loadData() {
    Map data = new HashMap();

    List d = new ArrayList();
    d.add("AU");
    d.add("DE");
    d.add("ES");
    d.add("UK");
    d.add("US");
    ...
    data.put("Applicant.region", d);

    return data;
  }

}
```

以下の列挙定義は、この Java クラスの例に相関します。参照は Java クラスで定義されているので、列挙にはファクトまたはフィールド名への参照は含まれません。

```
=(new SampleDataSource()).loadData()
```

= 演算子を使用して、Business Central がヘルパークラスから全列挙データを読み込めるようにします。エディターで使用するように列挙定義を要求すると、ヘルパーメソッドが静的に評価されます。



注記

現在、Business Central では、ファクトおよびフィールド定義なしで列挙を定義することはできません。この方法で関連の Java クラスの列挙を定義するには、Red Hat Process Automation Manager プロジェクトで DRL ソースを使用します。

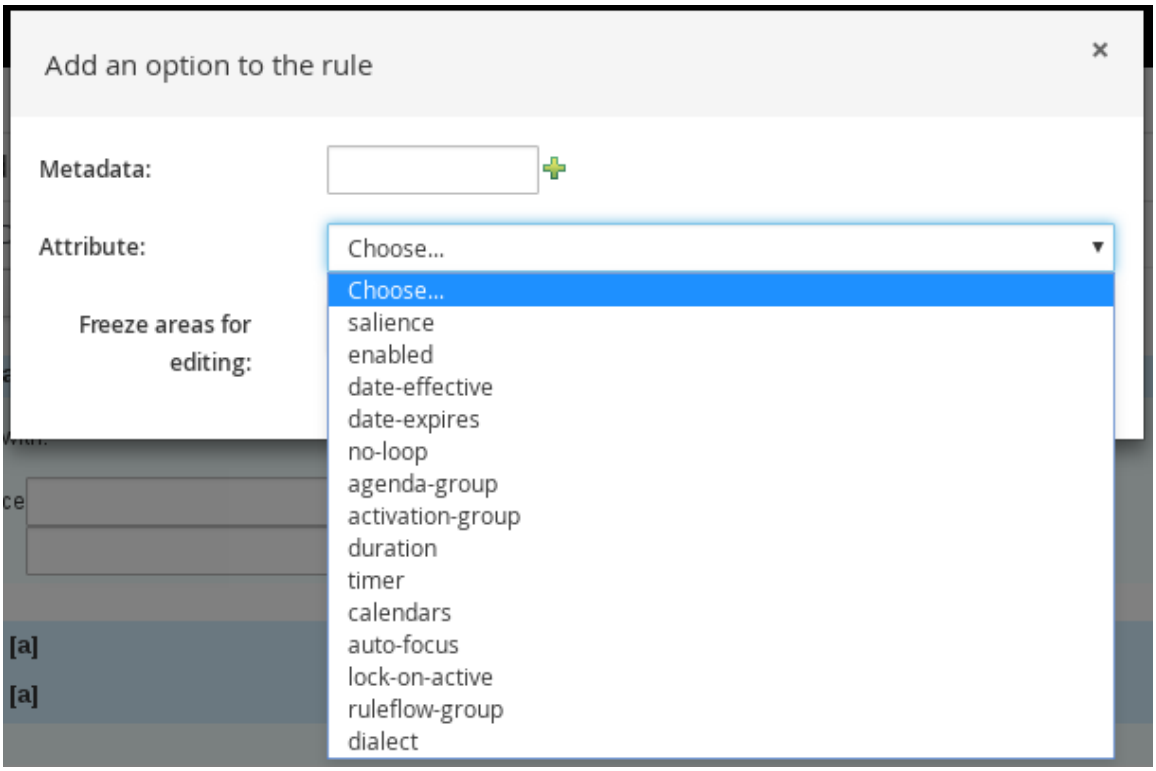
4.4. その他のルールオプションの追加

ルールデザイナーを使用してルールにメタデータを追加し、追加のルール属性 (**salience**、**no-loop** など) を定義し、条件またはアクションの変更を制限するために、ルールの領域を凍結します。

手順

1. ルールデザイナーの **THEN** セクションの下にある **(show options...)** をクリックします。
2. ウィンドウの右側にあるプラスアイコン ( ) をクリックして、オプションを追加します。
3. ルールに追加するオプションを選択します。
   - **Metadata:** メタデータのラベルを入力し、プラスアイコン ( ) をクリックします。次に、ルールデザイナーに提供されるフィールドに必要なデータを入力します。
   - **Attribute:** ルール属性のリストから選択します。次に、ルールデザイナーに表示されるフィールドまたはオプションに値を定義します。
   - **Freeze areas for editing (編集する領域を制限):**ルールデザイナーで修正する領域を制限する **条件** または **アクション** を選択します。

図4.10 ルールオプション



4. ルールデザイナーで **Save** をクリックして、設定した内容を保存します。

4.4.1. ルールの属性

ルール属性は、ルールの動作を変更するためにビジネスルールに追加できる追加の仕様です。

以下の表には、ルールに割り当て可能な属性の対応値と名前が一覧でまとめられています。

表4.1 ルールの属性

属性	値
salience	ルールの優先順位を定義する整数。ルールの salience 値を高くすると、アクティベーションキューに追加したときの優先順位が高くなります。 例: **salience 10**
enabled	ブール値。このオプションを選択すると、ルールが有効になります。このオプションを選択しないと、ルールは無効になります。 例: **enabled true**
date-effective	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"**

属性	値
date-expires	日時定義を含む文字列。現在の日時が **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	ブール値。このオプションを選択すると、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。条件を選択しないと、この状況でルールがループされます。 例: **no-loop true**
agenda-group	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	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すべてのルールの保留中のアクティベーションはすべてキャンセルされます。 例: **activation-group "GroupName"**
duration	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	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 (cron:* 0/15 * * * ?)** (15 分ごと)
calendar	ルールをスケジュールするための Quartz カレンダー定義。 例: **calendars "* * 0-7,18-23 ? * *"** (営業時間外を除く)
auto-focus	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。このオプションが選択されている場合は、次にルールがアクティブになると、そのルールが割り当てられたアジェンダグループに自動的にフォーカスが移ります。 例: **auto-focus true**

属性	値
lock-on-active	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。このオプションを選択すると、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取ると、(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。一致するルールのアクティベーションが、(ルール自体によるものだけでなく) アップデートが何に基づいて行われるか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	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、グループが関連するルールフローによってアクティブにされる場合に限りルールを発行できます。 例: **ruleflow-group "GroupName"**
dialect	ルールのコード表記に使用される言語を指定する文字列 (**JAVA** または **MVEL**)。デフォルトでは、ルールは、パッケージレベルに指定される言語を使用します。ここで指定される言語は、ルールのパッケージ言語設定を上書きします。 例: **dialect "JAVA"**

第5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 KIE Server でルールを実行してテストできます。

前提条件

- Business Central および KIE 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、『Red Hat Process Automation Manager インストールの計画』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 KIE 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のオプションに関する詳細は、『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』を参照してください。

注記

デフォルトでプロジェクト内のルールアセットが実行可能なルールモデルからビルドされていない場合には、以下の依存関係がプロジェクトの **pom.xml** ファイルに含まれているか確認して、プロジェクトを再構築してください。

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

この依存関係は、デフォルトで Red Hat Process Automation Manager のルールアセットが実行可能なルールモデルからビルドされるようにするために必要です。Red Hat Process Automation Manager のコアパッケージに、この依存関係は同梱されていますが、Red Hat Process Automation Manager のアップグレード履歴によっては、この依存関係を手動で追加して、実行可能なルールモデルの動作を有効にする必要がある場合があります。

実行可能なルールモデルに関する詳細は、『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』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、KIE 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、Business 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、「その他の DRL ルールの作成および実行方法」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

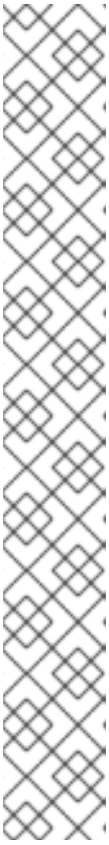
ジェクトデータをローカルにロードします。

- **kie-server-client**: クライアントアプリケーションで、KIE 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、KIE Server に接続した後に、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 7.7 の依存関係の例

```
<!-- For local execution -->
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-ci</artifactId>
  <version>7.33.0.Final-redhat-00002</version>
</dependency>

<!-- For remote execution on KIE Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>7.33.0.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>7.7.0.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、姓、時給、賃金を設定および取得する getter メソッドおよび setter 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) KIE 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するようにします。

ローカルでのルールの実行

```
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();
    }
  }
}
```

KIE Server でこのルールをテストするには、ローカルの例と同じように、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。

KIE Server でのルールの実行

```
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内の) 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内の) 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。ルールが期待通りに実行しない場合は、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、提供されるデータの妥当性を確認します。

第6章 次のステップ

- 『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 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)