Red Hat Process Automation Manager 7.12

Red Hat Process Automation Manager サービスのデプロイおよび管理

Last Updated: 2023-04-08

Red Hat Process Automation Manager 7.12 Red Hat Process Automation Manager サービスのデプロイおよび管理

法律上の通知

概要

本ガイドでは、Business Central インターフェイスまたは KIE API を使用して、Red Hat Process Automation Manager のプロジェクトおよびアセットをデプロイおよび管理する方法を説明します。

目次

はじめに

ビジネス意思決定およびプロセスの作成者は、Red Hat Process Automation Manager に作成したサービスの使用を開始するために、作成した Red Hat Process Automation Manager プロジェクトを KIE Server にデプロイする必要があります。Business Central インターフェイスまたは KIE API を使用して、Red Hat Process Automation Manager のプロジェクトおよびアセットをデプロイして管理できます。

多様性を受け入れるオープンソースの強化

Red Hat では、コード、ドキュメント、Web プロパティーにおける配慮に欠ける用語の置き換えに取り組んでいます。まずは、マスター (master)、スレーブ (slave)、ブラックリスト (blacklist)、ホワイトリスト (whitelist) の 4 つの用語の置き換えから始めます。この取り組みにより、これらの変更は今後の複数のリリースに対して段階的に実施されます。詳細は、**弊社の CTO である Chris Wright のメッセージ**を参照してください。

パート I. 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

ビジネスルール開発者は、Red Hat Process Automation Manager に作成したサービスの使用を開始するために、開発した Red Hat Process Automation Manager プロジェクトを KIE Server にビルドしてデプロイする必要があります。プロジェクトの開発およびデプロイメントには、Business Central、独立した Maven プロジェクト、Java アプリケーション、もしくは複数のプラットフォームを組み合わせて使用できます。たとえば、Business Central のプロジェクトを開発して、KIE Server REST API を使用してデプロイしたり、Business Central で設定した Maven にプロジェクトを開発して、Business Central を使用してデプロイしたりできます。

前提条件

- デプロイするプロジェクトを開発してテストしている。Business Central プロジェクトの場合は、テストシナリオを使用してプロジェクトのアセットをテストすることを検討してください。**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** を参照してください。

第1章 RED HAT PROCESS AUTOMATION MANAGER プロジェクトパッケージ

Red Hat Process Automation Manager プロジェクトには、Red Hat Process Automation Manager で開発するビジネスアセットが含まれます。Red Hat Process Automation Manager の各プロジェクトは、ナレッジ JAR (KJAR) ファイルとしてパッケージングされますが、その際にはプロジェクトのビルド、環境などの情報が含まれる Maven プロジェクトオブジェクトモデルファイル (**pom.xml**)、プロジェクトのアセットに対する KIE ベースおよび KIE セッションの設定が含まれる KIE モジュール記述子ファイル (**kmodule.xml**) などの設定ファイルが使用されます。KJAR ファイルから、パッケージ化した KJAR ファイルを、デシジョンサービス、プロセスアプリケーション、その他のデプロイ可能なアセット (総称して **services**) を実行する KIE Server にデプロイします。このようなサービスは、ランタイム時に、インスタンス化した KIE コンテナー、または **デプロイメントユニット** を介して使用されます。プロジェクトの KJAR ファイルは Maven リポジトリーに保存され、**GroupId**、**ArtifactId**、および **Version** (GAV) の 3 つの値で識別されます。この **Version** 値は、デプロイする可能性がある新しい全バージョンに対して一意である必要があります。(KJAR ファイルを含む) アーティファクトを識別するには、3 つの GAV 値がすべて必要になります。

Business Central のプロジェクトは、プロジェクトをビルドしてデプロイする際に自動的にパッケージ化されます。Business Central 以外のプロジェクト (独立した Maven プロジェクト、Java アプリケーションのプロジェクトなど) をビルドしてデプロイする場合は、追加した **kmodule.xml** ファイルに KIE モジュール記述子設定を追加するか、Java アプリケーションに直接指定する必要があります。

第2章 BUSINESS CENTRAL でのプロジェクトデプロイメント

Business Central を使用してビジネスアセットおよびサービスを開発し、プロジェクトデプロイメントに設定した KIE Server インスタンスを管理できます。プロジェクトを開発する際に、Business Central にプロジェクトをビルドして、KIE Server に自動的にデプロイできます。自動デプロイメントを有効にするために、Business Central には Maven リポジトリーが組み込まれています。Business Central から、ビルドしてデプロイしておいたサービスおよびプロジェクトバージョンを含むデプロイメントユニット (KIE コンテナー) を起動、停止、または削除できます。

(**Menu → Deploy → Execution Servers** で) 複数の KIE Server を同じ Business Central インスタンスに接続して、複数のサーバー設定にグループ分けすることもできます。同じサーバー設定に属するサーバーは同じサービスを実行しますが、別のサーバー設定の別のプロジェクト、または別のバージョンのプロジェクトをデプロイすることもできます。

たとえば、**Test** 設定のテストサーバーと、**Production** 設定の実稼働サーバーを使用できます。ビジネスアセットとサービスをプロジェクトに開発し、**Test** サーバー設定にプロジェクトをデプロイしてから、十分にテストしたプロジェクトのバージョンを **Production** サーバー設定にデプロイできます。このとき、プロジェクトの開発を継続するには、プロジェクト設定でバージョンを変更します。これにより、組み込み Maven リポジトリーで、新しいバージョンと古いバージョンが別のアーティファクトと見なされます。**Test** サーバー設定に新しいバージョンをデプロイし、**Production** サーバー設定で古いバージョンを実行し続けることができます。このデプロイメントプロセスは単純ですが、重要な制約があります。特に、開発者が直接プロジェクトを実稼働環境にデプロイできるため、アクセス制御は不十分です。

重要

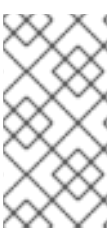
Business Central を使用して、KIE Server を別のサーバー設定に移動することはできません。サーバーの設定名を変更するには、サーバーの設定ファイルを変更する必要があります。

2.1. BUSINESS CENTRAL に接続する KIE SERVER の設定

警告

このセクションでは、テスト目的で使用可能なサンプルの設定を紹介します。一部の値は、実稼働環境には適しておらず、その旨を記載しています。

KIE Server を Red Hat Process Automation Manager 環境に設定していない場合、または Red Hat Process Automation Manager 環境に KIE Server を追加する必要がある場合は、KIE Server を設定して Business Central に接続する必要があります。

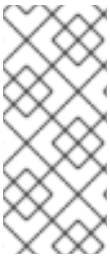


注記

Red Hat OpenShift Container Platform に KIE Server をデプロイする場合は、**Operator を使用した Red Hat OpenShift Container Platform 4 への Red Hat Process Automation Manager 環境のデプロイメント** で、Business Central に接続する設定手順を参照してください。

前提条件

- Business Central と KIE Server が Red Hat JBoss EAP インストールのベースディレクトリー (**EAP_HOME**) にインストールされている。



注記

実稼働環境では、Business Central と KIE Server は異なるサーバーにインストールする必要があります。このサンプルでは、**rest-all** と **kie-server** の両ロールを持つ **controllerUser** という名前のユーザー 1 人のみを使用します。ただし、開発環境などで、KIE Server と Business Central を同じサーバーにインストールする場合は、本セクションの説明に従って、共有の **standalone-full.xml** ファイルを変更します。

- 以下のロールを持つユーザーが存在している
  - Business Central: **rest-all** ロールを持つユーザー
  - KIE Server: **kie-server** ロールを持つユーザー

手順

1. Red Hat Process Automation Manager インストールディレクトリーで、**standalone-full.xml** ファイルに移動します。たとえば、Red Hat Process Automation Manager に Red Hat JBoss EAP インストールを使用する場合は **$EAP_HOME/standalone/configuration/standalone-full.xml** に移動します。
2. **standalone-full.xml** ファイルを開き、**<system-properties>** タグの下に、以下の JVM プロパティーを設定します。

表2.1 KIE Server インスタンスの JVM プロパティー

プロパティー	値	注記
org.kie.server.id	**default-kie-server**	KIE Server ID。
org.kie.server.controller	**http://localhost:8080/business-central/rest/controller**	Business Central の場所 Business Central の API に接続する URL。
org.kie.server.controller.user	**controllerUser**	Business Central にログイン可能な **rest-all** ロールを持つユーザー名。
org.kie.server.controller.pwd	**controllerUser1234;**	Business Central にログインできるユーザーのパスワード。
org.kie.server.location	**http://localhost:8080/kie-server/services/rest/server**	KIE Server の場所KIE Server の API に接続する URL。

表2.2 Business Central インスタンスの JVM プロパティー

プロパティー	値	注記

プロパティー	値	注記
org.kie.server.user	**controllerUser**	**kie-server** ロールを持つユーザー名。
org.kie.server.pwd	**controllerUser1234;**	ユーザーのパスワード。

以下の例は、KIE Server インスタンスを設定する方法を示しています。

```
<property name="org.kie.server.id" value="default-kie-server"/>
<property name="org.kie.server.controller" value="http://localhost:8080/business-central/rest/controller"/>
<property name="org.kie.server.controller.user" value="controllerUser"/>
<property name="org.kie.server.controller.pwd" value="controllerUser1234;"/>
<property name="org.kie.server.location" value="http://localhost:8080/kie-server/services/rest/server"/>
```

以下の例は、Business Central インスタンスに設定する方法を示しています。

```
<property name="org.kie.server.user" value="controllerUser"/>
<property name="org.kie.server.pwd" value="controllerUser1234;"/>
```

3. KIE サーバーが正常に起動したことを確認するには、KIE サーバーが動作しているときに、**http://SERVER:PORT/kie-server/services/rest/server/** に GET リクエストを送信します。KIE サーバー上での Red Hat Process Automation Manager の実行についての詳細は Running Red Hat Process Automation Managerを参照してください。
認証に成功すると、以下の例のような XML 応答が返されます。

```
<response type="SUCCESS" msg="Kie Server info">
  <kie-server-info>
    <capabilities>KieServer</capabilities>
    <capabilities>BRM</capabilities>
    <capabilities>BPM</capabilities>
    <capabilities>CaseMgmt</capabilities>
    <capabilities>BPM-UI</capabilities>
    <capabilities>BRP</capabilities>
    <capabilities>DMN</capabilities>
    <capabilities>Swagger</capabilities>
    <location>http://localhost:8230/kie-server/services/rest/server</location>
    <messages>
      <content>Server KieServerInfo{serverId='first-kie-server', version='7.5.1.Final-redhat-1', location='http://localhost:8230/kie-server/services/rest/server', capabilities=[KieServer, BRM, BPM, CaseMgmt, BPM-UI, BRP, DMN, Swagger]}started successfully at Mon Feb 05 15:44:35 AEST 2018</content>
      <severity>INFO</severity>
      <timestamp>2018-02-05T15:44:35.355+10:00</timestamp>
    </messages>
    <name>first-kie-server</name>
    <id>first-kie-server</id>
    <version>7.5.1.Final-redhat-1</version>
  </kie-server-info>
</response>
```

4. 登録が正常に完了したことを確認します。

   a. Business Central にログインします。

   b. **Menu → Deploy → Execution Servers** の順にクリックします。
   正常に登録されている場合は、登録されたサーバーの ID が表示されます。

2.2. KIE SERVER および BUSINESS CENTRAL での環境モードの設定

KIE Server は、**production** (実稼働) モードと **development** (開発) モードでの実行が設定可能です。開発モードでは、柔軟な開発ポリシーが提供され、小規模な変更の場合はアクティブなプロセスインスタンスを維持しながら、既存のデプロイメントユニット (KIE コンテナー) を更新できます。また、大規模な変更の場合は、アクティブなプロセスインスタンスを更新する前に、デプロイメントユニットの状態をリセットすることも可能です。実稼働モードは、各デプロイメントで新規デプロイメントユニットが作成される実稼働環境に最適です。

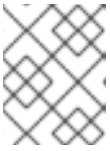
開発環境では、Business Central で **Deploy** をクリックすると、(該当する場合に) 実行中のインスタンスを中止することなくビルドした KJAR ファイルを KIE Server にデプロイすることができます。または、**Redeploy** をクリックすると、ビルドされた KJAR ファイルをデプロイしてすべてのインスタンスを置き換えることができます。次回、ビルドされた KJAR ファイルをデプロイまたは再デプロイすると、以前のデプロイメントユニット (KIE コンテナー) が同じターゲット KIE Server で自動的に更新されます。

実稼働環境では、Business Central の **Redeploy** オプションが無効になり、**Deploy** をクリックして、ビルドした KJAR ファイルを KIE Server 上の新規デプロイメントユニット (KIE コンテナー) にデプロイすることのみが可能です。

手順

1. KIE Server の環境モードを設定するには、**org.kie.server.mode** システムプロパティーを **org.kie.server.mode=development** または **org.kie.server.mode=production** に設定します。

2. Business Central のプロジェクトにデプロイメントの動作を設定するには、プロジェクトの **Settings → General Settings → Version** に移動して、**Development Mode** オプションを切り替えます。



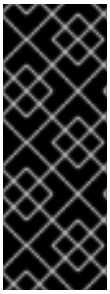
注記

デフォルトでは、KIE Server および Business Central のすべての新規プロジェクトは開発モードになっています。

Development Mode をオンにしたプロジェクトをデプロイしたり、実稼働モードになっている KIE Server に手動で **SNAPSHOT** バージョンの接尾辞を追加したプロジェクトをデプロイしたりすることはできません。

2.3. BUSINESS CENTRAL および KIE SERVER への外部 MAVEN リポジトリーの設定

Business Central および KIE Server が、内部のリポジトリーではなく、Nexus や Artifactory などの外部の Maven リポジトリーを使用するように設定できます。このように設定することで、Business Central と KIE Server は外部の Maven リポジトリーで管理されているアーティファクトにアクセスしてダウンロードできます。



重要

Maven ではアーティファクトが不変である必要があるため、リポジトリー内のアーティファクトは自動セキュリティーパッチを受け取りません。その結果、既知のセキュリティー問題のパッチがないアーティファクトはリポジトリーに残り、これらに依存するビルドが破損しないようにします。パッチが適用されたアーティファクトのバージョン番号がインクリメントされます。詳細は、JBoss Enterprise Maven リポジトリー を参照してください。

注記

Red Hat OpenShift Container Platform のオーサリング環境向けに外部の Maven リポジトリーを設定する方法については、以下のドキュメントを参照してください。

- **Operator を使用した Red Hat OpenShift Container Platform 4 への Red Hat Process Automation Manager 環境のデプロイメント**
- **テンプレートを使用した Red Hat OpenShift Container Platform 3 への Red Hat Process Automation Manager 環境のデプロイメント**

前提条件

- Business Central および KIE Server がインストールされている。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. 外部リポジトリーの接続およびアクセスの詳細が含まれる Maven **settings.xml** ファイルを作成します。**settings.xml** ファイルの詳細は Maven の Settings Reference を参照してください。
2. 既知の場所 (例: **/opt/custom-config/settings.xml**) にファイルを保存します。
3. Red Hat Process Automation Manager インストールディレクトリーで、**standalone-full.xml** ファイルに移動します。たとえば、Red Hat Process Automation Manager に Red Hat JBoss EAP インストールを使用する場合は **$EAP_HOME/standalone/configuration/standalone-full.xml** に移動します。
4. **standalone-full.xml** の **<system-properties>** タグで、**kie.maven.settings.custom** プロパティーに **settings.xml** ファイルのフルパス名を設定します。
以下に例を示します。

```
<property name="kie.maven.settings.custom" value="/opt/custom-config/settings.xml"/>
```

5. Business Central と KIE Server を起動または再起動します。

次のステップ

KJAR アーティファクトとして外部の Maven リポジトリーにエクスポートまたはプッシュする Business Central のプロジェクトごとに、プロジェクトの **pom.xml** ファイルにリポジトリーの情報を追加する必要があります。手順は、「Business Central プロジェクトの外部 Maven リポジトリーへのエクスポート」 を参照してください。

2.4. BUSINESS CENTRAL プロジェクトの外部 MAVEN リポジトリーへのエクスポート

Business Central および KIE Server 向けの外部 Maven リポジトリーを設定した場合は、外部のリポジトリーに KJAR アーティファクトとしてエクスポートまたはプッシュする Business Central プロジェクトごとに、リポジトリー情報を **pom.xml** ファイルに追加する必要があります。こうすることで、必要に応じてプロジェクトの KJAR ファイルをリポジトリーに移動して統合プロセスを実装し、Business Central または KIE Server REST API を使用して KJAR ファイルをデプロイできます。

前提条件

- Business Central と KIE Server が外部の Maven リポジトリーを使用するように設定されている。オンプレミスで Business Central をデプロイした場合の外部 Maven リポジトリーの設定に関する詳細は、「Business Central および KIE Server への外部 Maven リポジトリーの設定」を参照してください。Red Hat OpenShift Container Platform にオーサリング環境をデプロイした場合の詳細は、以下のドキュメントを参照してください。
  - **Operator を使用した Red Hat OpenShift Container Platform 4 への Red Hat Process Automation Manager 環境のデプロイメント**
  - **テンプレートを使用した Red Hat OpenShift Container Platform 3 への Red Hat Process Automation Manager 環境のデプロイメント**

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、プロジェクト内の任意のアセットを選択します。
2. 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。
3. プロジェクトの **pom.xml** ファイルの最後 (**</project>** の終了タグの前) に以下の設定を追加します。値は、**settings.xml** ファイルに定義した設定に対応する必要があります。

```
<distributionManagement>
<repository>
<id>${maven-repo-id}</id>
<url>${maven-repo-url}</url>
<layout>default</layout>
</repository>
</distributionManagement>
```

4. **Save** をクリックして **pom.xml** ファイルの変更を保存します。

外部の Maven リポジトリーに KJAR アーティファクトとしてエクスポートまたはプッシュする Business Central プロジェクトごとに、この手順を繰り返してください。

2.5. BUSINESS CENTRAL へのプロジェクトのビルドおよびデプロイメント

プロジェクトを作成したら、Business Central でプロジェクトをビルドして、設定した KIE Server にデプロイできます。プロジェクトをビルドしてデプロイする際に、Business Central のプロジェクトは、必要なすべてのコンポーネントとともに KJAR として自動的にパッケージ化されます。

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. 右上隅で **Deploy** をクリックしてプロジェクトをビルドし、KIE Server にデプロイします。KIE Server にデプロイせずにプロジェクトをコンパイルするには、**Build** をクリックします。

注記

Build & Install オプションを選択してプロジェクトをビルドし、KJAR ファイルを KIE Server にデプロイせずに設定済みの Maven リポジトリーに公開することもできます。開発環境では、**Deploy** をクリックすると、ビルドされた KJAR ファイルを KIE Server に、実行中のインスタンス (がある場合はそれ) を停止せずにデプロイできます。または **Redeploy** をクリックして、ビルドされた KJAR ファイルをデプロイしてすべてのインスタンスを置き換えることもできます。次回、ビルドされた KJAR ファイルをデプロイまたは再デプロイすると、以前のデプロイメントユニット (KIE コンテナー) が同じターゲット KIE Server で自動的に更新されます。実稼働環境では **Redeploy** オプションは無効になっており、**Deploy** をクリックして、ビルドされた KJAR ファイルを KIE Server 上の新規デプロイメントユニット (KIE コンテナー) にデプロイすることのみが可能です。

KIE Server の環境モードを設定するには、**org.kie.server.mode** システムプロパティーを **org.kie.server.mode=development** または **org.kie.server.mode=production** に設定します。Business Central の対応するプロジェクトでのデプロイメント動作を設定するには、プロジェクトの **Settings** → **General Settings** → **Version** に移動し、**Development Mode** オプションを選択します。デフォルトでは、KIE Server および Business Central のすべての新規プロジェクトは開発モードになっています。**Development Mode** をオンにしたプロジェクトをデプロイしたり、実稼働モードになっている KIE Server に手動で **SNAPSHOT** バージョンの接尾辞を追加したプロジェクトをデプロイしたりすることはできません。

Business Central に KIE Server を 1 つだけ接続する場合、または接続したすべての KIE Server が同じサーバー設定にある場合は、デプロイメントユニット (KIE コンテナー) にあるプロジェクトのサービスが自動的に起動します。

複数のサーバー設定が利用できる場合は、サーバーとデプロイメントの詳細の入力を求めるデプロイメントダイアログが Business Central に表示されます。

3. デプロイメントダイアログが表示されたら、以下の値を確認または設定します。

- **Deployment Unit Id / Deployment Unit Alias:** KIE Server でサービスを実行しているデプロイメントユニット (KIE コンテナー) の名前およびエイリアスを確認します。通常は、この設定を変更する必要はありません。KIE コンテナーのエイリアスの詳細は、「KIE コンテナーのエイリアス」 を参照してください。

- **Server Configuration:** このプロジェクトをデプロイするサーバー設定を選択します。後で、プロジェクトを再ビルドしなくても、設定したその他のサーバーにデプロイできます。

- **Start Deployment Unit?:** デプロイメントユニット (KIE コンテナー) を開始するためにこのボックスが選択されていることを確認します。このボックスの選択を解除すると、サービスはサーバーにデプロイされますが、起動されません。

プロジェクトのデプロイメントに関する詳細を確認するには、画面の上部にあるデプロイメントバナーの **View deployment details** か、**Deploy** のドロップダウンメニューをクリックします。このオプションを使用すると、**Menu** → **Deploy** → **Execution Servers** ページに移動します。

2.6. BUSINESS CENTRAL のデプロイメントユニット

設定した KIE Server 上のインスタンス化された KIE コンテナー、または **デプロイメントユニット** を介して、プロジェクト内のサービスはランタイム時に使用されます。Business Central にプロジェクトをビルドおよびデプロイすると、設定されたサーバーにデプロイメントユニットが自動的に作成されます。必要に応じて、Business Central にデプロイメントユニットを起動、停止、または削除できます。ビルドされているプロジェクトから追加デプロイメントユニットを作成したり、Business Central に設定した既存または新しい KIE Server のデプロイメントユニットを起動したりすることもできます。

2.6.1. Business Central でのデプロイメントユニットの作成

お使いの Red Hat Process Automation Manager にはすでにデプロイメントユニットが 1 つ以上あるはずですが、ない場合は、Business Central にビルドされているプロジェクトからデプロイメントユニットを作成できます。

前提条件

- 新しいデプロイメントユニットを作成するプロジェクトが Business Central にビルドされている。

手順

1. Business Central で **Menu → Deploy → Execution servers** に移動します。
2. **Server Configurations** の下で既存の設定を選択するか、**New Server Configuration** をクリックして設定を作成します。
3. **Deployment Units** の下で **Add Deployment Unit** をクリックします。
4. 必要に応じて、**Alias** フィールドにエイリアスを追加します。
5. ウィンドウのテーブルで GAV を選択し、GAV の横にある **Select** を選択して、デプロイメントユニットのデータフィールドを追加します。
6. **Start Deployment Unit?**ボックスを選択してサービスを直ちに起動するか、選択を解除して後で起動します。
7. **Finish** をクリックします。
   サービスに新しいデプロイメントユニットが作成され、このサーバー設定で指定した KIE Server に配置されました。**Start Deployment Unit?**を選択した場合は、サービスが起動します。

2.6.2. Business Central のデプロイメントユニットの起動、停止、および削除

デプロイメントユニットを起動したら、デプロイメントユニットのサービスが利用できるようになります。Business Central に KIE Server を 1 つだけ接続する場合、または接続したすべての KIE Server が同じサーバー設定にある場合は、プロジェクトのデプロイ時に、デプロイメントユニットでサービスが自動的に起動します。複数のサーバー設定が利用可能な場合は、デプロイメント時に、サーバーとデプロイメントの詳細を指定して、デプロイメントユニットを起動するように求められます。ただし、必要に応じていつでも手動で Business Central でデプロイメントユニットを起動、停止、または削除して、デプロイしたサービスを管理できます。

手順

1. Business Central で **Menu → Deploy → Execution servers** に移動します。

2. **Server Configurations** の下で、設定を選択します。
3. **Deployment Units** の下で、デプロイメントユニットを選択します。
4. 右上の **Start**、**Stop**、または **Remove** をクリックします。実行中のデプロイメントユニットを削除する場合は、停止してから削除します。

2.6.3. KIE コンテナーのエイリアス

KIE コンテナー (デプロイメントユニット) のエイリアスは、KIE Server インスタンスのプロキシーとして、同じコンテナーをデプロイする場合に異なるバージョンを処理しやすくします。単一のエイリアスを異なるバージョンのコンテナーにリンクできます。コンテナーをアップグレードすると、リンクされたエイリアスはコンテナーの新規バージョンを自動的に参照します。KIE コンテナーエイリアスの作成に関する詳細は、「Business Central でのデプロイメントユニットの作成」 を参照してください。

たとえば、新しいバージョンのコンテナーをデプロイするたびにクライアントアプリケーションが変わる場合は、クライアントアプリケーションがコンテナーのエイリアスを参照するようにできます。新しいコンテナーのバージョンがデプロイされると、関連付けられたエイリアスが自動的に更新され、クライアントアプリケーションを変更しなくても、全要求が自動的に新規コンテナーにルーティングされるようになります。

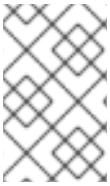
プロセスを 1 つ持ち、以下のプロパティーを使用するプロジェクト例を見ていきます。

- **GroupId**: org.jbpm
- **ArtifactId**: my-project
- **Version**: 1.0
- **containerID**: my-project

上記のプロジェクトを更新、ビルド、デプロイする場合に、関連付けられたプロジェクトは KIE Server で最新のバージョンに更新され、以下のプロパティーが含まれます。

- **GroupId**: org.jbpm
- **ArtifactId**: my-project
- **Version**: 2.0

プロジェクトの最新バージョンをデプロイする場合、**my-project** は以前のバージョンを参照しているため、**containerID** を **my-project2** に更新する必要があります。



注記

すべてのプロジェクトバージョンには、異なる **containerID** 名が含まれます。関連付けられたクライアントアプリケーションでは、対話するプロジェクトの全バージョンを認識しておく必要があります。

コンテナーエイリアスを使用すると、KIE コンテナーを管理しやすくなります。コンテナーの作成時にコンテナーのエイリアスを明示的に設定することも、関連付けられた **ArtifactId** 名をもとに暗黙的に設定することもできます。必要に応じて、複数のコンテナーにエイリアスを 1 つ追加できます。コンテナーエイリアスを指定しない場合は、プロジェクトの **ArtifactId** が、デフォルトでコンテナーエイリアスとして設定されます。

GroupId と **ArtifactId** の名前を含む複数のコンテナーにエイリアスを設定する場合、KIE Server と対話するたびに同じエイリアスを使用できます。

通常、コンテナーエイリアスは以下のユースケースで使用します。

- プロセスバージョンが最新のクライアントアプリケーションで、**新しいプロセスインスタンスを起動する**
- 特定のバージョンの **既存のプロセスを操作する**
- プロセス内の **既存のタスクを操作する**
- プロセス定義の **イメージやフォームを操作する**

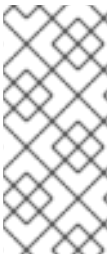
たとえば、プロジェクトのバージョン 1.0 をデプロイしたら、POST 要求を以下の KIE Server REST API エンドポイントに送信してプロジェクトのプロセスを開始します。

/http://localhost:8230/kie-server/services/rest/server/containers/my-project/processes/evaluation/instances

送信要求は、**org.jbpm:my-project:1.0** から新しいプロセスインスタンスを起動します。ここで、**my-project** はコンテナーのエイリアスとして定義しています。後に、プロジェクトのバージョン 2.0 をデプロイする場合に、同じ要求を送信すると、新規インスタンスが **org.jbpm:my-project:2.0** から起動します。**containerID** 名を追加せずに、プロセスの最新バージョンをデプロイできます。

2.7. BUSINESS CENTRAL プロジェクトの GAV 値の編集

GroupId、**ArtifactId**、および **Version** (GAV) 値は、Maven リポジトリーのプロジェクトを識別します。Business Central と KIE Server が同じファイルシステムにあり、同じ Maven リポジトリーを使用する場合は、新しいバージョンのプロジェクトをビルドするたびに、リポジトリーでプロジェクトが自動的に更新されます。ただし、Business Central と KIE Server が別のファイルシステムにあり、ローカルの Maven リポジトリーをそれぞれ使用している場合は、新しいバージョンのプロジェクトでプロジェクトの GAV 値 (通常はバージョン) を更新し、古いバージョンと新しいバージョンのプロジェクトが別のアーティファクトとして表示されるようにします。



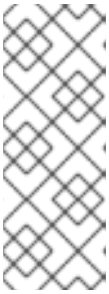
注記

開発目的でのみ、プロジェクトの **Settings** → **General Settings** → **Version** で **Development Mode** オプションを切り替えて、プロジェクトバージョンに **SNAPSHOT** の接尾辞を追加できます。この接尾辞で、Maven に対して、Maven ポリシーに従って新しいスナップショットの更新を取得するように指示します。**開発モード** を使用したり、手動で実稼働環境に **SNAPSHOT** バージョンの接尾辞を追加したりしないでください。

プロジェクトの **Settings** 画面に GAV 値を設定できます。

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 **Settings** タブをクリックします。
3. 必要に応じて、**General Settings** の **Group ID** フィールド、**Artifact ID**フィールド、または **Version** フィールドを修正します。プロジェクトをデプロイし、新しいバージョンを開発中の場合、通常はバージョン番号を変更する必要があります。



注記

開発目的でのみ、プロジェクトの **Settings → General Settings → Version** で **Development Mode** オプションを切り替えて、プロジェクトバージョンに **SNAPSHOT** の接尾辞を追加できます。この接尾辞で、Maven に対して、Maven ポリシーに従って新しいスナップショットの更新を取得するように指示します。**開発モード** を使用したり、手動で実稼働環境に **SNAPSHOT** バージョンの接尾辞を追加したりしないでください。

4. **Save** をクリックして終了します。

2.8. BUSINESS CENTRAL における重複した GAV の検出

Business Central のすべての Maven リポジトリーで、プロジェクトの **GroupId**、**ArtifactId**、および **Version** (GAV) の各値が重複しているかどうかが確認されます。GAV が重複していると、実行された操作が取り消されます。

注記

重複する GAV の検出は、**Development Mode** のプロジェクトでは無効になっています。Business Central で重複する GAV 検出を有効にするには、プロジェクトの **Settings → General Settings → Version** に移動して、**Development Mode** オプションを **OFF** (該当する場合) に切り替えます。

重複した GAV の検出は、以下の操作を実行するたびに実行されます。

- プロジェクトのプロジェクト定義の保存。
- **pom.xml** ファイルの保存。
- プロジェクトのインストール、ビルド、またはデプロイメント。

以下の Maven リポジトリーで重複の GAV が確認されます。

- **pom.xml** ファイルの **<repositories>** 要素および **<distributionManagement>** 要素で指定されたリポジトリー。
- Maven の **settings.xml** 設定ファイルに指定されたリポジトリー。

2.8.1. Business Central における重複した GAV 検出設定の管理

admin ロールを持つ Business Central ユーザーは、プロジェクトで **GroupId** 値、**ArtifactId** 値、および **Version** 値 (GAV) が重複しているかどうかを確認するリポジトリーの一覧を修正できます。

注記

重複する GAV の検出は、**Development Mode** のプロジェクトでは無効になっています。Business Central で重複する GAV 検出を有効にするには、プロジェクトの **Settings → General Settings → Version** に移動して、**Development Mode** オプションを **OFF** (該当する場合) に切り替えます。

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 **Settings** タブをクリックし、**Validation** をクリックしてリポジトリーの一覧を開きます。
3. 一覧表示したリポジトリーオプションの中から選択するか選択を解除して、重複した GAV の検出を有効または無効にします。
今後、重複した GAV の報告は、検証を有効にしたリポジトリーに対してのみ行われます。

注記

この機能を無効にするには、システムの起動時に Business Central の **org.guvnor.project.gav.check.disabled** システムプロパティーを **true** に設定します。

```
$ ~/EAP_HOME/bin/standalone.sh -c standalone-full.xml
-Dorg.guvnor.project.gav.check.disabled=true
```

第3章 BUSINESS CENTRAL を使用しないプロジェクトデプロイメント

Business Central インターフェイスにプロジェクトを開発およびデプロイする代わりに、独立した Maven プロジェクトまたは独自の Java アプリケーションを使用して、Red Hat Process Automation Manager プロジェクトを開発し、KIE コンテナー (デプロイメントユニット) のプロジェクトを、設定した KIE Server にデプロイします。KIE Server REST API を使用して、ビルドおよびデプロイしたサービスおよびプロジェクトバージョンを含む KIE コンテナーを起動、停止、または削除できます。このような柔軟性により、引き続き既存のアプリケーションのワークフローを使用して、Red Hat Process Automation Manager 機能を使用するビジネスアセットを開発できます。

Business Central のプロジェクトは、プロジェクトをビルドしてデプロイする際に自動的にパッケージ化されます。Business Central 以外のプロジェクト (独立した Maven プロジェクト、Java アプリケーションのプロジェクトなど) をビルドしてデプロイする場合は、追加した **kmodule.xml** ファイルに KIE モジュール記述子設定を追加するか、Java アプリケーションに直接指定する必要があります。

3.1. KIE モジュール記述子ファイルの設定

KIE モジュールは、追加メタデータファイル **META-INF/kmodule.xml** を持つ Maven プロジェクトまたはモジュールです。Red Hat Process Automation Manager プロジェクトを適切にパッケージングしてデプロイするには **kmodule.xml** ファイルが必要になります。この **kmodule.xml** ファイルは、プロジェクトのアセットの KIE ベースおよび KIE セッション設定を定義する KIE モジュール記述子ファイルです。KIE ベースには、Red Hat Process Automation Manager のルール、プロセス、その他のビジネスアセットがすべて含まれるリポジトリーですが、ランタイムデータは含まれません。KIE セッションは、ランタイムデータを保存および実行し、**kmodule.xml** ファイルに KIE セッションを定義した場合は KIE ベース、または KIE コンテナーから直接作成されます。

Business Central 以外のプロジェクト (独立した Maven プロジェクト、Java アプリケーションのプロジェクトなど) を作成する場合は、追加した **kmodule.xml** ファイルに KIE モジュール記述子設定を指定するか、Java アプリケーションに直接指定することでプロジェクトをビルドしてデプロイします。

手順

1. プロジェクトの **~/resources/META-INF** ディレクトリーに、最低でも以下の内容を含む **kmodule.xml** メタデータを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns="http://www.drools.org/xsd/kmodule">
</kmodule>
```

プロジェクトの **resources** パスで見つかったすべてのファイルを含むデフォルトの KIE ベースを 1 つ作成するには、この空の **kmodule.xml** ファイルで十分です。デフォルトの KIE ベースには、ビルド時にアプリケーションに KIE コンテナーを作成する際に発生するデフォルト KIE セッションも 1 つ含まれます。

以下は、より高度な **kmodule.xml** ファイルの例です。

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns="http://www.drools.org/xsd/kmodule">
  <configuration>
     <property key="drools.evaluator.supersetOf"
value="org.mycompany.SupersetOfEvaluatorDefinition"/>
  </configuration>
```

```
  <kbase name="KBase1" default="true" eventProcessingMode="cloud"
equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg1">
    <ksession name="KSession1_1" type="stateful" default="true" />
    <ksession name="KSession1_2" type="stateful" default="true" beliefSystem="jtms" />
  </kbase>
  <kbase name="KBase2" default="false" eventProcessingMode="stream"
equalsBehavior="equality" declarativeAgenda="enabled" packages="org.domain.pkg2,
org.domain.pkg3" includes="KBase1">
    <ksession name="KSession2_1" type="stateless" default="true" clockType="realtime">
      <fileLogger file="debugInfo" threaded="true" interval="10" />
      <workItemHandlers>
        <workItemHandler name="name" type="new org.domain.WorkItemHandler()" />
      </workItemHandlers>
      <listeners>
        <ruleRuntimeEventListener type="org.domain.RuleRuntimeListener" />
        <agendaEventListener type="org.domain.FirstAgendaListener" />
        <agendaEventListener type="org.domain.SecondAgendaListener" />
        <processEventListener type="org.domain.ProcessListener" />
      </listeners>
    </ksession>
  </kbase>
</kmodule>
```

この例では、KIE ベースが 2 つ定義されます。ルールアセットの特定の **パッケージ** は両方 KIE ベースに含まれます。このようにパッケージを指定した場合は、指定したパッケージを反映するディレクトリー構造にルールファイルを整理する必要があります。KIE ベース **KBase1** から 2 つの KIE セッションをインスタンス化し、**KBase2** から KIE セッションを 1 つインスタンス化します。**KBase2** の KIE セッションは **ステートレス** な KIE セッションですが、これは 1 つ前の KIE セッションで呼び出されたデータ (1 つ前のセッションの状態) が、セッションの呼び出しと呼び出しの間で破棄されることを示しています。また、その KIE セッションには、ファイル (またはコンソール) ロガー、**WorkItemHandler**、サポートされる 3 種類のリスナー (**ruleRuntimeEventListener**、**agendaEventListener**、および **processEventListener**) も指定されます。**<configuration>** 要素は、**kmodule.xml** ファイルをさらにカスタマイズするのに使用できる任意のプロパティーを定義します。

プロジェクトに **kmodule.xml** ファイルを手動で追加する代わりに、Java アプリケーションの **KieModuleModel** インスタンスを使用するか、プログラムで **kmodule.xml** ファイルを作成し、KIE ベースおよび KIE セッションを定義し、KIE 仮想ファイルシステム **KieFileSystem** に、プロジェクトのリソースをすべて追加します。

プログラムを使用して kmodule.xml を作成し、 KieFileSystem に追加

```
import org.kie.api.KieServices;
import org.kie.api.builder.model.KieModuleModel;
import org.kie.api.builder.model.KieBaseModel;
import org.kie.api.builder.model.KieSessionModel;
import org.kie.api.builder.KieFileSystem;

KieServices kieServices = KieServices.Factory.get();
KieModuleModel kieModuleModel = kieServices.newKieModuleModel();

KieBaseModel kieBaseModel1 = kieModuleModel.newKieBaseModel("KBase1")
  .setDefault(true)
  .setEqualsBehavior(EqualityBehaviorOption.EQUALITY)
  .setEventProcessingMode(EventProcessingOption.STREAM);
```

```
KieSessionModel ksessionModel1 = kieBaseModel1.newKieSessionModel("KSession1_1")
  .setDefault(true)
  .setType(KieSessionModel.KieSessionType.STATEFUL)
  .setClockType(ClockTypeOption.get("realtime"));

KieFileSystem kfs = kieServices.newKieFileSystem();
kfs.writeKModuleXML(kieModuleModel.toXML());
```

2. 手動またはプログラムで **kmodule.xml** ファイルをプロジェクトに設定したら、設定を検証する KIE コンテナーから KIE ベースおよび KIE セッションを取得します。

```
KieServices kieServices = KieServices.Factory.get();
KieContainer kContainer = kieServices.getKieClasspathContainer();

KieBase kBase1 = kContainer.getKieBase("KBase1");
KieSession kieSession1 = kContainer.newKieSession("KSession1_1"),
    kieSession2 = kContainer.newKieSession("KSession1_2");

KieBase kBase2 = kContainer.getKieBase("KBase2");
StatelessKieSession kieSession3 = kContainer.newStatelessKieSession("KSession2_1");
```

kmodule.xml ファイルに、**KieBase** または **KieSession** を **default="true"** と設定している場合は、先ほどの **kmodule.xml** 例のように、名前を渡さずに KIE コンテナーから取得できます。

```
KieContainer kContainer = ...

KieBase kBase1 = kContainer.getKieBase();
KieSession kieSession1 = kContainer.newKieSession(),
    kieSession2 = kContainer.newKieSession();

KieBase kBase2 = kContainer.getKieBase();
StatelessKieSession kieSession3 = kContainer.newStatelessKieSession();
```

Red Hat Process Automation Manager ディストリビューションの以下のシステムプロパティーの値を変更して、デシジョンエンジンにキャッシュする KIE モジュールまたはアーティファクトバージョンの最大数を増減できます。

- **kie.repository.project.cache.size**: デシジョンエンジンにキャッシュする最大 KIE モジュール数。デフォルト値は **100** です。
- **kie.repository.project.versions.cache.size**: デシジョンエンジンにキャッシュする同一のアーティファクトに対して最大指定可能なバージョン数。デフォルト値は **10** です。

KIE リポジトリー設定の完全一覧については、Red Hat カスタマーポータル から **Red Hat Process Automation Manager 7.12.0 Source Distribution** の ZIP ファイルをダウンロードして、**~/rhpam-7.12.0-sources/src/drools-$VERSION/drools-compiler/src/main/java/org/drools/compiler/kie/builder/impl/KieRepositoryImpl.java** に移動してください。

kmodule.xml ファイルの詳細は、(まだダウンロードしていない場合には) Red Hat カスタマーポータル から **Red Hat Process Automation Manager 7.12.0 Source Distribution**の ZIP ファイルをダウンロードし、**$FILE_HOME/rhpam-$VERSION-sources/kie-api-parent-$VERSION/kie-api/src/main/resources/org/kie/api/** に保存してある XML スキーマ **kmodule.xsd** を参照してください。

3.1.1. KIE モジュール設定のプロパティー

プロジェクトにおいて、KIE モジュール記述子ファイル (**kmodule.xml**) の任意の **<configuration>** 要素は、プロパティーの **キー** および **値** ペアを定義し、**kmodule.xml** ファイルをさらにカスタマイズするのに使用できます。

kmodule.xml ファイルの設定プロパティーの例

```
<kmodule>
  ...
  <configuration>
    <property key="drools.dialect.default" value="java"/>
    ...
  </configuration>
  ...
</kmodule>
```

以下は、プロジェクトの KIE モジュール記述子ファイル (**kmodule.xml**) でサポートされる **<configuration>** プロパティーのキーおよび値です。

drools.dialect.default

デフォルトの Drools 方言を設定します。
サポートされる値: **java**、**mvel**

```
<property key="drools.dialect.default"
  value="java"/>
```

drools.accumulate.function.$FUNCTION

指定した関数名に累積関数を実装するクラスのリンク。デシジョンエンジンにカスタムの累積関数を追加できます。

```
<property key="drools.accumulate.function.hyperMax"
  value="org.drools.custom.HyperMaxAccumulate"/>
```

drools.evaluator.$EVALUATION

デシジョンエンジンにカスタムのエバリュエーターを追加できるように、指定したエバリュエーター名にエバリュエーター定義を実装するクラスをリンクします。エバリュエーターは、カスタムオペレーターと類似しています。

```
<property key="drools.evaluator.soundslike"
  value="org.drools.core.base.evaluators.SoundslikeEvaluatorsDefinition"/>
```

drools.dump.dir

Red Hat Process Automation Manager の **dump/log** ディレクトリーにパスを設定します。

```
<property key="drools.dump.dir"
  value="$DIR_PATH/dump/log"/>
```

drools.defaultPackageName

プロジェクトのビジネスアセットにデフォルトパッケージを設定します。

```
<property key="drools.defaultPackageName"
  value="org.domain.pkg1"/>
```

drools.parser.processStringEscapes

文字列のエスケープ機能を設定します。このプロパティーを **false** に設定すると、**\n** 文字が改行文字として解釈されません。
サポートされる値: **true** (デフォルト)、**false**

```
<property key="drools.parser.processStringEscapes"
  value="true"/>
```

drools.kbuilder.severity.$DUPLICATE

KIE ベースがビルドされたときに報告される重複したルール、プロセス、または関数のインスタンスの重大度を設定します。たとえば、**duplicateRule** を **ERROR** に設定すると、KIE ベースのビルド時に検出された重複ルールに対してエラーが生成されます。
サポートされるキー接尾辞: **duplicateRule**、**duplicateProcess**、**duplicateFunction**

サポートされる値: **INFO**、**WARNING**、**ERROR**

```
<property key="drools.kbuilder.severity.duplicateRule"
  value="ERROR"/>
```

drools.propertySpecific

デシジョンエンジンのプロパティーの反応を設定します。
サポートされる値: **DISABLED**、**ALLOWED**、**ALWAYS**

```
<property key="drools.propertySpecific"
  value="ALLOWED"/>
```

drools.lang.level

DRL 言語レベルを設定します。
サポートされる値: **DRL5**、**DRL6**、**DRL6_STRICT** (デフォルト)

```
<property key="drools.lang.level"
  value="DRL_STRICT"/>
```

3.1.2. KIE モジュールでサポートされる KIE ベース属性

KIE ベースは、プロジェクトの KIE モジュール記述子ファイル (**kmodule.xml**) を定義するリポジトリーで、Red Hat Process Automation Manager のルール、プロセス、その他のビジネスアセットが含まれます。**kmodule.xml** ファイルで KIE ベースを定義した場合は、特定の属性および値を指定して、KIE ベース設定をさらにカスタマイズできます。

kmodule.xml ファイルの KIE ベース設定例

```
<kmodule>
  ...
  <kbase name="KBase2" default="false" eventProcessingMode="stream" equalsBehavior="equality"
declarativeAgenda="enabled" packages="org.domain.pkg2, org.domain.pkg3" includes="KBase1"
```

```
sequential="false">
    ...
  </kbase>
  ...
</kmodule>
```

以下は、プロジェクトの KIE モジュール記述ファイル (**kmodule.xml**) でサポートされる **kbase** 属性および値です。

表3.1 KIE モジュールでサポートされる KIE ベース属性

属性	サポートされている値	説明
name	すべての名前	**KieContainer** から **KieBase** を取得する名前を定義します。**この属性は必須です。**
includes	KIE モジュールのその他の KIE ベースオブジェクトのコンマ区切り一覧	この KIE ベースに追加するその他の KIE ベースオブジェクトとアーティファクトを定義します。モジュールの **pom.xml** ファイルに依存関係として宣言している場合は、KIE ベースを複数の KIE モジュールに追加できます。
packages	KIE ベースに追加するパッケージのコンマ区切りの一覧 デフォルト値: **all**	(ルールやプロセスなど) この KIE ベースに追加するアーティファクトのパッケージを定義します。デフォルトでは、~/**resources** ディレクトリーのすべてのアーティファクトは KIE ベースに含まれます。この属性は、コンパイルしたアーティファクトの数を制限できます。この属性に指定した一覧に含まれるパッケージだけがコンパイルされます。
default	**true**、**false** デフォルト値: **false**	KIE ベースは、モジュールのデフォルトの KIE ベースで、名前を渡さずに KIE コンテナーから作成できます。各モジュールにはデフォルトの KIE ベースを 1 つだけ指定できます。

属性	サポートされている値	説明
equalsBehavior	**identity**、**equality** デフォルト値: **identity**	新しいファクトが作業メモリーに挿入された場合の Red Hat Process Automation Manager の動作を定義します。**identity** に設定した場合は、同じオブジェクトがワーキングメモリーに存在する場合を除いて、常に新しい **FactHandle** が作成されます。**equality** に設定時には、挿入した **equals()** メソッドに従い、新たに挿入したオブジェクトが既存のファクトと異なる場合に限り新しい **FactHandle** が作成されます。オブジェクトを明示的な ID ではなく、機能の同等性をもとに評価する場合は、**equality** モードを使用します。
eventProcessingMode	**cloud**、**stream** デフォルト値: **cloud**	イベントが KIE ベースで処理される方法を指定します。このプロパティーを **cloud** に設定すると、KIE ベースはイベントを通常のファクトとして扱います。**stream** に設定すると、イベントに対する一時的な推論が可能になります。
declarativeAgenda	**disabled**、**enabled** デフォルト値: **disabled**	宣言型アジェンダが有効かどうかを指定します。
sequential	**true**、**false** デフォルト値: **false**	シーケンシャルモードが有効かどうかを判断します。順次モードでは、デシジョンエンジンは、ワーキングメモリーでの変更に関係なく、デシジョンエンジンアジェンダにリスト化された順番でルールを一度評価します。ステートレスの KIE セッションを使用しており、ルールを実行することで、アジェンダで次に出てくるルールに影響を与えないようにするには、このプロパティーを有効にします。

3.1.3. KIE モジュールでサポートされる KIE セッション属性

KIE セッションがランタイムデータを保存および実行し、プロジェクトの KIE モジュール記述子ファイル (**kmodule.xml**) で KIE セッションを定義した場合は KIE ベース、または KIE コンテナーから直接作成されます。KIE ベースおよび KIE セッションを **kmodule.xml** ファイルに定義すると、特定の属性および値を指定して、KIE セッション設定をさらにカスタマイズできます。

kmodule.xml ファイルの KIE セッション設定例

```
<kmodule>
  ...
  <kbase>
    ...
```

```
    <ksession name="KSession2_1" type="stateless" default="true" clockType="realtime">
    ...
  </kbase>
  ...
</kmodule>
```

以下は、プロジェクトの KIE モジュール記述子ファイル (**kmodule.xml**) でサポートされている **ksession** 属性および値です。

表3.2 KIE モジュールでサポートされる KIE セッション属性

属性	サポートされている値	説明
name	すべての名前	**KieContainer** から **KieSession** を取得する名前を定義します。**この属性は必須です。**
type	**stateful**、**stateless** デフォルト値: **stateful**	KIE セッションの呼び出しと呼び出しの間にデータを保持 (**stateful**) するか、破棄 (**stateless**) するかを指定します。**stateful** に設定したセッションでは、ワーキングメモリーを繰り返し使用できますが、**stateless** に設定したセッションでは、通常、アセットを 1 回だけ実行するために使用されます。**stateless** セッションは、新しいファクトが追加、更新、または削除され、ルールが実行されるたびに変更するナレッジの状態を保存します。**stateless** セッションの実行には、ルール実行など、以前のアクションに関する情報はありません。
default	**true**、**false** デフォルト値: **false**	KIE セッションをモジュールのデフォルトセッションにし、名前を渡さずに KIE コンテナーから作成できるようにするかどうかを指定します。各モジュールにはデフォルトの KIE セッションを 1 つだけ指定できます。
clockType	**realtime**、**pseudo** デフォルト値: **realtime**	イベントのタイムスタンプを、アプリケーションが制御するシステムクロック、または疑似クロックから割り当てられるかどうかを指定します。このクロックは、一時的なルールをテストするユニットで特に便利です。

属性	サポートされている値	説明
beliefSystem	**simple**、**jtms**、**defeasible** デフォルト値: **simple**	KIE セッションが使用する信念体系の種類を定義します。信念体系は、ナレッジ (ファクト) から事実を推測します。たとえば、後ほどデシジョンエンジンから削除される別のファクトに基づいて新しいファクトが挿入されると、この体系では、新たに挿入されたファクトも削除する必要があると判断します。

3.2. RED HAT PROCESS AUTOMATION MANAGER プロジェクトの MAVEN でのパッケージ化およびデプロイ

Business Central 以外の Maven プロジェクトを、設定した KIE Server にデプロイする場合は、プロジェクトの **pom.xml** ファイルを編集して、プロジェクトを KJAR ファイルとしてパッケージ化し、プロジェクトのアセットに対する KIE ベースおよび KIE セッションの設定が含まれる **kmodule.xml** ファイルを追加します。

前提条件

- Red Hat Process Automation Manager ビジネスアセットを含む Maven プロジェクトがある。
- KIE Server がインストールされており、**kie-server** ユーザーアクセスが設定されている。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. 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ー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している Red Hat Process Automation Manager の Maven アーティファクトバージョンです (例: 7.59.0.Final-redhat-00006)。これらの設定は、デプロイメント用に Maven プロジェクトを適切にパッケージ化する必要があります。

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 BOM (bill of materials) 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 Decision Manager と Red Hat Process Automation Manager の両方に適用されます。BOM ファイルを追加すると、提供される Maven リポジトリーから、推移的依存関係の適切なバージョンがプロジェクトに含められます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.12.0.redhat-00008</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) に関する詳細情報は、What is the mapping between Red Hat Process Automation Manager and the Maven library version? を参照してください。

2. 必要に応じて、プロジェクトに Decision Model and Notation (DMN) アセットが含まれる場合は、**pom.xml** ファイルに以下の依存関係を追加して、DMN 実行可能モデルを有効にしてください。DMN 実行可能モデルを使用すると、DMN プロジェクトの DMN デシジョンテーブルロジックをより効果的に評価できるようになります。

```
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-dmn-core</artifactId>
  <scope>provided</scope>
  <version>${rhpam.version}</version>
</dependency>
```

3. Maven プロジェクトの ~/**resources** ディレクトリーに、最低でも以下の内容を含む **META-INF/kmodule.xml** メタデータファイルを作成します。

```
<?xml version="1.0" encoding="UTF-8"?>
<kmodule xmlns="http://www.drools.org/xsd/kmodule">
</kmodule>
```

この **kmodule.xml** ファイルは、すべての Red Hat Process Automation Manager プロジェクトに必要な KIE モジュール記述子です。KIE モジュールを使用して、1 つ以上の KIE ベースを定義し、各 KIE ベースに 1 つ以上の KIE セッションを定義します。

kmodule.xml 設定の詳細は 「KIE モジュール記述子ファイルの設定」 を参照してください。

4. Maven プロジェクトの関連リソースで、**.java** クラスを設定して KIE コンテナーおよび KIE セッションを作成して、KIE ベースを読み込みます。

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

```
public void testApp() {

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

}
```

この例では、KIE コンテナーは、**testApp** プロジェクトのクラスパスからビルドしたファイルを読み込みます。**KieServices** API を使用すれば、KIE ビルド設定およびランタイム設定のすべてにアクセスできます。

プロジェクトの **ReleaseId** を **KieServices** API に渡して KIE コンテナーを作成することもできます。**ReleaseId** は、プロジェクトの **pom.xml** ファイルの **GroupId** 値、**ArtifactId** 値、**Version** 値 (GAV) から生成します。

```
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 void testApp() {

  // Identify the project in the local repository:
  ReleaseId rid = new ReleaseIdImpl("com.sample", "my-app", "1.0.0");

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

}
```

5. 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、以下のコマンドを実行してプロジェクトをビルドします。

```
mvn clean install
```

DMN 実行可能なモデルの場合は、以下のコマンドを実行します。

```
mvn clean install -DgenerateDMNModel=YES
```

ビルドに失敗したら、コマンドラインのエラーメッセージに記載されている問題に対応し、ビルドに成功するまでファイルの妥当性確認を行います。

注記

デフォルトで Maven 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ない場合は、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ことを確認して、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の詳細は、「実行可能ルールモデル」 を参照してください。

6. プロジェクトをローカルで正常にビルドしてテストした後に、プロジェクトをリモートの Maven リポジトリーにデプロイします。

```
mvn deploy
```

3.3. RED HAT PROCESS AUTOMATION MANAGER プロジェクトの JAVA アプリケーションでのパッケージ化およびデプロイ

お使いの Java アプリケーションから、設定した KIE Server にプロジェクトをデプロイする場合は、**KieModuleModel** インスタンスを使用して **kmodule.xml** ファイルをプログラムで作成して KIE ベースおよび KIE セッションを定義し、プロジェクトのすべてのリソースを、KIE 仮想ファイルシステム **KieFileSystem** に追加します。

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。
- KIE Server がインストールされており、**kie-server** ユーザーアクセスが設定されている。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. 必要に応じて、プロジェクトに Decision Model and Notation (DMN) アセットが含まれる場合は、Java プロジェクトの該当のクラスパスに以下の依存関係を追加して、DMN 実行可能モデルを有効にしてください。DMN 実行可能モデルを使用すると、DMN プロジェクトの DMN デシジョンテーブルロジックをより効果的に評価できるようになります。

```
<dependency>
  <groupId>org.kie</groupId>
  <artifactId>kie-dmn-core</artifactId>
```

```
  <scope>provided</scope>
  <version>${rhpam.version}</version>
</dependency>
```

<version> は、プロジェクトで現在使用している Red Hat Process Automation Manager の Maven アーティファクトバージョンです (例: 7.59.0.Final-redhat-00006)。

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 BOM (bill of materials) 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 Decision Manager と Red Hat Process Automation Manager の両方に適用されます。BOM ファイルを追加すると、提供される Maven リポジトリーから、推移的依存関係の適切なバージョンがプロジェクトに含められます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.12.0.redhat-00008</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

Red Hat Business Automation BOM (Bill of Materials) に関する詳細情報は、What is the mapping between Red Hat Process Automation Manager and the Maven library version? を参照してください。

2. **KieServices** API を使用して、必要な KIE ベースおよび KIE セッションを持つ **KieModuleModel** インスタンスを作成します。**KieServices** API を使用すれば、KIE ビルド設定およびランタイム設定のすべてにアクセスできます。**KieModuleModel** インスタンスは、プロジェクトの **kmodule.xml** ファイルを生成します。
kmodule.xml 設定の詳細は 「KIE モジュール記述子ファイルの設定」 を参照してください。

3. **KieModuleModel** インスタンスを XML に変換し、XML を **KieFileSystem** に追加します。

プログラムを使用して kmodule.xml を作成し、 KieFileSystem に追加

```
import org.kie.api.KieServices;
import org.kie.api.builder.model.KieModuleModel;
import org.kie.api.builder.model.KieBaseModel;
import org.kie.api.builder.model.KieSessionModel;
import org.kie.api.builder.KieFileSystem;

 KieServices kieServices = KieServices.Factory.get();
 KieModuleModel kieModuleModel = kieServices.newKieModuleModel();

 KieBaseModel kieBaseModel1 = kieModuleModel.newKieBaseModel("KBase1")
  .setDefault(true)
  .setEqualsBehavior(EqualityBehaviorOption.EQUALITY)
  .setEventProcessingMode(EventProcessingOption.STREAM);
```

```
KieSessionModel ksessionModel1 = kieBaseModel1.newKieSessionModel("KSession1")
  .setDefault(true)
  .setType(KieSessionModel.KieSessionType.STATEFUL)
  .setClockType(ClockTypeOption.get("realtime"));

KieFileSystem kfs = kieServices.newKieFileSystem();
kfs.writeKModuleXML(kieModuleModel.toXML());
```

4. プロジェクトで使用する残りの Red Hat Process Automation Manager アセットをすべて **KieFileSystem** インスタンスに追加します。アーティファクトは、Maven プロジェクトファイル構造に含まれる必要があります。

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

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

この例では、プロジェクトアセットは、**String** 変数および **Resource** インスタンスの両方として追加されます。**Resource** インスタンスは **KieResources** ファクトリーを使用して作成され、**KieServices** インスタンスにより提供されます。**KieResources** クラスは、**InputStream** オブジェクト、**URL** オブジェクト、および **File** オブジェクト、またはファイルシステムのパスを示す **String** を、**KieFileSystem** が管理する **Resource** インスタンスに変換する factory メソッドを提供します。

プロジェクトのアーティファクトを **KieFileSystem** に追加する際に、**ResourceType** プロパティーを **Resource** オブジェクトに明示的に割り当てることもできます。

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

KieFileSystem kfs = ...
kfs.write("src/main/resources/myDrl.txt",
  kieServices.getResources().newInputStreamResource(drlStream)
    .setResourceType(ResourceType.DRL));
```

5. **buildAll()** メソッドと **KieBuilder** を併用して **KieFileSystem** のコンテンツをビルドし、KIE コンテナーを作成してデプロイします。

```
import org.kie.api.KieServices;
import org.kie.api.KieServices.Factory;
import org.kie.api.builder.KieFileSystem;
import org.kie.api.builder.KieBuilder;
import org.kie.api.runtime.KieContainer;

KieServices kieServices = KieServices.Factory.get();
KieFileSystem kfs = ...

KieBuilder kieBuilder = ks.newKieBuilder( kfs );
kieBuilder.buildAll()
assertEquals(0, kieBuilder.getResults().getMessages(Message.Level.ERROR).size());

KieContainer kieContainer = kieServices
  .newKieContainer(kieServices.getRepository().getDefaultReleaseId());
```

ERROR ビルドは、プロジェクトのコンパイルに失敗し、**KieModule** が作成されず、**KieRepository** シングルトンに何も追加されないことを示しています。**WARNING** または **INFO** の結果は、プロジェクトのコンパイルが成功したことと、ビルドプロセスの詳細を示しています。

注記

実行可能なルールモデルから Java アプリケーションプロジェクトでルールアセットをビルドするには、プロジェクトの **pom.xml** ファイルに以下の依存関係があることを確認します。

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

この依存関係は、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。

依存関係を確認したあとに、以下のように変更した **buildAll()** オプションを使用して実行可能なモデルを有効にします。

```
kieBuilder.buildAll(ExecutableModelProject.class)
```

実行可能なルールモデルの詳細は、「実行可能ルールモデル」 を参照してください。

3.4. 実行可能ルールモデル

Red Hat Process Automation Manager のルールアセットは、標準の **kie-maven-plugin** を使用して、デフォルトで実行可能なルールモデルからビルドされます。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 ベースの表記を提供します。実行可能モデルは、以前の Red Hat Process Automation Manager バージョンの標準アセットパッケージングの代わるもので、より効率的です。KIE コンテナーと KIE ベースの作成をより迅速に実行でき、DRL (Drools Rule Language) ファイルや他の Red Hat Process Automation Manager アセットの一覧のサイズが大きい場合にとくに効果的です。

kie-maven-plugin プラグインを使用しない場合や、必要な **drools-model-compiler** 依存関係がプロジェクトにない場合、ルールアセットは実行可能モデルなしでビルドされます。したがって、ビルド時に実行可能なモデルを生成するには、プロジェクトの **pom.xml** ファイルに **kie-maven-plugin** プラグインおよび **drools-model-compiler** 依存関係が追加されていることを確認します。

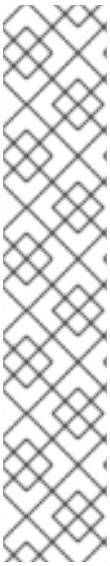
実行可能なルールモデルでは、プロジェクトにとって具体的に以下のような利点があります。

- **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、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化して、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、はるかに速い方法で KIE コンテナーと KIE ベース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** プラグイン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ため、**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で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 24 個以上使用しているため、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);
```

3.4.1. Red Hat Process Automation Manager プロジェクトでの実行可能ルールモデルの変更または無効化

Red Hat Process Automation Manager のルールアセットは、標準の **kie-maven-plugin** を使用して、デフォルトで実行可能なルールモデルからビルドされます。実行可能モデルは、Red Hat Process Automation Manager のこれまでのバージョンで使用した標準アセットパッケージよりもより効率的です。ただし、必要に応じて、実行可能ルールモデルを変更または無効にして、デフォルトのモデルベースの KJAR ではなく、DRL ベースの KJAR として Red Hat Process Automation Manager プロジェクトをビルドできます。

手順

通常の方法で Red Hat Process Automation Manager プロジェクトをビルドします。ただし、プロジェクトの種類に合わせて、別のビルドオプションを用意してください。

- Maven プロジェクトの場合は、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ます。

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

<VALUE> は、3 つの値のいずれかに置き換えます。

- **YES_WITHDRL**: (デフォルト) 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)。
- **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。
- **NO**: 実行可能なモデルは生成されません。

デフォルトの実行可能モデルの動作を無効にするビルドコマンドの例:

```
mvn clean install -DgenerateModel=NO
```

- プログラムで設定した Java アプリケーションの場合、実行可能なモデルはデフォルトで無効になっています。KIE 仮想ファイルシステム **KieFileSystem** にルールアセットを追加して、以下の **buildAll()** メソッドのいずれかと **KieBuilder** をあわせて使用してください。
  - **buildAll()** (デフォルト) または **buildAll(DrlProject.class)**: 実行可能なモデルは生成されません。
  - **buildAll(ExecutableModelProject.class)**: 元のプロジェクトにある DRL ファイルに対応する実行可能モデルが生成されます。

実行可能モデルの動作を有効にするコードの例:

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

3.5. KIE スキャナーを使用した KIE コンテナーの監視および更新

Red Hat Process Automation Manager の KIE スキャナーは、Maven リポジトリーで Red Hat Process Automation Manager プロジェクトの新しい **SNAPSHOT** バージョンを監視してから、プロジェクトの最新バージョンを指定された KIE コンテナーにデプロイします。開発環境に KIE スキャナーを使用して、新規バージョンが利用できるようになった時に、Red Hat Process Automation Manager プロジェクトのデプロイメントをより効率的に管理できます。

重要

実稼働環境では、誤ってまたは予期せずにプロジェクトが更新されてしまわないように、**SNAPSHOT** のプロジェクトバージョンで、KIE スキャナーを使用しないでください。KIE スキャナーは、**SNAPSHOT** プロジェクトバージョンを使用する開発環境向けに設計されています。

ビジネスプロセスと KIE スキャナーを併用しないようにしてください。プロセスで KIE スキャナーを使用すると、予期せぬ更新が発生し、プロセスインスタンスの実行と互換性のない変更が加えられた場合に、長時間実行中のプロセスでエラーが発生する可能性があります。

前提条件

- **kie-ci.jar** ファイルを Red Hat Process Automation Manager プロジェクトのクラスパスに用意しておく。

手順

1. プロジェクト内の、該当する **.java** クラスで、以下のコード例のように、KIE スキャナーを登録して起動します。

KIE コンテナー向けの KIE スキャナーの登録および起動

```
import org.kie.api.KieServices;
import org.kie.api.builder.ReleaseId;
import org.kie.api.runtime.KieContainer;
import org.kie.api.builder.KieScanner;

...

KieServices kieServices = KieServices.Factory.get();
ReleaseId releaseId = kieServices
  .newReleaseId("com.sample", "my-app", "1.0-SNAPSHOT");
KieContainer kContainer = kieServices.newKieContainer(releaseId);
KieScanner kScanner = kieServices.newKieScanner(kContainer);

// Start KIE scanner for polling the Maven repository every 10 seconds (10000 ms)
kScanner.start(10000L);
```

この例では、KIE スキャナーは一定の間隔で実行されるように設定しています。KIE スキャナーの最小のポーリング間隔は 1 ミリ秒 (ms) で、最大のポーリング間隔はデータ型 **long** の最大値です。ポーリングの間隔が 0 以下の場合は、**java.lang.IllegalArgumentException: pollingInterval must be positive** エラーが発生します。また、KIE スキャナーを **scanNow()** メソッドで呼び出してオンデマンドで実行するように設定することも可能です。

この例のプロジェクトグループ ID、アーティファクト ID、およびバージョン (GAV) の設定は、**com.sample:my-app:1.0-SNAPSHOT** で設定されています。プロジェクトバージョンには、**-SNAPSHOT** の接尾辞を含めて、KIE スキャナーが指定のアーティファクトバージョンの最新ビルドを取得できるようにする必要があります。**1.0.1-SNAPSHOT** への更新など、スナップショットのプロジェクトバージョン番号を変更する場合は、KIE スキャナー設定の GAV 定義のバージョンも更新する必要があります。KIE スキャナーは、**com.sample:my-app:1.0** など、静的バージョンのプロジェクトの更新を取得しません。

2. Maven リポジトリーの **settings.xml** ファイルで、**updatePolicy** 設定を **always** に指定し、KIE スキャナーが正しく機能するようにします。

```
<profile>
  <id>guvnor-m2-repo</id>
  <repositories>
    <repository>
      <id>guvnor-m2-repo</id>
      <name>BA Repository</name>
      <url>http://localhost:8080/business-central/maven2/</url>
      <layout>default</layout>
      <releases>
        <enabled>true</enabled>
        <updatePolicy>always</updatePolicy>
      </releases>
      <snapshots>
        <enabled>true</enabled>
        <updatePolicy>always</updatePolicy>
      </snapshots>
    </repository>
  </repositories>
</profile>
```

KIE スキャナーがポーリングを開始した後に、KIE スキャナーが指定の KIE コンテナーで **SNAPSHOT** プロジェクトの更新バージョンを検出した場合に、KIE スキャナーは自動的に新しいプロジェクトバージョンをダウンロードして、新規プロジェクトのインクリメンタルビルドをトリガーします。この時点以降、KIE コンテナーから作成された新規の **KieBase** オブジェクトおよび **KieSession** オブジェクトで、新規プロジェクトバージョンが使用されるようになります。

KIE Server API を使用して KIE スキャナーを開始または停止する方法は、**KIE API を使用した Red Hat Process Automation Manager の操作** を参照してください。

3.6. KIE SERVER でのサービスの起動

Business Central 以外の Maven または Java プロジェクトから Red Hat Process Automation Manager アセットをデプロイした場合は、KIE Server REST API コールを使用して、KIE コンテナー (デプロイメントユニット) およびそのサービスを起動できます。KIE Server REST API を使用して、デプロイメントの種類 (Business Central からのデプロイメントを含む) にかかわらずサービスを起動できますが、Business Central からデプロイしたプロジェクトは自動的に起動するか、Business Central インターフェイス内で起動できます。

前提条件

- KIE Server がインストールされており、**kie-server** ユーザーアクセスが設定されている。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

コマンドターミナルで以下の API 要求を実行し、KIE Server の KIE コンテナーにサービスを読み込んで起動します。

```
$ curl --user "<username>:<password>" -H "Content-Type: application/json" -X PUT -d '{"container-id" : "<containerID>","release-id" : {"group-id" : "<groupID>","artifact-id" : "<artifactID>","version" : "
```

```
<version>"}}' http://<serverhost>:<serverport>/kie-
server/services/rest/server/containers/<containerID>
```

以下の値を置き換えます。

- **<username>、<password>: kie-server** ロールを持つユーザーのユーザー名およびパスワード。
- **<containerID>:** KIE コンテナー (デプロイメントユニット) の識別子。ランダムの識別子を使用することもできますが、コマンドの URL およびデータの両方で同じものを使用する必要があります。
- **<groupID>、<artifactID>、<version>:** プロジェクトの GAV 値。
- **<serverhost>:** KIE Server のホスト名 (KIE Server と同じホストでコマンドを実行する場合は **localhost**)。
- **<serverport>:** KIE Server のポート番号。

以下に例を示します。

```
curl --user "rhpamAdmin:password@1" -H "Content-Type: application/json" -X PUT -d '{"container-id"
: "kie1","release-id" : {"group-id" : "org.kie.server.testing","artifact-id" : "container-crud-tests1","version"
: "2.1.0.GA"}}' http://localhost:39043/kie-server/services/rest/server/containers/kie1
```

3.7. KIE SERVER でのサービスの停止および削除

Business Central 以外の Maven または Java プロジェクトから Red Hat Process Automation Manager サービスを起動した場合は、KIE Server REST API コールを使用して、サービスを含む KIE コンテナー (デプロイメントユニット) を停止して削除できます。KIE Server REST API を使用して、デプロイメントの種類 (Business Central からのデプロイメントを含む) にかかわらずサービスを停止できますが、Business Central からのサービスは Business Central インターフェイス内で停止できます。

前提条件

- KIE Server がインストールされており、**kie-server** ユーザーアクセスが設定されている。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

コマンドターミナルで、以下の API 要求を実行して、KIE Server のサービスで KIE コンテナーを停止および削除します。

```
$ curl --user "<username>:<password>" -X DELETE http://<serverhost>:<serverport>/kie-
server/services/rest/server/containers/<containerID>
```

以下の値を置き換えます。

- **<username>、<password>: kie-server** ロールを持つユーザーのユーザー名およびパスワード。
- **<containerID>:** KIE コンテナー (デプロイメントユニット) の識別子。ランダムの識別子を使用することもできますが、コマンドの URL およびデータの両方で同じものを使用する必要があります。

- **<serverhost>:** KIE Server のホスト名 (KIE Server と同じホストでコマンドを実行する場合は **localhost**)。
- **<serverport>:** KIE Server のポート番号。

以下に例を示します。

```
curl --user "rhpamAdmin:password@1" -X DELETE http://localhost:39043/kie-server/services/rest/server/containers/kie1
```

第4章 関連情報

- **DRL ルールを使用した意思決定サービスの設計** の「ルールの実行」
- **KIE API を使った Red Hat Process Automation Manager の操作**
- **Operator を使用した Red Hat OpenShift Container Platform 4 への Red Hat Process Automation Manager 環境のデプロイメント**
- **テンプレートを使用した Red Hat OpenShift Container Platform 3 への Red Hat Process Automation Manager 環境のデプロイメント**

パート II. BUSINESS CENTRAL でのプロジェクトの管理

プロセス管理者は、Red Hat Process Automation Manager の Business Central を使用して、1 つまたは複数のブランチで新しいプロジェクト、サンプルプロジェクト、およびインポートしたプロジェクトを管理できます。

前提条件

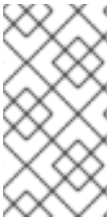
- Red Hat JBoss Enterprise Application Platform 7.4 がインストールされている。詳細情報は **Red Hat JBoss EAP 7.4 インストールガイド**を参照してください。
- Red Hat Process Automation Manager がインストールされ、KIE Server で設定されている。詳細は **Red Hat JBoss EAP 7.4 への Red Hat Process Automation Manager のインストールおよび設定** を参照してください。
- Red Hat Process Automation Manager が稼働し、**developer** ロールで Business Central にログインできる。詳細は、**Red Hat Process Automation Manager インストールの計画**を参照してください。

第5章 RED HAT PROCESS AUTOMATION MANAGER プロジェクト

Red Hat Process Automation Manager プロジェクトには、Red Hat Process Automation Manager で開発し、スペースに割り当てられるビジネスアセットが含まれています (例: **MySpace** の **MyProject**)。プロジェクトには、Maven プロジェクトのオブジェクトモデルファイル (**pom.xml**) などの設定ファイルも含まれますが、設定ファイルには、ビルド、環境、その他のプロジェクト関連情報、KIE モジュール記述子ファイル (**kmodule.xml**) が含まれます。KIE モジュール記述子ファイルには、プロジェクトのアセットに関する KIE ベースおよび KIE セッションの設定が含まれます。

第6章 新しいプロセスデザイナーへのビジネスプロセスの移行

Business Central のレガシーのプロセスデザイナーは、Red Hat Process Automation Manager 7.12.0 で非推奨になりました。このツールは、今後の Red Hat Process Automation Manager リリースで削除予定です。そのため、レガシーのプロセスデザイナーには新しい機能拡張や機能は追加されません。新しいプロセスデザイナーを使用する場合は、お使いのプロセスを新しいデザイナーに移行し始めます。新しいプロセスデザイナーですべての新規プロセスを作成します。



注記

プロセスエンジンは、今後も継続して KIE Server のレガシーデザイナーで生成されたビジネスプロセスの実行やデプロイメントをサポートします。レガシーのビジネスプロセスが機能しており、交換する予定がない場合は、現時点では新しいデザイナーへの移行は必須ではありません。

新規デザイナーでサポートされているビジネスプロセスが含まれるビジネスプロセスのみを移行できます。Red Hat Process Automation Manager の今後のバージョンに、さらにノードが追加される予定です。

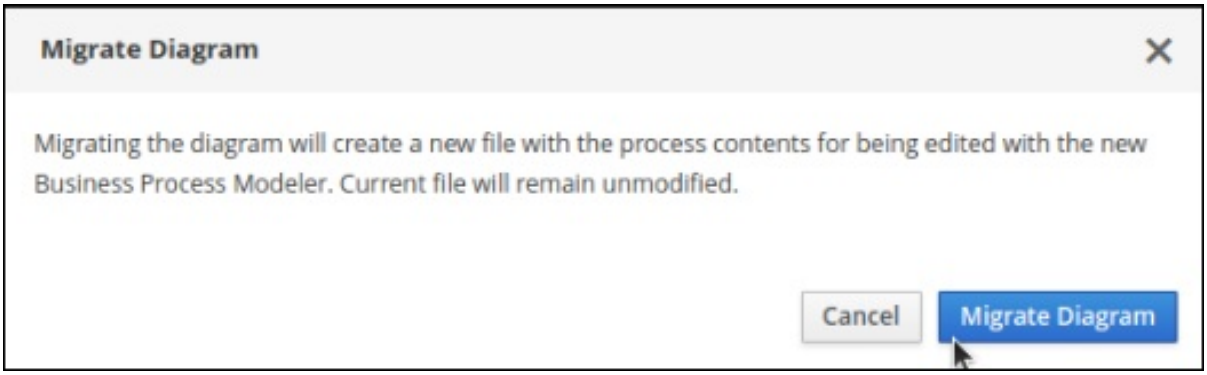
前提条件

- レガシーのプロセスデザイナーで作成したビジネスプロセスアセットを含む既存のプロジェクトがある。

手順

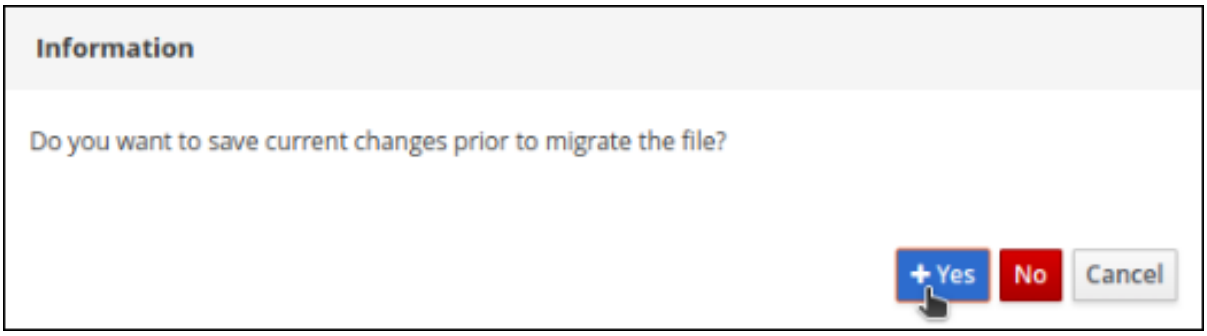
1. Business Central で **Menu → Design → Projects** の順にクリックします。
2. 移行するプロジェクト (例: **Mortgage_Process**) をクリックします。
3. **OK** をクリックしてプロジェクトのアセット一覧を開きます。
4. プロジェクトの **Business Process** アセットをクリックして、レガシーのプロセスデザイナーでそのアセットを開きます。
5. **Migrate → Migrate Diagram** をクリックします。

   図6.1 移行の確認メッセージ



6. **Yes** または **No** を選択して、変更を確定します。このオプションは、レガシーのプロセスデザイナーに変更を加えた場合のみ利用できます。

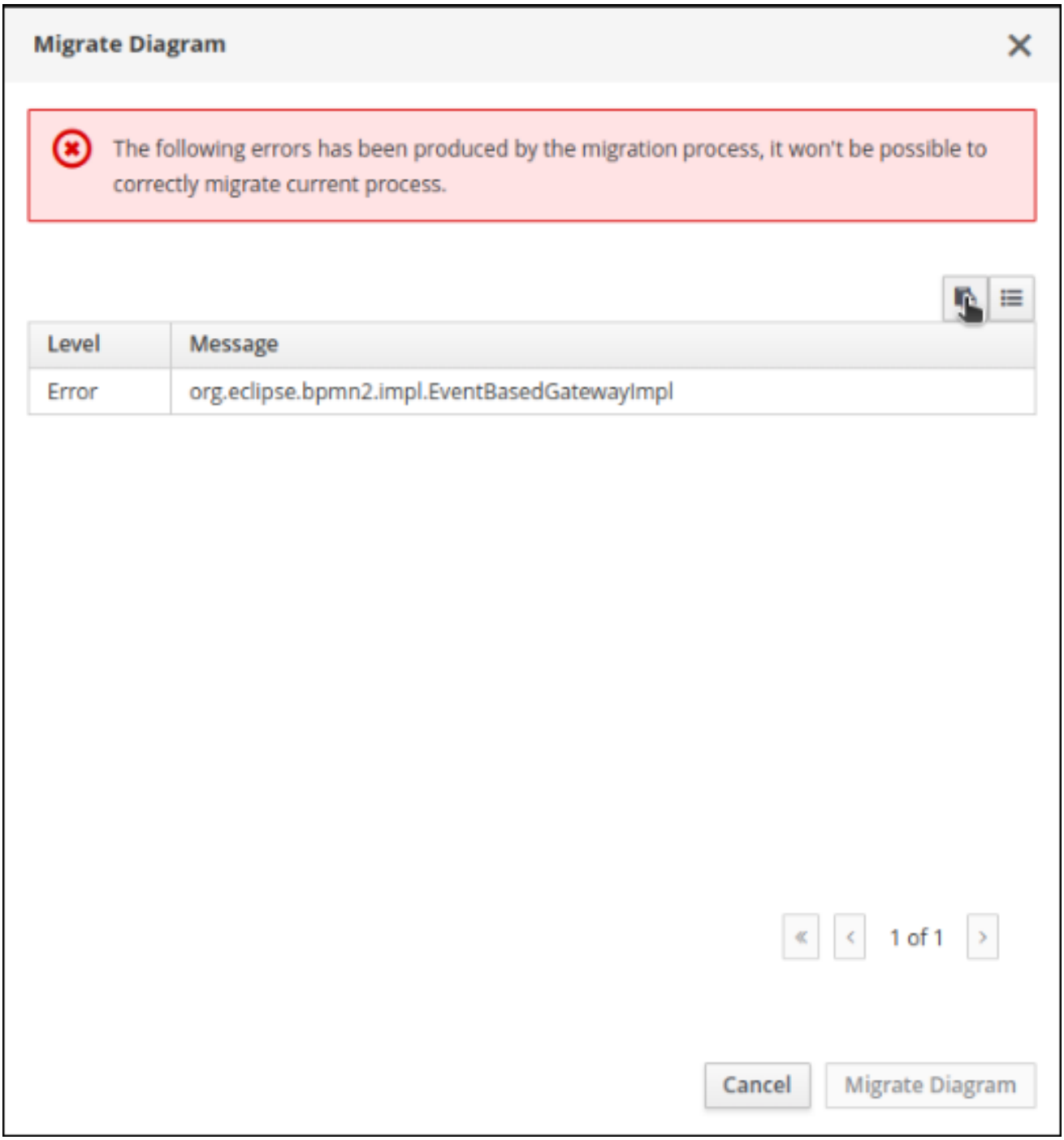
図6.2 ダイアグラム変更の保存確認



移行に成功したら、ビジネスプロセスが新規のビジネスプロセスで開き、ビジネスプロセス名の拡張子が *.bpmn2 から *.bpmn に変わります。

ノードタイプがサポートされていないことが原因で移行に失敗したら、Business Central で以下のエラーメッセージが表示されます。

図6.3 移行失敗のメッセージ



第7章 BUSINESS CENTRAL の既存プロジェクトの変更

Business Central には、製品と機能に慣れるのに使用できるサンプルプロジェクトが多数あります。サンプルプロジェクトは、さまざまなビジネスシナリオを紹介するために設計され、作成されました。サンプルプロジェクトを変更し、ビジネス固有のニーズを満たすことができます。たとえば、Red Hat Process Automation Manager 7.12 には **Mortgage_Process** のサンプルプロジェクトが含まれており、このプロジェクトは、事前定義済みのデータオブジェクト、ガイド付きデシジョンテーブル、ガイド付きのルール、フォーム、およびビジネスプロセスで設定されます。このサンプルを編集して、ビジネスプロセスを改良できます。

要件を満たす既存の Business Central のプロジェクトサンプルがない場合は、新規プロジェクトを作成するか、git リポジトリーからプロジェクトをインポートできます。詳細は、*9章Git リポジトリーからのプロジェクトのインポート* を参照してください。git から他のプロジェクトをインポートできます。たとえば、別の Business Central インスタンスで開発したプロジェクトなどがあります。

第8章 MORTGAGE-PROCESS プロジェクトの作成

プロジェクトは、データオブジェクト、ビジネスプロセス、ガイド付きルール、デシジョンテーブル、フォームなど、アセットのコンテナーです。作成するプロジェクトは、Business Central に既存の **Mortgage_Process** サンプルプロジェクトによく似ています。

手順

1. Business Central で、**Menu → Design → Projects** に移動します。
   Red Hat Process Automation Manager は以下のイメージのように **MySpace** と呼ばれるデフォルトスペースを提供します。このデフォルトスペースを使用してサンプルプロジェクトを作成およびテストできます。

   図8.1 デフォルトのスペース

2. **Add Project** をクリックします。
3. **Name** フィールドに **mortgage-process** と入力します。
4. **Configure Advanced Options** をクリックして GAV フィールドを以下の値に変更します。
   - Group ID: **com.myspace**
   - Artifact ID: **mortgage-process**
   - Version: **1.0.0**
5. **Add** をクリックします。

プロジェクトの **Assets** ビューを開きます。

8.1. MORTGAGE_PROCESS サンプルプロジェクトの変更

Mortgage_Process サンプルプロジェクトは、事前定義済みのデータオブジェクト、ガイド付きデシジョンテーブル、ガイド付きルール、フォーム、およびビジネスプロセスで設定されています。サンプルプロジェクトを利用すれば、Red Hat Process Automation Manager の使い方を簡単に覚えることができます。実際のビジネスシナリオでは、ビジネス要件に適したデータを使用してすべてのアセットを作成します。

Mortgage_Process のサンプルプロジェクトに移動し、事前定義済みアセットを表示します。

手順

1. Business Central で、**Menu → Design → Projects** に移動します。
2. 画面の右上隅にある **Try Samples** の横にある矢印をクリックし、**Try Samples** を選択します。
3. **Mortgage_Process** を選択し、**OK** をクリックします。プロジェクトの **Assets** ビューを開きます。
4. 変更するアセットをクリックします。アセットは、プロジェクト要件に合わせて、すべて編集できます。

8.2. アーキタイプを使用したプロジェクトの作成

アーキタイプとは、Apache Maven リポジトリーにインストールされ、特定のテンプレート構造が含まれるプロジェクトのことです。アーキタイプを使用して、プロジェクトテンプレートのパラメーター化されたバージョンを生成することも可能です。アーキタイプを使用してプロジェクトを作成すると、Red Hat Process Automation Manager インストールに接続されている Git リポジトリーに追加されます。

前提条件

- Business Central の **Settings** でアーキタイプを作成して、**Archetypes** ページに追加しました。アーキタイプの作成に関する詳細は、Guide to Creating Archetypes を参照してください。
- Business Central のスペースでアーキタイプをデフォルトとして設定しました。

アーキタイプに管理に関する詳細は、**Business Central の設定およびプロパティーの設定** を参照してください。

手順

1. Business Central で、**Menu → Design → Projects** に移動します。
2. アーキタイプテンプレートから新しいプロジェクトを追加するスペースを選択または作成します。
3. **Add Project** をクリックします。
4. **Name** フィールドおよび **Description** フィールドに、プロジェクト名と説明を入力します。
5. **Configure Advanced Options** をクリックします。
6. **Based on template** チェックボックスを選択します。
7. 必要に応じてドロップダウンオプションからアーキタイプを選択します。スペースで設定済みのデフォルトのアーキタイプが選択されています。
8. **Add** をクリックします。

選択したアーキタイプのテンプレートに基づいて、プロジェクトのアセットビューが表示されます。

第9章 GIT リポジトリーからのプロジェクトのインポート

Git は分散バージョン管理システムです。リビジョンをコミットオブジェクトとして実装します。リポジトリーに変更を保存すると、Git リポジトリーに新しいコミットオブジェクトが作成されます。

Business Central は Git を使用してプロジェクトデータ (ルールやプロセスなどのアセットを含む) を格納します。Business Central でプロジェクトを作成すると、Business Central に接続される Git リポジトリーに追加されます。Git リポジトリーにプロジェクトがある場合は、プロジェクトの master ブランチをインポートするか、Business Central スペースを使用して 、他の特定のブランチと master ブランチをあわせて Business Central Git リポジトリーにインポートできます。

前提条件

- Red Hat Process Automation Manager プロジェクトが外部の Git リポジトリーに存在している。
- 外部の Git リポジトリーへの読み取りアクセスに必要な認証情報がある。

手順

1. Business Central で、**Menu → Design → Projects** に移動します。
2. プロジェクトをインポートするスペースを選択または作成します。デフォルトのスペースは **MySpace** です。
3. 画面の右上隅にある **Add Project** の横にある矢印をクリックし、**Import Project** を選択します。
4. **Import Project** ウィンドウに、インポートするプロジェクトが含まれる Git リポジトリーの URL および認証情報を入力し、**Import** をクリックします。**Import Projects** ページが表示されます。
5. オプション: master と固有のブランチをインポートするには、以下のタスクを実行します。
   a. **Import Projects** ページで、ブランチ  アイコンをクリックします。
   b. **Branches to be imported** ウィンドウで、一覧からブランチを選択します。

      注記

      最低でも master ブランチを選択する必要があります。

   c. **OK** をクリックします。
6. **Import Projects** ページで、プロジェクトページがハイライトされていることを確認し、**OK** をクリックします。

第10章 プロジェクトバージョンの改訂

プロジェクトの新規インスタンスをビルドおよびデプロイする前に、Red Hat Process Automation Manager のプロジェクトバージョン番号を改訂できます。プロジェクトの新規バージョンを作成すると、以前のバージョンを保存して、新規バージョンで問題が発生した場合に、以前のバージョンに戻すことができます。

前提条件

- KIE Server がデプロイされて Business Central に接続されている。

手順

1. Business Central で、**Menu → Design → Projects** に移動します。
2. デプロイするプロジェクト (例: **Mortgage_Process**) をクリックします。
3. **デプロイ** をクリックします。
   - プロジェクト名にコンテナーがない場合は、デフォルト値でコンテナーが自動的に作成されます。
   - 以前のバージョンのプロジェクトがすでにデプロイされている場合は、プロジェクト設定に移動して、プロジェクトバージョンを変更します。終了したら、変更を保存して **Deploy** をクリックします。これにより、最新の変更が適用された同じプロジェクトの新しいバージョンが、古いバージョンとともにデプロイされます。

注記

Build & Installオプションを選択してプロジェクトをビルドし、KJAR ファイルを KIE Server にデプロイせずに設定済みの Maven リポジトリーに公開することもできます。開発環境では、**Deploy** をクリックすると、ビルドされた KJAR ファイルを KIE Server に、実行中のインスタンス (がある場合はそれ) を停止せずにデプロイできます。または **Redeploy** をクリックして、ビルドされた KJAR ファイルをデプロイしてすべてのインスタンスを置き換えることもできます。次回、ビルドされた KJAR ファイルをデプロイまたは再デプロイすると、以前のデプロイメントユニット (KIE コンテナー) が同じターゲット KIE Server で自動的に更新されます。実稼働環境では **Redeploy** オプションは無効になっており、**Deploy** をクリックして、ビルドされた KJAR ファイルを KIE Server 上の新規デプロイメントユニット (KIE コンテナー) にデプロイすることのみが可能です。

KIE Server の環境モードを設定するには、**org.kie.server.mode** システムプロパティーを **org.kie.server.mode=development** または **org.kie.server.mode=production** に設定します。Business Central の対応するプロジェクトでのデプロイメント動作を設定するには、プロジェクトの **Settings → General Settings → Version** に移動し、**Development Mode** オプションを選択します。デフォルトでは、KIE Server および Business Central のすべての新規プロジェクトは開発モードになっています。**Development Mode** をオンにしたプロジェクトをデプロイしたり、実稼働モードになっている KIE Server に手動で **SNAPSHOT** バージョンの接尾辞を追加したプロジェクトをデプロイしたりすることはできません。

4. プロジェクトのデプロイメントに関する詳細を確認するには、画面の上部にあるデプロイメントバナーの **View deployment details** か、**Deploy** のドロップダウンメニューをクリックしま

す。このオプションを使用すると、**Menu → Deploy → Execution Servers** ページに移動します。

5. プロセス定義を確認するには、**Menu → Manage → Process Definitions** をクリックし、  をクリックします。

6. **Actions** 列の  をクリックして、**Start** を選択し、プロセスの新規インスタンスを起動します。

第11章 プロジェクトの設定

Red Hat Process Automation Manager 7.12 以降 の Business Central では、さらなるプロジェクト設定カテゴリーが新規プロセスデザイナーに追加されています。

前提条件

- Business Central プロジェクトが作成されている。

手順

1. プロジェクトの **Settings** タブにアクセスするには、Business Central で **Menu → Design → Projects** に移動します。
2. プロジェクト名をクリックします。
3. **Settings** をクリックして、以下のプロジェクト設定を表示または変更します。
   - **General Settings**: ユーザーが、属性 **Name**、**Description**、**Group ID**、**Artifact ID**、**Version** (GAV) および **Development Mode** を設定できるようにします。この設定には以下のオプションも含まれます。
     - **URL**: プロジェクトのクローン作成用に git リポジトリーとして読み取り専用の URL を指定するのに使用します。
     - **Disable GAV conflict check** GAV 競合チェックを有効化するか、無効化するかを指定します。この機能を無効にすると、同じ GAV 値を複数のプロジェクトに指定できます。
     - **Allow child GAV edition**: サブプロジェクトに GAV エディションを設定できます。
   - **Dependencies**: これを使用して、**Group ID**、**Artifact ID**、および **Version** を入力するか、Business Central のリポジトリープロジェクトから、手動で依存関係を追加します。依存関係ごとに、**Package white list**オプションで **All** または **None** を選択します。
   - **KIE Bases**: 以前は **ナレッジベース** と呼ばれていましたものの新しい名前。デフォルトとして KIE ベースを指定する必要があります。以下の詳細を指定して KIE ベースを追加します。
     - **Name**
     - **Included KIE bases**
     - **Package**
     - **Equal Behavior**: **Identity** または **Equality**
     - **Event Processing Model**: **Stream** または **Cloud**
     - **KIE sessions**
   - **External Data Objects**: ルール作成者が必要とする可能性のあるプロジェクトまたはプロジェクトの依存関係内では、データオブジェクトは明示的に定義されません。外部のデータオブジェクトは、通常 **java.util.List** など Java ランタイムで指定されます。

- **Validation**: 新規プロジェクトまたはモジュールを作成するとき、または Maven リポジトリーにプロジェクトをインストールまたはデプロイするときに、プロジェクトの GAV の一意性を確認するのに使用する Maven リポジトリー。
- **Service Tasks**: 以下のサービスタスクをプロジェクトに追加できます。
  - **BusinessRuleTask**: ビジネスルールタスクを実行します。
  - **Decision Task**: DMN デシジョンタスクを実行します。
  - **Email**: メールを送信します。
  - **JMSSendTask**: JMS メッセージを送信します。
  - **Rest**: REST 呼び出しを実行します。
  - **ServiceTask**: サービスタスクを実行します。
  - **WebService**: Web サービスの呼び出しを実行します。
- **Deployments**: デプロイメントは以下のカテゴリーに分類されます。
  - **General Settings**: **Runtime Strategy**、**Persistence Unit Name**、**Persistence Mode**、**Audit Persistence Unit Name**、および **Audit Mode**
  - **Marshalling strategies**
  - **Global**
  - **Event listeners**
  - **Required roles**
  - **Remoteable classes**
  - **Task event listeners**
  - **Configuration**
  - **Environment entries**
  - **Work item handlers**
- **Persistence**: 永続化は以下のカテゴリーに分類されます。
  - **Persistence Unit**
  - **Persistence Provider**
  - **Data Source**
  - **Properties**: 以下のプロパティーの値を設定する場合や、新規プロパティーを作成する場合に使用します。
    - **hibernate.dialect**
    - **hibernate.max_fetch_depth**
    - **hibernate.hbm2ddl.auto**

        - hibernate.show_sql
        - hibernate.id.new_generator_mappings
        - hibernate.transaction.jta.platform
    - Project Persistable Data Objects

- **Branch Management**: ブランチ名と割り当てられたユーザーロールをもとにブランチのロールアクセスを指定します。

4. **Save** をクリックします。

第12章 BUSINESS CENTRAL での複数のブランチ

Business Central では複数のブランチの使用をサポートしており、すべてのアセットを含む、既存のブランチをもとに新規ブランチを作成できます。新規プロジェクト、インポートされたプロジェクト、サンプルプロジェクトはすべてデフォルトの **master** ブランチで開きます。必要な数だけブランチを作成し、**master** ブランチのオリジナルのプロジェクトに影響を与えずに、複数のブランチ間を切り替えて作業することができます。

Red Hat Process Automation Manager 7.12 には、ブランチの永続化サポートがあります。このサポートにより、Business Central が最後に使用したブランチを記憶し、ログインしなおしたときにそのブランチが開くようになります。

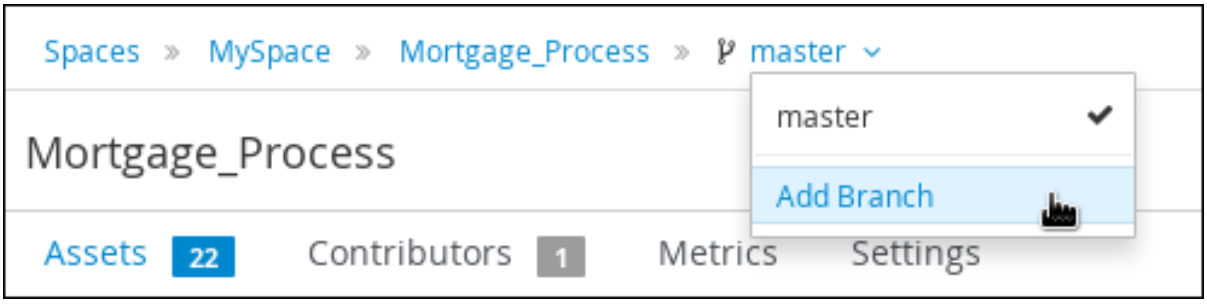
12.1. ブランチの作成

Business Central で新規ブランチを作成して、任意の名前を指定できます。最初は、デフォルトの **master** ブランチだけが存在します。プロジェクトに新しいブランチを作成するときに、選択したブランチのコピーが作成されます。オリジナルの **master** ブランチバージョンに影響を与えることなく、新規ブランチのプロジェクトに変更を加えることができます。

手順

1. Business Central で、**Menu → Design → Projects** に移動します。
2. **Mortgage_Process** のサンプルプロジェクトなど、新規ブランチを作成するプロジェクトをクリックします。
3. **master → Add Branch** をクリックします。

   図12.1 新規ブランチの作成メニュー



4. **Name** フィールドに **testBranch1** と入力して、**Add Branch** ウィンドウから **master** を選択します。**testBranch1** は、新規ブランチに指定する名前に置き換えます。
5. **Add Branch** ウィンドウから、新規ブランチのベースとなるブランチを選択します。既存のブランチであれば、どれでも選択できます。
6. **Add** をクリックします。

図12.2 新規ブランチ追加のウィンドウ

新規ブランチの追加後に、そのブランチにリダイレクトされます。このブランチには、**master** ブランチのプロジェクト内にあったアセットがすべて含まれます。

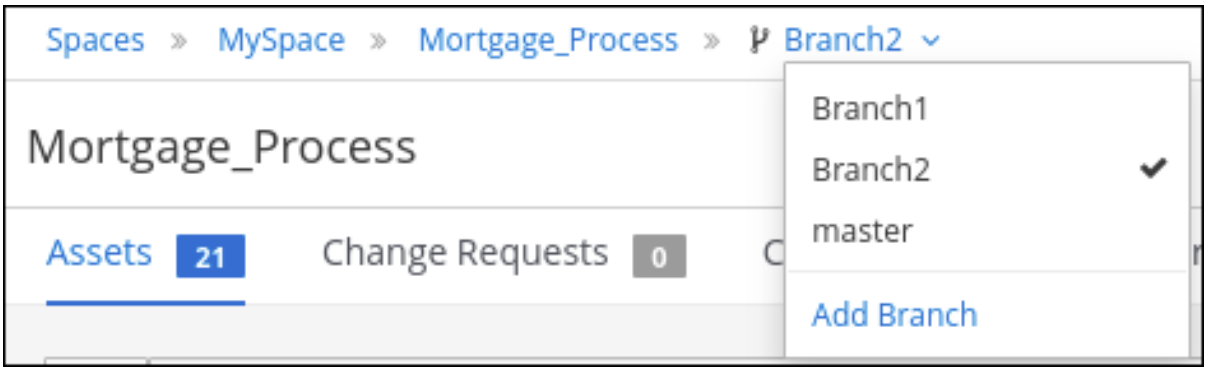
12.2. ブランチの選択

ブランチ間を切り替えて、プロジェクトアセットに変更を加えて、変更した機能をテストできます。

手順

1. 現在のブランチ名をクリックして、ドロップダウンリストから任意のプロジェクトブランチを選択します。

図12.3 ブランチメニューの選択



ブランチの選択後に、対象のプロジェクトと、定義したアセットすべてが含まれるブランチにリダイレクトされます。

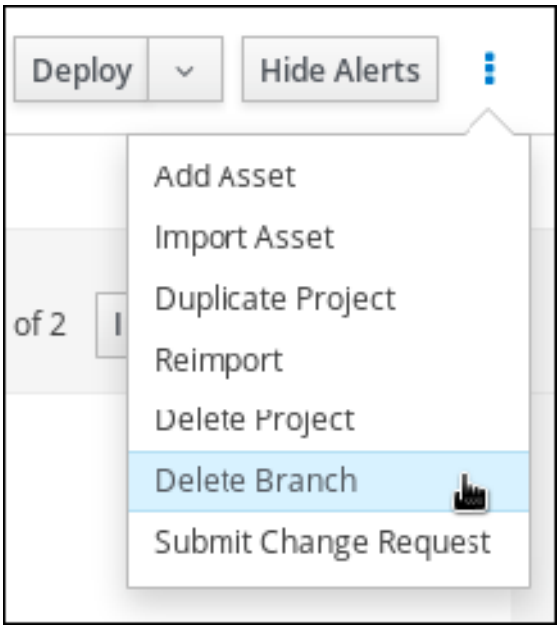
12.3. ブランチの削除

master ブランチ以外のブランチはどれでも削除できます。Business Central では、環境が壊れないように、**master** ブランチを削除できません。以下の手順は、**master** 以外のブランチで行わないと、機能しません。

手順

1. 画面の右上隅の  をクリックして、**Delete Branch** を選択します。

図12.4 ブランチの削除



2. **Delete Branch** ウィンドウで、削除するブランチの名前を入力します。
3. **Delete Branch** をクリックします。ブランチが削除され、プロジェクトブランチが master ブランチに切り替わります。

12.4. プロジェクトのビルドおよびデプロイ

プロジェクトを作成したら、Business Central の指定のブランチでプロジェクトをビルドして、設定した KIE Server にデプロイできます。

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. 右上隅の **Deploy** をクリックしてプロジェクトをビルドし、KIE Server にデプロイします。

注記

Build & Install オプションを選択してプロジェクトをビルドし、KJAR ファイルを KIE Server にデプロイせずに設定済みの Maven リポジトリーに公開することもできます。開発環境では、**Deploy** をクリックすると、ビルドされた KJAR ファイルを KIE Server に、実行中のインスタンス (がある場合はそれ) を停止せずにデプロイできます。または **Redeploy** をクリックして、ビルドされた KJAR ファイルをデプロイしてすべてのインスタンスを置き換えることもできます。次回、ビルドされた KJAR ファイルをデプロイまたは再デプロイすると、以前のデプロイメントユニット (KIE コンテナー) が同じターゲット KIE Server で自動的に更新されます。実稼働環境では **Redeploy** オプションは無効になっており、**Deploy** をクリックして、ビルドされた KJAR ファイルを KIE Server 上の新規デプロイメントユニット (KIE コンテナー) にデプロイすることのみが可能です。

KIE Server の環境モードを設定するには、**org.kie.server.mode** システムプロパティーを **org.kie.server.mode=development** または **org.kie.server.mode=production** に設定します。Business Central の対応するプロジェクトでのデプロイメント動作を設定するには、プロジェクトの **Settings** → **General Settings** → **Version** に移動し、**Development Mode** オプションを選択します。デフォルトでは、KIE Server および Business Central のすべての新規プロジェクトは開発モードになっています。**Development Mode** をオンにしたプロジェクトをデプロイしたり、実稼働モードになっている KIE Server に手動で **SNAPSHOT** バージョンの接尾辞を追加したプロジェクトをデプロイしたりすることはできません。

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。

プロジェクトのデプロイメントに関する詳細を確認するには、画面の上部にあるデプロイメントバナーの **View deployment details** か、**Deploy** のドロップダウンメニューをクリックします。このオプションを使用すると、**Menu** → **Deploy** → **Execution Servers** ページに移動します。

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。

第13章 BUSINESS CENTRAL での要求の変更

Business Central プロジェクトに複数のブランチがあり、ブランチに変更を加えて別のブランチにマージする場合に、変更要求を作成できます。ターゲットのブランチ (通常は **master** ブランチ) を表示するパーミッションがあるユーザーには、この変更要求が表示されます。

13.1. 変更要求の作成

プロジェクトに変更を加えた後 (例: アセットに対する属性追加や削除後など) に、Business Central プロジェクトで変更要求を作成できます。

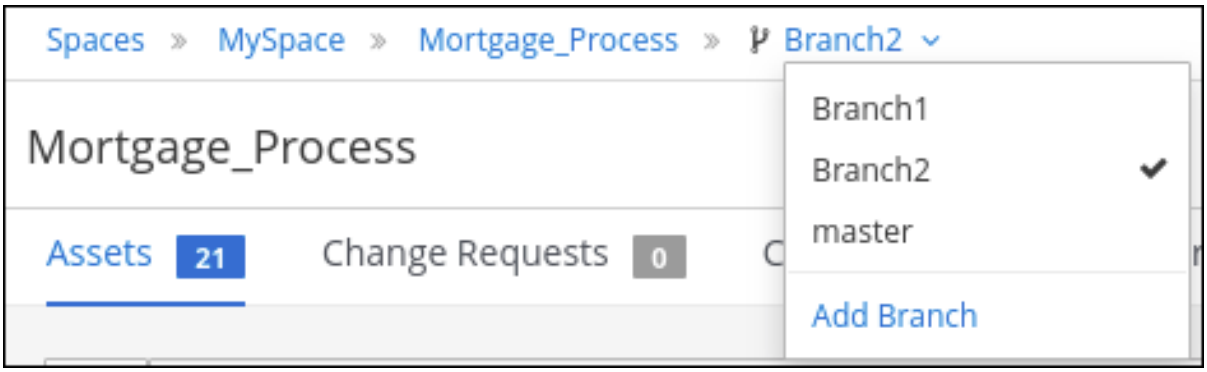
前提条件

- Business Central プロジェクトにブランチが複数ある。
- 別のブランチにマージするブランチに、変更を加えた。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、マージする変更内容が含まれるスペースとプロジェクトを選択します。
2. プロジェクトページで、変更が含まれるブランチを選択します。

   図13.1 ブランチメニューの選択

3. 変更要求を送信するには、以下のいずれかのタスクを実行します。
   - 画面の右上隅の  をクリックして、**Submit Change Request** を選択します。
   - **Change Requests** タブをクリックし、**Submit Change Request** をクリックします。**Submit Change Request** ウィンドウが表示されます。
4. 概要と説明を入力し、ターゲットブランチを選択して **Submit** をクリックします。ターゲットブランチは、変更のマージ先のブランチです。**Submit** をクリックすると、変更要求ウィンドウが表示されます。

13.2. 変更要求の使用

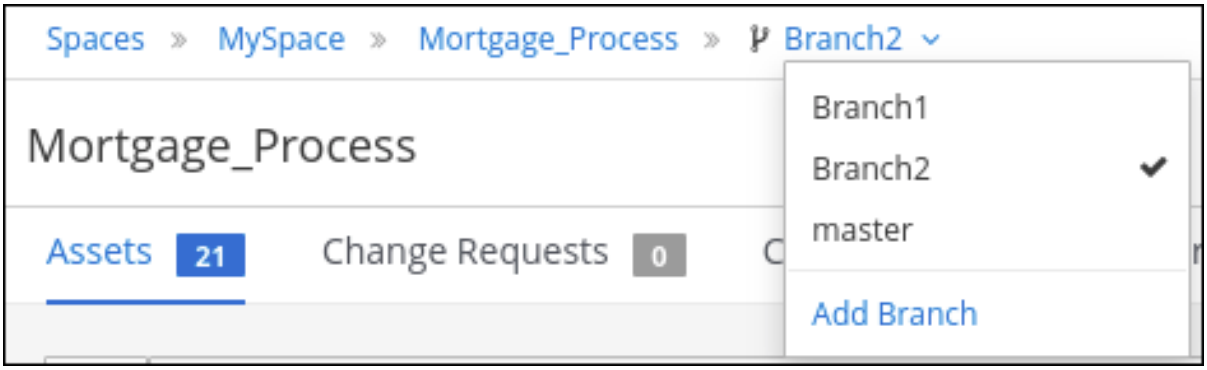
アクセス権のあるブランチの変更要求を表示できます。変更要求を受け入れるには、管理者権限が必要です。

前提条件

- Business Central プロジェクトにブランチが複数ある。

手順

1. Business Central で、**Menu → Design → Projects** に移動して、スペースとプロジェクトを選択します。
2. プロジェクトページで、正しいブランチを指定していることを確認します。



3. **Change Requests** タブをクリックします。保留中の変更要求の一覧が表示されます。
4. 変更要求をフィルターリングするには、**Search** ボックスの左側にある **Open**、**Closed**、または **All** を選択します。
5. 特定の変更要求を検索するには、**Search** ボックスに ID またはテキストを入力し、拡大鏡をクリックします。
6. 変更要求の詳細を表示するには、概要リンクをクリックします。変更要求ウィンドウには、タブが 2 つあります。
   a. 変更要求に関する全般情報については、**Overview** タブを確認してください。
   b. **Changed Files** タブをクリックし、ファイルを展開して、変更案を確認します。
7. 右上隅のボタンをクリックします。
   - **Squash and Merge** をクリックして、全コミットを 1 つのコミットにまとめて、まとめたコミットをターゲットブランチにマージします。
   - 変更をターゲットブランチにマージするには、**Merge** をクリックします。
   - また、変更を拒否し、ターゲットブランチに変更を加えないようにするには、**Reject** をクリックします。
   - 拒否も許可もせずに、変更要求を閉じるには、**Close** をクリックします。変更要求を閉じることができるのは、この要求を送信したユーザーのみです。
   - 変更せずにプロジェクトウィンドウに戻るには、**Cancel** をクリックします。

パート III. BUSINESS CENTRAL におけるアセットの管理

プロセス管理者は、Red Hat Process Automation Manager の Business Central を使用して、ルール、ビジネスプロセス、デシジョンテーブルなどのアセットを管理します。

前提条件

- Red Hat JBoss Enterprise Application Platform 7.4 がインストールされている。詳細情報は、Red Hat JBoss Enterprise Application Platform 7.4 Installation Guideを参照してください。
- Red Hat Process Automation Manager がインストールされ、KIE Server で設定されている。詳細は Red Hat JBoss EAP 7.4 への Red Hat Process Automation Manager のインストールおよび設定 を参照してください。
- Red Hat Process Automation Manager が稼働し、**developer** ロールで Business Central にログインできる。詳細は、Red Hat Process Automation Manager インストールの計画を参照してください。

第14章 アセットの概要

Business Central で作成されるビジネスルール、プロセス定義ファイル、その他のアセットおよびリソースは、KIE Server がアクセスするアーティファクトリーポジトリー (ナレッジストア) に保存されます。

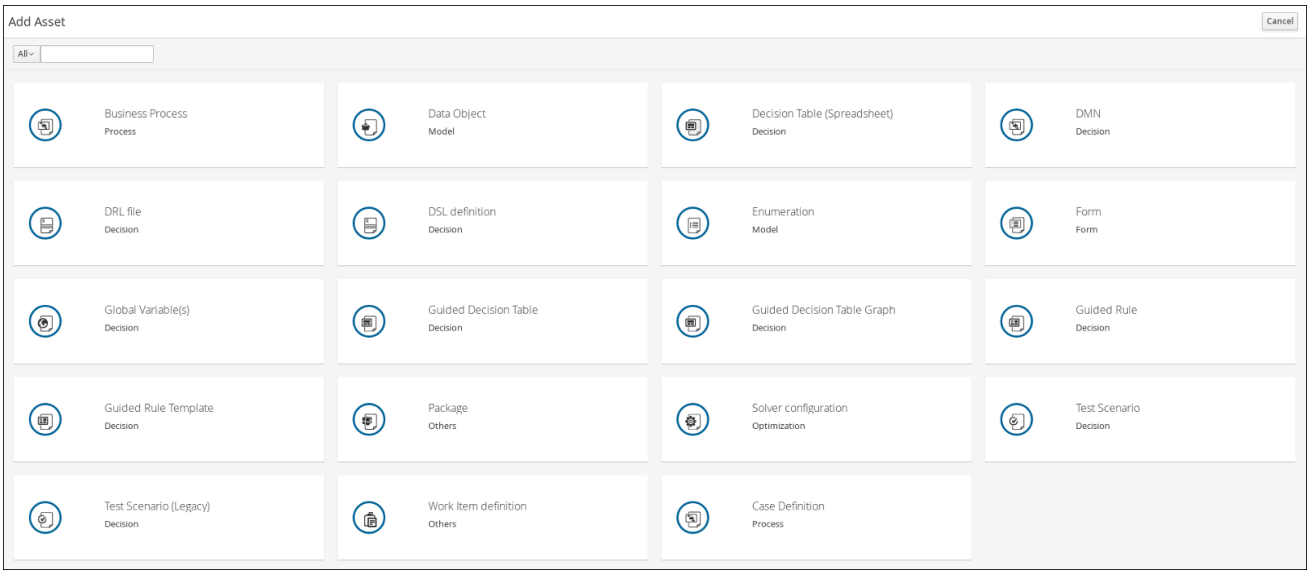
アーティファクトリーポジトリーは、ビジネスナレッジを保存するために一元化されたリポジトリーのことです。複数の GIT リポジトリーに接続して、異なる場所にあるさまざまな種類のナレッジおよびアーティファクトを保存し、1 つの環境から GIT リポジトリーに接続します。GIT は分散バージョン管理システムであり、リビジョンをコミットオブジェクトとして実装します。変更をリポジトリーに保存するたびに、GIT リポジトリーに新規コミットオブジェクトが作成されます。同様に、既存リポジトリーをコピーすることもできます。通常、このコピープロセスはクローンと呼ばれ、作成されるリポジトリーはクローンと呼ばれます。すべてのクローンには、ファイルのコレクションの完全な履歴が含まれ、クローンされたリポジトリーには、元のリポジトリーと同じコンテンツが含まれます。

Business Central は、保存したコンテンツを表示して更新できる Web フロントエンドを提供します。アーティファクトリーポジトリーアセットにアクセスするには、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。

第15章 アセットの種類

Business Central リポジトリーでバージョン管理されているものはすべてアセットです。プロジェクトには、ルール、パッケージ、ビジネスプロセス、デシジョンテーブル、ファクトモデル、ドメイン固有言語 (DSL)、またはプロジェクト要件に固有のその他のアセットを含めることができます。

次の図では、Red Hat Process Automation Manager 7.12 で利用可能なアセットを紹介します。



注記

ケース管理 (プレビュー) およびケース定義のアセットタイプは、ケースプロジェクトでのみ使用できます。

以下のセクションでは、Red Hat Process Automation Manager 7.12 の各アセットを説明します。

- ビジネスプロセス
  ビジネスプロセスは、ビジネス目標を達成するのに必要なステップを示すダイアグラムです。

- ケース管理 (プレビュー)
  ケース管理は、Business Process Management (BPM) の拡張機能で、適用可能なビジネスプロセスを管理します。ケース管理は、ルーティンで、予測可能なタスクを対象する BPM の効率指向アプローチとは対照的に、繰り返さず、予測できないプロセスに対する問題解決を提供します。ここでは、プロセスが前もって予測できない、一回限りの状況が管理されます。

重要

ビジネスプロセスアプリケーションサンプルには、テクノロジープレビューとして提供されている機能が含まれます。テクノロジープレビュー機能は、Red Hat 製品のサービスレベルアグリーメント (SLA) ではサポートされておらず、機能的に完全ではない可能性があるため、実稼働環境での使用は推奨されません。テクノロジープレビュー機能は、最新の製品機能をいち早く提供し、お客様には開発段階で機能性をテストし、フィードバックをお寄せいただくことができます。

- ケース定義
  ケースは、Business Central のケース定義プロセスデザイナーを使用して作成されます。ケース作成は、ケース管理に基づいて、各ケースに固有のゴールおよびタスクを設定します。ケースフローは、動的タスクまたはプロセスを追加して、実行時に動的に変更できます。

- 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設定要素です。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タイプです。たとえば、データフィールドの Name、Address、および Date of Birth を使用して、ローン申請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タイプは、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。

- デシジョンテーブル (スプレッドシート)
  デシジョンテーブルは、スプレッドシートまたは Red Hat Decision Manager ユーザーインターフェイスにガイド付きデシジョンテーブルとして保存されるルールの集まりです。外部の XLS ファイルまたは XLSX ファイルにルールを定義したら、Business Central のプロジェクトに、そのファイルをデシジョンテーブルとしてアップロードします。

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。

- DMN
  Decision Model and Notation (DMN) は、ビジネスデシジョンの設計とデシジョン実装の間のギャップを標準的に埋めていきます。Business Central の DMN デザイナーを使用すると、DMN 意思決定要件ダイアグラム (DRD) を設計し、完全で機能的な DMN 意思決定モデルの意思決定論理を定義できます。

- DRL ファイル
  ルールファイルは一般的に、.drl 拡張子を持つファイルです。DRL ファイルには、複数のルール、クエリー、関数だけでなく、お使いのルールやクエリーが割り当てて、使用する import、global、属性などのリソース宣言が含まれています。ただし、複数のルールファイルでルールを使用することもできます (その場合は、.rule という拡張子が推奨されますが必須ではありません)。ルールを複数のファイルで使用すると、多くのルールを管理しやすくなります。DRL ファイルは単なるテキストファイルです。

- DSL 定義
  ドメイン固有言語 (または DSL) は、問題があるドメインに対するルール言語を作成するために使用します。DSL 定義のセットは、DSL センテンスから DRL コンストラクトへの変換から設定され、基礎となるすべてのルール言語とデシジョンエンジン機能の使用を可能にします。

- 列挙
  データの列挙は任意のアセットタイプで、ガイド付きデザイナーのドロップダウンリストを提供するように設定できます。これらは他のアセットと同じように保存および編集され、所属するパッケージに適用されます。

- フォーム
  フォームは、ビジネスプロセスのユーザーデータを集めるのに使用されます。Business Central は、フォームを自動的に生成するオプションを提供しますが、特定のビジネスプロセス要件を満たすように変更できます。

- グローバル変数
  グローバル変数を使用すると、ルールに利用できるアプリケーションオブジェクトを作成できます。一般的に、グローバル変数は、ルールが使用するデータまたはサービス (特に、ルール結

果で使用されるアプリケーションサービス) を提供したり、ルールからデータ (ルール結果で追加されるログや値など) を返したり、ルールがアプリケーションと対話してコールバックを行ったりするために使用されます。

- ガイド付きデシジョンテーブル
  デシジョンテーブルは、スプレッドシートまたは Red Hat Decision Manager ユーザーインターフェイスにガイド付きデシジョンテーブルとして保存されるルールの集まりです。

- ガイド付きデシジョンテーブルのグラフ
  ガイド付きデシジョンテーブルのグラフは、1 つのデザイナーに表示される関連するガイド付きデシジョンテーブルの集まりです。このデザイナーを使用して、1 つの場所で関連するさまざまなデシジョンテーブルを視覚化し、使用できます。さらに、条件またはアクションが同じデータ型を、別のテーブルの条件またはアクションとしても使用する場合、テーブルは、テーブルのグラフデザイナーの行に物理的にリンクされます。

  たとえば、1 つのデシジョンテーブルがローンの申請料を決定し、別のテーブルがその申請料を使用してその他のアクションを決定する場合、ガイド付きデシジョンテーブルのグラフでは 2 つのデシジョンテーブルがリングされます。

- ガイド付きルール
  ルールは、デシジョンエンジンが実行するロジックを提供します。ルールには、名前、属性、ルールの左側にある **when** ステートメント、およびルールの右側にある **then** ステートメントが含まれます。

- ガイド付きルールテンプレート
  ガイド付きルールテンプレートは、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る、複数のルールで再利用可能なルール構造を提供します。

- パッケージ
  すべてのアセットは Business Central のパッケージに含まれます。パッケージはルールのディレクトリー、および名前空間となります。

- Solver の設定
  Solver 設定は Solver デザイナーにより作成され、KJAR のデプロイ後に Execution Solver またはプレーンな Java コードで実行できます。Business Central に Solver 設定を修正および作成できます。

- テストシナリオ
  Red Hat Process Automation Manager のテストシナリオでは、ルール、モデル、およびイベントの機能を実稼働環境にデプロイする前に検証できます。テストシナリオでは、ファクトまたはプロジェクトモデルのインスタンスと類似する条件のデータを使用します。このデータは指定のルールセットと照合され、想定された結果が実際の結果と一致するとテストに成功します。想定された結果が実際の結果と一致しないと、テストは失敗します。

- テストシナリオ (レガシー)
  デフォルトのテストシナリオアセットが開発中であるため、Red Hat Process Automation Manager 7.12 には、レガシーのテストシナリオのサポートが含まれています。

- 作業アイテムの定義
  作業アイテムの定義は、カスタムタスクの表示方法を定義します。たとえば、タスク名、アイコン、パラメーター、その他の属性などです。

第16章 アセットの作成

Business Central プロジェクトに、ビジネスプロセス、ルール、DRL ファイル、その他のアセットを作成できます。

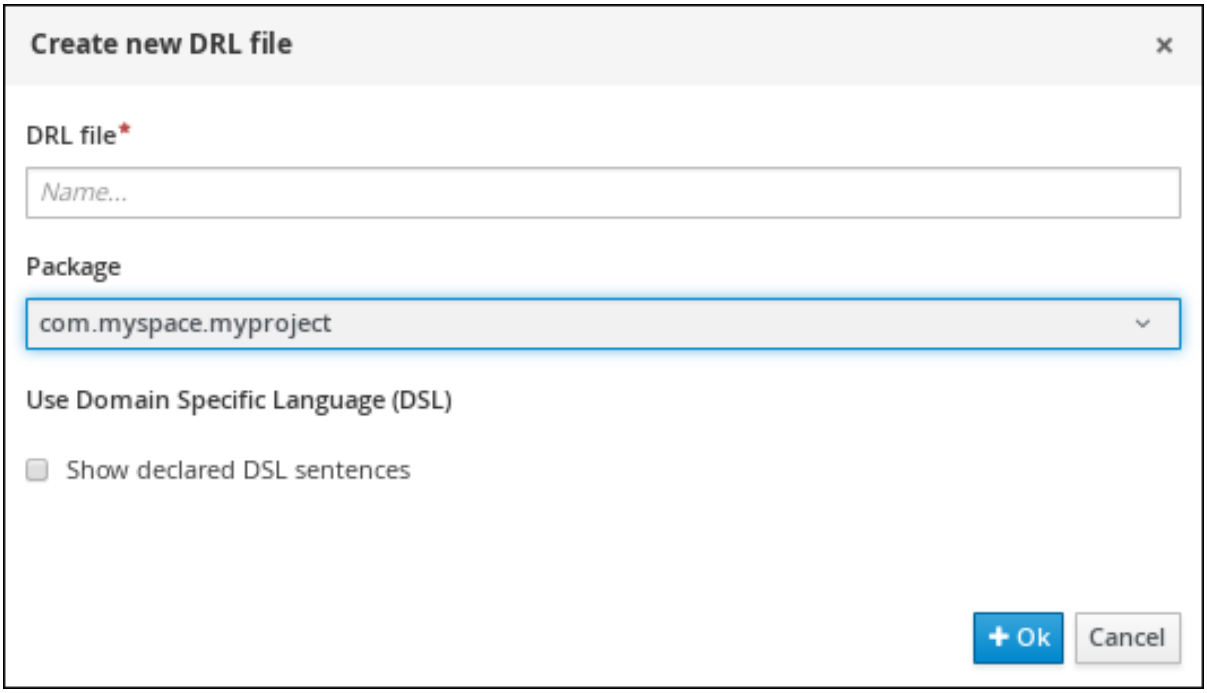
注記

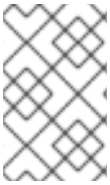
ビジネスプロセスを移行した場合は元に戻すことができません。

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。たとえば **Evaluation** です。
2. **Add Asset** をクリックし、アセットの種類を選択します。
3. **Create new asset_type** ウィンドウに必要な情報を追加して、**OK** をクリックします。

図16.1 アセットの定義





注記

プロジェクトを作成していない場合は、プロジェクトを追加するか、サンプルプロジェクトを使用するか、既存のプロジェクトをインポートします。詳細は、Business Central におけるプロジェクトの管理 を参照してください。

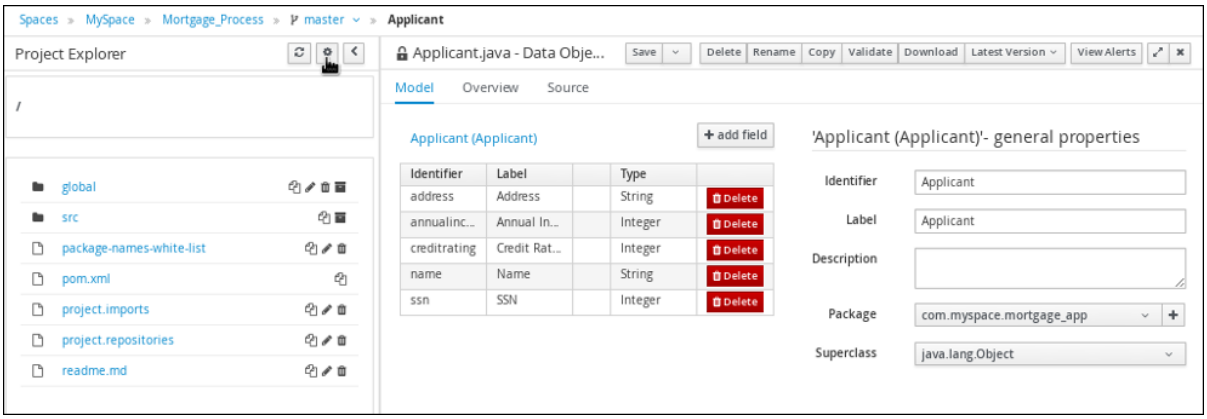
第17章 アセットの名前変更、コピー、または削除

アセットを作成して定義したら、必要に応じて **Project Explorer** の **Repository View** を使用して、アセットのコピー、名前変更、削除、またはアーカイブを行います。

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. アセット名をクリックし、左上隅の  をクリックして **Project Explorer** を展開します。
3. **Project Explorer** ツールバーで  をクリックし、**Repository View** を選択してアセットを設定するディレクトリーとファイルを表示します。
4. 必要に応じて、一覧にあるアセットのコピー、名前変更、削除、またはアーカイブを行う各アセットの横にあるアイコンを使用します。すべてのアセットに利用できないオプションもあります。

図17.1 アセットのコピー、名前変更、削除、またはアーカイブ



5. アセットのコピー、名前変更、削除には、以下のツールバーボタンを使用します。

図17.2 ツールバーオプション

第18章 アセットのメタデーターとバージョン履歴の管理

Business Central の多くのアセットには、アセットに関連付けられたメタデータおよびバージョン情報があり、プロジェクトでのアセットの識別と整理に利用できます。Business Central のアセットデザイナーで、アセットのメタデータおよびバージョン履歴を管理できます。

手順

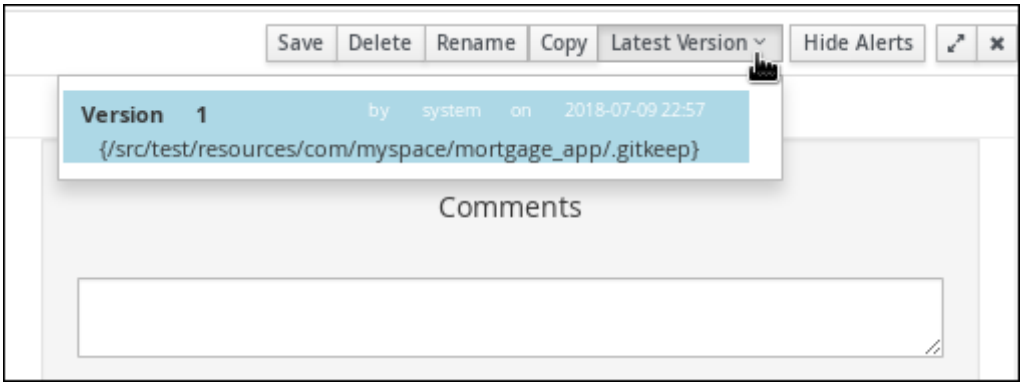
1. Business Central で、**Menu → Design → Projects** に移動して、プロジェクト名をクリックします。
2. 一覧からアセットを選択して、アセットデザイナーを開きます。
3. アセットデザイナーウィンドウで **Overview** を選択します。アセットに **Overview** タブがない場合は、メタデータがそのアセットに関連付けられていません。

4. **Version History** タブまたは **Metadata** タブを選択して、バージョンおよびメターデータの詳細を編集および更新します。

注記

または、アセットデザイナーの右上の **Latest Version** をクリックしても、アセットの作業バージョンを更新できます。

図18.1 アセットの最新バージョン



5. **Save** をクリックして変更を保存します。

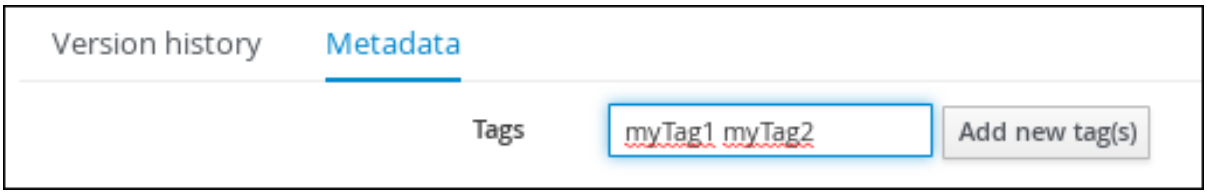
第19章 タグによるアセットのフィルターリング

Project Explorer で、各アセットのメタデータにタグを適用し、その後タグでアセットをまとめることができます。この機能を使用すれば、特定のカテゴリーのアセットをすばやく検索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. アセット名をクリックしてアセットエディターを開きます。
3. アセットエディターウィンドウで、**Overview → Metadata** に移動します。
4. **Tags** フィールドで、新しいタグの名前を入力し、**Add new tag(s)** をクリックします。空白文字でタグ名を区切り、複数のタグをアセットに割り当てることができます。

   図19.1 タグの作成

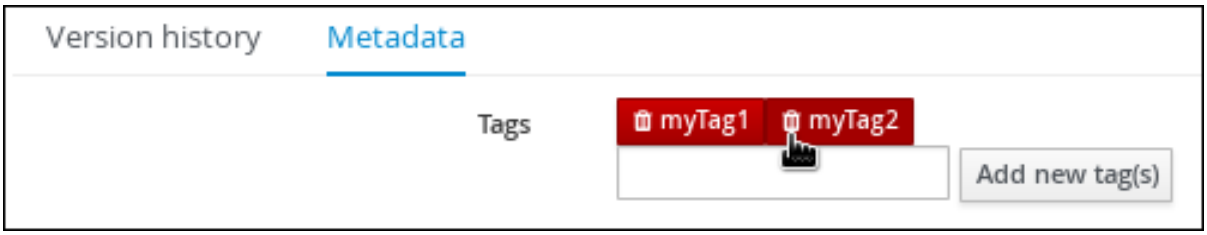
   割り当てたタグは、ボタンとして **Tags** フィールドの横に表示されます。

   図19.2 メタデータビューのタグ

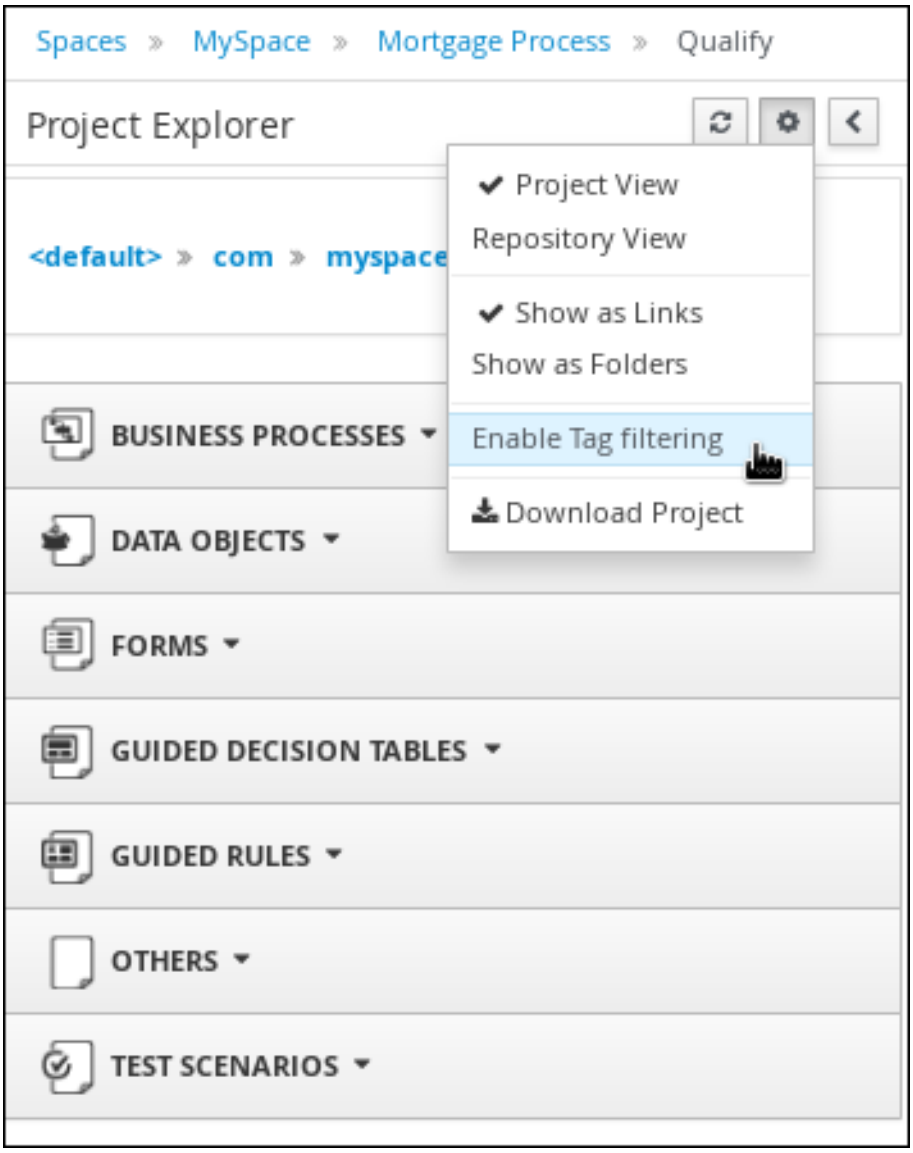
   タグボタンのゴミ箱アイコンをクリックしてタグを削除します。

   図19.3 メタデータビューでタグの削除

5. **Save** をクリックして、メタデータ変更を保存します。
6. 左上隅の  をクリックして **Project Explorer** を展開します。
7. **Project Explorer** ツールバーで  をクリックし、**Enable Tag filtering** を選択します。

図19.4 タグフィルターリングの有効化



これにより、Project Explorer に **Filter by Tag** ドロップダウンメニューが表示されます。

図19.5 タグによるフィルター

このフィルターでアセットを並べ替え、選択したメタデータタグを含むすべてのアセットおよびサービスタスクを表示します。

第20章 アセットのロック解除

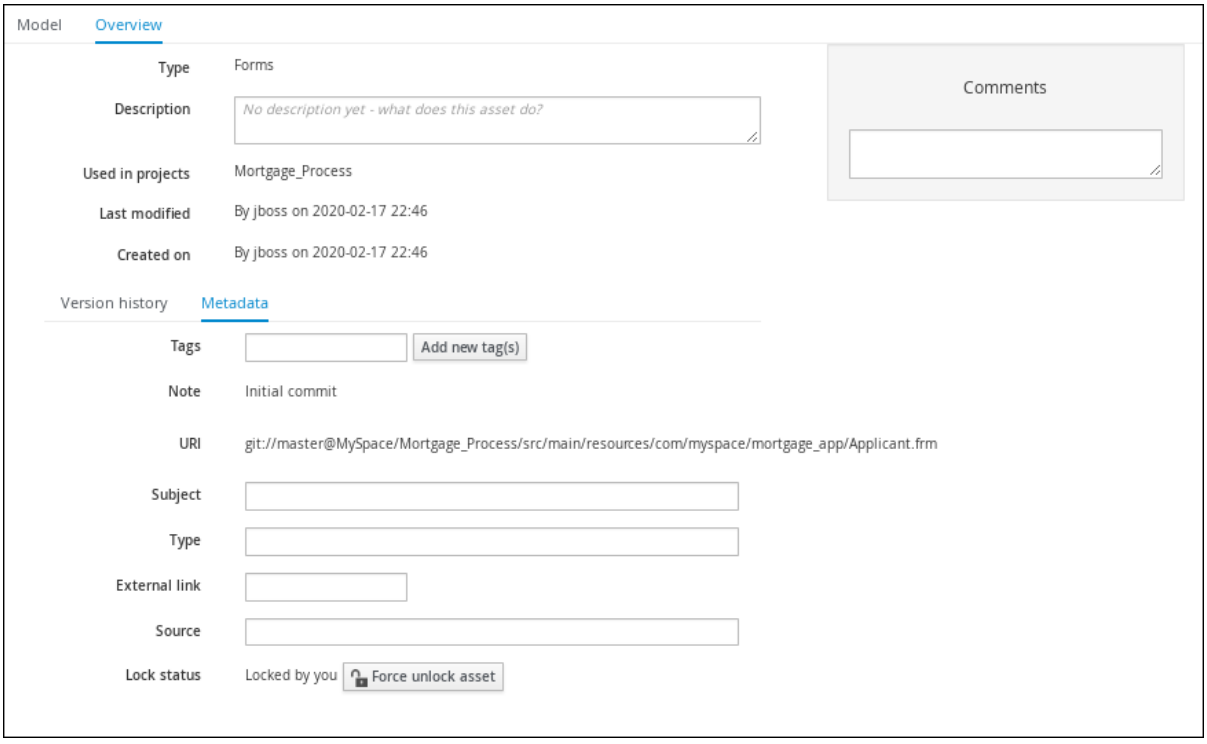
デフォルトでは、Business Central でアセットを開いて修正すると、アセットは自動的にロックされ、マルチユーザー設定で競合しないように排他的に使用されます。セッションが終了した場合や、アセットを保存または閉じない場合は、ロックが自動的に解除されます。このロック機能により、ユーザーが互いに変更を上書きできないようにします。

ただし、別のユーザーがロックしたファイルの編集が必要な場合は、アセットのロックを強制的に解除できます。

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. 一覧からアセットを選択して、アセットデザイナーを開きます。
3. **Overview → Metadata** に移動し、**Lock Status** を表示します。

   図20.1 メタデータのロック解除ビュー



   アセットが別のユーザーによってすでに編集されている場合は、以下が **Lock status** フィールドに表示されます。

   Locked by <user_name>

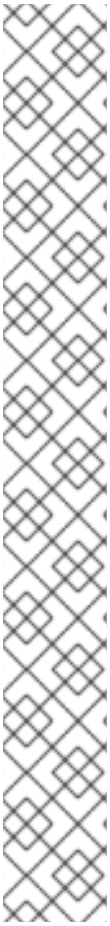
4. **Force unlock asset** をクリックして、ロックを解除します。
   以下の確認ポップアップメッセージが表示されます。

   Are you sure you want to release the lock of this asset?This might cause <user_name> to lose unsaved changes!

5. **Yes** をクリックして確定します。
   アセットのロックが解除された状態に戻り、ロックアイコンオプションがアセットの横に表示されます。

パート IV. KIE API を使用した RED HAT PROCESS AUTOMATION MANAGER の操作

ビジネスルールの作成者やシステム管理者は、KIE API を使用して Red Hat Process Automation Manager の KIE Server、KIE コンテナー、およびビジネスアセットを操作できます。KIE コンテナーおよびビジネスアセット (ビジネスルール、プロセス、ソルバーなど) には KIE Server REST API と Java クライアント API を、KIE Server テンプレートとインスタンスには Process Automation Manager コントローラー REST API と Java クライアント API を、Business Central 内のスペースとプロジェクトには Knowledge Store REST API を使用して操作します。



KIE SERVER および PROCESS AUTOMATION MANAGER コントローラー向け REST API エンドポイント

KIE Server および Process Automation Manager コントローラー向け REST API エンドポイント一覧は本書とは別に発行されており、エンドポイントオプションとデータが最新のものに維持されるように、動的にメンテナンスされています。KIE Server および Process Automation Manager コントローラー REST API で可能になることとその使い方については本書を使用し、特定エンドポイントの詳細については別にメンテナンスされている REST API エンドポイント一覧を参照してください。

KIE Server REST API エンドポイントの完全一覧と説明については、以下の関連資料を参照してください。

- jBPM ドキュメントページ (静的) の Execution Server REST API
- **http://SERVER:PORT/kie-server/docs** (動的。稼働中の KIE Server が必要) ページの KIE Server REST API 用 Swagger UI

Process Automation Manager コントローラー REST API エンドポイントの完全一覧と説明は、以下のリソースを参照してください。

- jBPM ドキュメントページ (静的) の Controller REST API
- **http://SERVER:PORT/CONTROLLER/docs** (動的。稼働中の Process Automation Manager コントローラーが必要) ページの Process Automation Manager コントローラー REST API 用 Swagger UI

前提条件

- Red Hat Process Automation Manager をインストールして実行している。インストールおよび起動オプションは、**Red Hat Process Automation Manager インストールの計画**を参照してください。
- 以下のロールを持つユーザーで Red Hat Process Automation Manager へのアクセスがある。
  - **kie-server**: KIE Server API 機能、および Business Central なしでヘッドレス Process Automation Manager コントローラー API 機能にアクセスするため (該当する場合)。
  - **rest-all**: ビルトインの Process Automation Manager コントローラーおよび Business Central ナレッジストア用の Business Central API 機能にアクセスするため。
  - **admin**: Red Hat Process Automation Manager への完全な管理者アクセス用。
    各 KIE API ですべてのユーザーロールが必要なわけではありませんが、これらすべてを取得しておくといずれの KIE API にも問題なくアクセスできるようになります。ユーザーロールについての詳細は、**Red Hat Process Automation Manager インストールの計画**を参照し

てください。

第21章 KIE コンテナーおよびビジネスアセット用の KIE SERVER REST API

Red Hat Process Automation Manager は KIE Server REST API を提供し、これを使用することで Business Central ユーザーインターフェイスを使用せずに Red Hat Process Automation Manager の KIE コンテナーやビジネスアセット (ビジネスルール、プロセス、ソルバーなど) を操作することができます。この API のサポートにより、Red Hat Process Automation Manager のリソースをより効率的に維持でき、Red Hat Process Automation Manager の統合と開発を最適化できるようになります。

KIE Server REST API を使用すると、以下のアクションが可能になります。

- KIE コンテナーのデプロイまたは破棄
- KIE コンテナー情報の取得および更新
- KIE Server ステータスおよび基本的情報の確認
- ビジネスアセット情報の取得および更新
- ビジネスアセット (ルールやプロセスなど) の実行

KIE Server REST API 要求には以下のコンポーネントが必要です。

認証

KIE Server REST API は、ユーザーロール **kie-server** に HTTP の Basic 認証またはトークンベースの認証を必要とします。お使いの Red Hat Process Automation Manager に設定されているユーザーロールを表示するには、**~/$SERVER_HOME/standalone/configuration/application-roles.properties** と **~/application-users.properties** に移動します。
ユーザーに **kie-server** ロールを追加するには、**~/$SERVER_HOME/bin** に移動して以下のコマンドを実行します。

```
$ ./bin/jboss-cli.sh --commands="embed-server --std-out=echo,/subsystem=elytron/filesystem-realm=ApplicationRealm:add-identity(identity=<USERNAME>),/subsystem=elytron/filesystem-realm=ApplicationRealm:set-password(identity=<USERNAME>, clear={password='<PASSWORD>'}),/subsystem=elytron/filesystem-realm=ApplicationRealm:add-identity-attribute(identity=<USERNAME>, name=role, value=['kie-server'])"
```

ユーザーロールと Red Hat Process Automation Manager のインストールオプションの詳細は、Red Hat Process Automation Manager インストールの計画 を参照してください。

HTTP ヘッダー

KIE Server REST API は、API 要求に以下の HTTP ヘッダーを必要とします。

- **Accept**: 要求元のクライアントが受け付けるデータ形式:
  - **application/json** (JSON)
  - **application/xml** (XML、JAXB または XSTREAM 用)
- **Content-Type**: **POST** または **PUT** API 要求データ向けのデータ形式:
  - **application/json** (JSON)
  - **application/xml** (XML、JAXB または XSTREAM 用)

- **X-KIE-ContentType**: **application/xml** XSTREAM API 要求および応答に必要なヘッダー:
  - **XSTREAM**

HTTP メソッド

KIE Server REST API は、API 要求に以下の HTTP メソッドを必要とします。

- **GET**: 指定したリソースのエンドポイントから指定した情報を取得する
- **POST**: リソースまたはリソースインスタンスを更新する
- **PUT**: リソースまたはリソースインスタンスを更新もしくは作成する
- **DELETE**: リソースまたはリソースインスタンスを削除する

ベース URL

KIE Server REST API 要求のベース URL は **http://SERVER:PORT/kie-server/services/rest/** で、たとえば **http://localhost:8080/kie-server/services/rest/** となります。

エンドポイント

特定の KIE コンテナーにおける **/server/containers/{containerId}** など、KIE Server REST API のエンドポイントは、KIE Server REST API ベース URL に追記する URI で、Red Hat Process Automation Manager の対応するリソースやリソースタイプにアクセスするためのものです。

/server/containers/{containerId} エンドポイントの要求 URL 例

http://localhost:8080/kie-server/services/rest/server/containers/MyContainer

要求パラメーターおよび要求データ

多くの KIE Server REST API 要求では、特定リソースの確認またはフィルターリングを行い、特定のアクションを実行するために、要求 URL パスで特定のパラメーターを必要とします。URL パラメーターは、**?<PARAM>=<VALUE>&<PARAM>=<VALUE>** の形式でエンドポイントに追記します。

GET 要求 URL のパラメーター例

http://localhost:8080/kie-server/services/rest/server/containers?groupId=com.redhat&artifactId=Project1&version=1.0&status=STARTED

HTTP **POST** と **PUT** の要求は、さらに要求のボディもしくはデータのあるファイルが必要になる場合があります。

POST 要求 URL と JSON 要求のボディデータの例

http://localhost:8080/kie-server/services/rest/server/containers/MyContainer/release-id

```
{
  "release-id": {
    "artifact-id": "Project1",
    "group-id": "com.redhat",
    "version": "1.1"
  }
}
```

21.1. REST クライアントまたは CURL ユーティリティーを使用した KIE SERVER REST API による要求送信

KIE Server REST API を使用すると、Business Central ユーザーインターフェイスを使わずに Red Hat Process Automation Manager の KIE コンテナーやビジネスアセット (ビジネスルール、プロセス、ソルバーなど) を操作することができます。KIE Server REST API 要求は、REST クライアントまたは curl ユーティリティーを使用して送信できます。

前提条件

- KIE Server をインストールし、実行している。
- **kie-server** ユーザーロールで KIE Server にアクセスできる。

手順

1. [GET] /server/containers など、要求の送信先に適した **API endpoint** を特定し、KIE Server から KIE コンテナーを取得します。
2. REST クライアントまたは curl ユーティリティーで、**/server/containers** への **GET** 要求に以下のコンポーネントを記入します。ご自分のユースケースに合わせて、要求詳細を調整します。
   REST クライアントの場合:
   - **Authentication**: **kie-server** ロールを持つ KIE Server ユーザーのユーザー名とパスワードを入力します。
   - **HTTP Headers**: 以下のヘッダーを設定します。
     - **Accept**: **application/json**
   - **HTTP method**: **GET** に設定します。
   - **URL**: KIE Server REST API ベース URL とエンドポイントを入力します。たとえば、**http://localhost:8080/kie-server/services/rest/server/containers** となります。

   curl ユーティリティーの場合:
   - **-u**: **kie-server** ロールを持つ KIE Server ユーザーのユーザー名とパスワードを入力します。
   - **-H**: 以下のヘッダーを設定します。
     - **Accept**: **application/json**
   - **-X**: **GET** に設定します。
   - **URL**: KIE Server REST API ベース URL とエンドポイントを入力します。たとえば、**http://localhost:8080/kie-server/services/rest/server/containers** となります。

   ```
   curl -u 'baAdmin:password@1' -H "Accept: application/json" -X GET
   "http://localhost:8080/kie-server/services/rest/server/containers"
   ```
3. 要求を実行し、KIE Server の応答を確認します。
   サーバー応答の例 (JSON):

   ```
   {
     "type": "SUCCESS",
   ```

```
"msg": "List of created containers",
"result": {
  "kie-containers": {
    "kie-container": [
      {
        "container-id": "itorders_1.0.0-SNAPSHOT",
        "release-id": {
          "group-id": "itorders",
          "artifact-id": "itorders",
          "version": "1.0.0-SNAPSHOT"
        },
        "resolved-release-id": {
          "group-id": "itorders",
          "artifact-id": "itorders",
          "version": "1.0.0-SNAPSHOT"
        },
        "status": "STARTED",
        "scanner": {
          "status": "DISPOSED",
          "poll-interval": null
        },
        "config-items": [],
        "container-alias": "itorders"
      }
    ]
  }
}
}
```


4. この例では、プロジェクトの **group-id**、**artifact-id**、および **version** (GAV) のデータを応答で返されたデプロイ済み KIE コンテナーのいずれかからコピーするか、書き留めます。

5. REST クライアントまたは curl ユーティリティーで、**/server/containers/{containerId}** への **PUT** 要求を以下のコンポーネントで送信し、コピーしたプロジェクトの GAV データで新規 KIE コンテナーをデプロイします。ご自分のユースケースに合わせて、要求詳細を調整します。REST クライアントの場合:

   - **Authentication**: **kie-server** ロールを持つ KIE Server ユーザーのユーザー名とパスワードを入力します。
   - **HTTP Headers**: 以下のヘッダーを設定します。
     - **Accept**: **application/json**
     - **Content-Type**: **application/json**

     注記

     fields=not_null を **Content-Type** に追加すると、null フィールドは REST API レスポンスから除外されます。

   - **HTTP method**: **PUT** に設定します。
   - **URL**: KIE Server REST API ベース URL とエンドポイントを入力します。たとえば、**http://localhost:8080/kie-server/services/rest/server/containers/MyContainer** となります。

- **要求のボディ**: 新規 KIE コンテナー用の設定アイテムのある JSON 要求のボディを追加します。

```
{
  "config-items": [
    {
      "itemName": "RuntimeStrategy",
      "itemValue": "SINGLETON",
      "itemType": "java.lang.String"
    },
    {
      "itemName": "MergeMode",
      "itemValue": "MERGE_COLLECTIONS",
      "itemType": "java.lang.String"
    },
    {
      "itemName": "KBase",
      "itemValue": "",
      "itemType": "java.lang.String"
    },
    {
      "itemName": "KSession",
      "itemValue": "",
      "itemType": "java.lang.String"
    }
  ],
  "release-id": {
    "group-id": "itorders",
    "artifact-id": "itorders",
    "version": "1.0.0-SNAPSHOT"
  },
  "scanner": {
    "poll-interval": "5000",
    "status": "STARTED"
  }
}
```

curl ユーティリティーの場合:

- **-u**: **kie-server** ロールを持つ KIE Server ユーザーのユーザー名とパスワードを入力します。
- **-H**: 以下のヘッダーを設定します。
  - **Accept**: **application/json**
  - **Content-Type**: **application/json**

注記

fields=not_null を **Content-Type** に追加すると、null フィールドは REST API レスポンスから除外されます。

- **-X**: **PUT** に設定します。

- **URL**: KIE Server REST API ベース URL とエンドポイントを入力します。たとえば、**http://localhost:8080/kie-server/services/rest/server/containers/MyContainer** となります。
- **-d**: 新規 KIE コンテナー用の設定アイテムのある JSON 要求のボディまたはファイル (**@file.json**) を追加します。

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Content-Type: application/json" -X PUT "http://localhost:8080/kie-server/services/rest/server/containers/MyContainer" -d "{ \"config-items\": [ { \"itemName\": \"RuntimeStrategy\", \"itemValue\": \"SINGLETON\", \"itemType\": \"java.lang.String\" }, { \"itemName\": \"MergeMode\", \"itemValue\": \"MERGE_COLLECTIONS\", \"itemType\": \"java.lang.String\" }, { \"itemName\": \"KBase\", \"itemValue\": \"\", \"itemType\": \"java.lang.String\" }, { \"itemName\": \"KSession\", \"itemValue\": \"\", \"itemType\": \"java.lang.String\" } ], \"release-id\": { \"group-id\": \"itorders\", \"artifact-id\": \"itorders\", \"version\": \"1.0.0-SNAPSHOT\" }, \"scanner\": { \"poll-interval\": \"5000\", \"status\": \"STARTED\" }}"
```

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Content-Type: application/json" -X PUT "http://localhost:8080/kie-server/services/rest/server/containers/MyContainer" -d @my-container-configs.json
```

6. 要求を実行し、KIE Server の応答を確認します。
サーバー応答の例 (JSON):


```
{
  "type": "SUCCESS",
  "msg": "Container MyContainer successfully deployed with module itorders:itorders:1.0.0-SNAPSHOT.",
  "result": {
    "kie-container": {
      "container-id": "MyContainer",
      "release-id": {
        "group-id": "itorders",
        "artifact-id": "itorders",
        "version": "1.0.0-SNAPSHOT"
      },
      "resolved-release-id": {
        "group-id": "itorders",
        "artifact-id": "itorders",
        "version": "1.0.0-SNAPSHOT"
      },
      "status": "STARTED",
      "scanner": {
        "status": "STARTED",
        "poll-interval": 5000
      },
      "config-items": [],
      "messages": [
        {
          "severity": "INFO",
          "timestamp": {
            "java.util.Date": 1540584717937
          },
          "content": [
```

```
            "Container MyContainer successfully created with module itorders:itorders:1.0.0-
SNAPSHOT."
          ]
        }
      ],
      "container-alias": null
    }
  }
}
```

要求エラーが発生した場合は、返されたエラーコードメッセージを確認して、それに応じて要求を調整します。

プロセスインスタンスの REST API 要求

複雑なデータオブジェクトをプロセスインスタンスのエンドポイント (**/server/containers/{containerId}/processes/{processId}/instances**) に送信する REST API 要求の場合は、要求ボディーに、完全修飾クラス名 (**com.myspace.Person** など) または単純なクラス名 (**Person** など) を含めるようにしてください。Red Hat Process Automation Manager で、正しいビジネスオブジェクトに要求ボディーをマッピングするには、クラス名が必要です。要求からクラス名を除外すると、KIE Server では、想定するタイプにオブジェクトがアンマーシャルされません。

プロセスインスタンスの要求ボディー (正)

```
{
  "id": 4,
  "lease": {
    "com.myspace.restcall.LeaseModel": {
      "annualRent": 109608,
      "isAutoApproved": false
    }
  }
}
```

プロセスインスタンスの要求ボディー (誤)

```
{
  "id": 4,
  "lease": {
    "annualRent": 109608,
    "isAutoApproved": false
  }
}
```

21.2. SWAGGER インターフェイスを使用した KIE SERVER REST API による要求送信

KIE Server REST API は Swagger Web インターフェイスをサポートします。スタンドアロンの REST クライアントや curl ユーティリティーの代わりのこれを使用すると、Business Central ユーザーインターフェイスを使わずに Red Hat Process Automation Manager の KIE コンテナーやビジネスアセット (ビジネスルールやプロセス、ソルバーなど) を操作することができます。

注記

デフォルトでは、**org.kie.swagger.server.ext.disabled=false** システムプロパティーが指定されており、KIE Server の Swagger Web インターフェイスが有効になっています。KIE Server で Swagger Web インターフェイスを無効にするには、このシステムプロパティーを **true** に設定してください。

前提条件

- KIE Server をインストールし、実行している。
- **kie-server** ユーザーロールで KIE Server にアクセスできる。

手順

1. Web ブラウザーで **http://SERVER:PORT/kie-server/docs** を開きます。たとえば、**http://localhost:8080/kie-server/docs** となります。**kie-server** ロールを持つ KIE Server ユーザーのユーザー名とパスワードでログインします。
2. Swagger ページで、要求の送信先となる関連 API エンドポイントを選択します。たとえば、**KIE Server and KIE containers→ [GET] /server/containers** で KIE コンテナーを KIE Server から取得します。
3. **Try it out** をクリックして、結果のフィルターリングに使用する任意のパラメーターを提供します。
4. **Response content type** ドロップダウンメニューで、サーバー応答のフォーマットを選択します (例: JSON フォーマットでは **application/json**)。
5. **Execute** をクリックし、KIE Server の応答を確認します。
サーバー応答の例 (JSON):


```
{
  "type": "SUCCESS",
  "msg": "List of created containers",
  "result": {
    "kie-containers": {
      "kie-container": [
        {
          "container-id": "itorders_1.0.0-SNAPSHOT",
          "release-id": {
            "group-id": "itorders",
            "artifact-id": "itorders",
            "version": "1.0.0-SNAPSHOT"
          },
          "resolved-release-id": {
            "group-id": "itorders",
            "artifact-id": "itorders",
            "version": "1.0.0-SNAPSHOT"
          },
          "status": "STARTED",
          "scanner": {
            "status": "DISPOSED",
            "poll-interval": null
          },
          "config-items": [],
```

```
          "container-alias": "itorders"
        }
       ]
     }
   }
}
```

6. この例では、プロジェクトの **group-id**、**artifact-id**、および **version** (GAV) のデータを応答で返されたデプロイ済み KIE コンテナーのいずれかからコピーするか、書き留めます。

7. Swagger ページで **KIE Server and KIE containers→ [PUT] /server/containers/{containerId}** エンドポイントに移動し、コピーしたプロジェクト GAV データで新規 KIE コンテナーをデプロイするための別の要求を送信します。ご自分のユースケースに合わせて、要求詳細を調整します。

8. **Try it out** をクリックして、以下の要求のコンポーネントを入力します。

   - **containerId**: 新規 KIE コンテナーの ID を入力します (例: **MyContainer**)。
   - **body**: **Parameter content type** を希望の要求のボディ形式 (JSON の場合は **application/json** など) に設定し、要求のボディに新規 KIE コンテナーの設定アイテムを追加します。

```
{
 "config-items": [
  {
   "itemName": "RuntimeStrategy",
   "itemValue": "SINGLETON",
   "itemType": "java.lang.String"
  },
  {
   "itemName": "MergeMode",
   "itemValue": "MERGE_COLLECTIONS",
   "itemType": "java.lang.String"
  },
  {
   "itemName": "KBase",
   "itemValue": "",
   "itemType": "java.lang.String"
  },
  {
   "itemName": "KSession",
   "itemValue": "",
   "itemType": "java.lang.String"
  }
 ],
 "release-id": {
  "group-id": "itorders",
  "artifact-id": "itorders",
  "version": "1.0.0-SNAPSHOT"
 },
 "scanner": {
  "poll-interval": "5000",
  "status": "STARTED"
 }
}
```

9. **Response content type** ドロップダウンメニューで、サーバー応答のフォーマットを選択します (例: JSON フォーマットでは **application/json**)。

10. **Execute** をクリックし、KIE Server の応答を確認します。
サーバー応答の例 (JSON):


```
{
  "type": "SUCCESS",
  "msg": "Container MyContainer successfully deployed with module itorders:itorders:1.0.0-SNAPSHOT.",
  "result": {
    "kie-container": {
      "container-id": "MyContainer",
      "release-id": {
        "group-id": "itorders",
        "artifact-id": "itorders",
        "version": "1.0.0-SNAPSHOT"
      },
      "resolved-release-id": {
        "group-id": "itorders",
        "artifact-id": "itorders",
        "version": "1.0.0-SNAPSHOT"
      },
      "status": "STARTED",
      "scanner": {
        "status": "STARTED",
        "poll-interval": 5000
      },
      "config-items": [],
      "messages": [
        {
          "severity": "INFO",
          "timestamp": {
            "java.util.Date": 1540584717937
          },
          "content": [
            "Container MyContainer successfully created with module itorders:itorders:1.0.0-SNAPSHOT."
          ]
        }
      ],
      "container-alias": null
    }
  }
}
```


要求エラーが発生した場合は、返されたエラーコードメッセージを確認して、それに応じて要求を調整します。

プロセスインスタンスの REST API 要求

複雑なデータオブジェクトをプロセスインスタンスのエンドポイント (**/server/containers/{containerId}/processes/{processId}/instances**) に送信する REST API 要求の場合は、要求ボディーに、完全修飾クラス名 (**com.myspace.Person** など) または単純なクラス名 (**Person** など) を含めるようにしてください。Red Hat Process Automation Manager で、正しいビジネスオブジェクトに要求ボディーをマッピングするには、クラス名が必要です。要求からクラス名を除外すると、KIE Server では、想定するタイプにオブジェクトがアンマーシャルされません。

プロセスインスタンスの要求ボディー (正)

```
{
  "id": 4,
  "lease": {
    "com.myspace.restcall.LeaseModel": {
      "annualRent": 109608,
      "isAutoApproved": false
    }
  }
}
```

プロセスインスタンスの要求ボディー (誤)

```
{
  "id": 4,
  "lease": {
    "annualRent": 109608,
    "isAutoApproved": false
  }
}
```

21.3. サポート対象の KIE SERVER REST API エンドポイント

KIe Server REST API は、Red Hat Process Automation Manager で以下のタイプのリソースにエンドポイントを提供します。

- KIE Server および KIE コンテナー
- KIE セッションアセット (ランタイムコマンド用)
- DMN アセット
- プラニングソルバー
- プロセス
- プロセスイメージ
- プロセスおよびタスクフォーム
- タスク
- ケース

- ドキュメント
- ジョブ
- プロセス、タスク、およびケースのクエリー
- カスタムクエリー

KIE Server REST API のベース URL は **http://SERVER:PORT/kie-server/services/rest/** です。ユーザーロール **kie-server** では、すべての要求で HTTP の Basic 認証またはトークンベースの認証が必要です。

KIE Server REST API エンドポイントの完全一覧と説明については、以下の関連資料を参照してください。

- jBPM ドキュメントページ (静的) の Execution Server REST API
- **http://SERVER:PORT/kie-server/docs** (動的。稼働中の KIE Server が必要) ページの KIE Server REST API 用 Swagger UI

注記

デフォルトでは、**org.kie.swagger.server.ext.disabled=false** システムプロパティーが指定されており、KIE Server の Swagger Web インターフェイスが有効になっています。KIE Server で Swagger Web インターフェイスを無効にするには、このシステムプロパティーを **true** に設定してください。

API がプロセスイメージにアクセスするには、システムプロパティー **<storesvgonsave enabled="true"/>** が **$SERVER_HOME/standalone/deployments/business-central.war/org.kie.workbench.KIEWebapp/profiles/jbpm.xml** で Red Hat Process Automation Manager プロジェクト用に設定されている必要があります。このプロパティーは、デフォルトで **true** に設定されています。API がプロセスイメージを使用していない場合は、ファイルで **true** に設定して KIE Server を再起動し、関連プロセスを変更して保存します。その後、プロジェクトをビルドしてデプロイします。このプロパティーの設定により SVG イメージが保存されるので、KIE Server REST API が取得できるようになります。

21.3.1. カスタムクエリー

カスタムクエリーエンドポイントを使用して、Red Hat Process Automation Manager でカスタムクエリーを作成およびアクセスできます。カスタムクエリーは、Red Hat Process Automation Manager データベースから任意のデータをリクエストできます。

Red Hat Process Automation Manager には、多数のカスタムクエリーが含まれています。これらのクエリーを使用して、プロセスインスタンスとユーザータスクの完全なリストにアクセスできます。

カスタムクエリーの実行時に、**mapper** パラメーターに **クエリーマッパー** の名前を指定する必要があります。マッパーは、SQL クエリー結果を JSON 応答のオブジェクトにマッピングします。独自のクエリー結果マッパーか、Red Hat Process Automation Manager で提供されているマッパーのいずれかを実装することができます。Red Hat Process Automation Manager のクエリーマッパーは、テーブルをエントリーにマッピングする、Hibernate などの他のオブジェクト関係マッピング (ORM) プロバイダーと同様のものです。

たとえば、カスタムクエリーがプロセスインスタンスデータを返す場合は、**org.jbpm.kie.services.impl.query.mapper.ProcessInstanceQueryMapper** マッパーを使用することもできます。これも **ProcessInstances** として登録されています。カスタムクエリーがヒューマンタスクデータを返す場合

は、**org.jbpm.kie.services.impl.query.mapper.UserTaskInstanceQueryMapper** マッパーを使用することもできます。これも **UserTasks** として登録されています。追加情報を提供する他のマッパーを使用することもできます。

Red Hat Process Automation Manager に含まれるクエリーマッパーのリストは、GitHub リポジトリーを参照してください。

21.3.2. 特定の DMN モデルの REST エンドポイント

Red Hat Process Automation Manager には、Business Central ユーザーインターフェイスを使用せずに、特定の DMN モデルとの対話すに使用できるモデル固有の DMN KIE Server エンドポイントが含まれます。

Red Hat Process Automation Manager のコンテナーの DMN モデルごとに、以下の KIE Server REST エンドポイントは、DMN モデルのコンテンツに基づいて自動的に生成されます。

- **POST /server/containers/{containerId}/dmn/models/{modelname}**: コンテナーで指定された DMN モデルを評価するための business-domain エンドポイント
- **POST /server/containers/{containerId}/dmn/models/{decisionServiceName}**: コンテナーで利用可能な特定の DMN モデルで指定のデシジョンサービスコンポーネントを評価するためのビジネスドメインエンドポイント
- **POST /server/containers/{containerId}/dmn/models/{modelname}/dmnresult**: 指定した DMN モデルの評価および business-domain コンテキスト、ヘルパーメッセージ、ヘルパーデシジョンポインターなど、**DMNResult** 応答を返すエンドポイント
- **POST /server/containers/{containerId}/dmn/models/{modelname}/{decisionServiceName}/dmnresult:** 特定の DMN モデルで指定のデシジョンサービスコンポーネントを評価して、ビジネスドメインのコンテキスト、ヘルパーメッセージ、およびヘルプデシジョンポインターなど、**DMNResult** 応答を返すエンドポイント
- **GET /server/containers/{containerId}/dmn/models/{modelname}**: デシジョンロジックなしの標準の DMN XML を返し、指定の DMN モデルの入力とデシジョンを含むエンドポイント
- **GET /server/containers/{containerId}/dmn/openapi.json(|.yaml)**: 指定のコンテナーで DMN モデルの Swagger または OAS を取得するエンドポイント

これらのエンドポイントを使用して、モデル内の DMN モデルまたは特定のデシジョンサービスと対話できます。これらの REST エンドポイントの business-domain と **dmnresult** を使用する場合は、以下の考慮事項を確認してください。

- **REST ビジネスドメインエンドポイント**: クライアントアプリケーションが正の評価結果だけを対象としていて、**Info** または **Warn** メッセージの解析を行わず、エラーに HTTP 5xx 応答だけが必要な場合にはこのエンドポイントタイプを使用します。また、このタイプのエンドポイントは、デシジョンサービスの結果のシングルトン強制が DMN のモデル動作に似ているので、単一ページアプリケーションのようなクライアントにも役立ちます。
- **REST dmnresult エンドポイント**: クライアントが **Info**、**Warn**、または **Error** メッセージの解析が必要な場合は、このエンドポイントタイプを使用します。

エンドポイントごとに、REST クライアントまたは curl ユーティリティーを使用して、以下のコンポーネントで要求を送信します。

- **ベース URL**: **http://HOST:PORT/kie-server/services/rest/**

- **パスパラメーター**:
  - **{containerId}**: **mykjar-project** などのコンテナーの文字列識別子
  - **{modelName}**: **Traffic Violation** などの DMN モデルの文字列識別子
  - **{decisionServiceName}**: **TrafficViolationDecisionService** などの DMN DRG のデシジョンサービスコンポーネントの文字列識別子
  - **dmnresult**: エンドポイントが、詳細にわたる **Info**、**Warn** および **Error** メッセージを含む、完全な **DMNResult** 応答を返すことができるようにする文字列識別子
- HTTP **ヘッダー**: **POST** 要求のみ:
  - **accept**: **application/json**
  - **content-type**: **application/json**
- HTTP **メソッド**: **GET** または **POST**

以下のエンドポイントの例は、**TrafficViolationDecisionService** デシジョンサービスコンポーネントが含まれる **Traffic Violation** DMN モデルを格納する **mykjar-project** コンテナーをもとにしています。

これらの全エンドポイントに対して、DMN 評価の **Error** メッセージが発生すると、**DMNResult** 応答が HTTP 5xx エラーと共に返されます。DMN の **Info** または **Warn** メッセージが表示されると、クライアント側のビジネスロジックに使用される **X-Kogito-decision-messages** 拡張 HTTP ヘッダーで、business- domain REST ボディーと共に関連する応答が返されます。クライアント側に詳細にわたるビジネスロジックの要件がある場合には、クライアントはエンドポイントの **dmnresult** バリアントを使用できます。

指定のコンテナー内の DMN モデルの Swagger または OAS 取得

GET /server/containers/{containerId}/dmn/openapi.json (|.yaml)

REST エンドポイントの例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/openapi.json (|.yaml)

デシジョンロジックのない DMN XML を返します。

GET /server/containers/{containerId}/dmn/models/{modelname}

REST エンドポイントの例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation

curl 要求例

```
curl -u wbadmin:wbadmin -X GET "http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic%20Violation" -H "accept: application/xml"
```

応答例 (XML)

```
<?xml version='1.0' encoding='UTF-8'?>
<dmn:definitions xmlns:dmn="http://www.omg.org/spec/DMN/20180521/MODEL/"
```

```
xmlns="https://github.com/kiegroup/drools/kie-dmn/_A4BCA8B8-CF08-433F-93B2-
A2598F19ECFF" xmlns:di="http://www.omg.org/spec/DMN/20180521/DI/"
xmlns:kie="http://www.drools.org/kie/dmn/1.2"
xmlns:feel="http://www.omg.org/spec/DMN/20180521/FEEL/"
xmlns:dmndi="http://www.omg.org/spec/DMN/20180521/DMNDI/"
xmlns:dc="http://www.omg.org/spec/DMN/20180521/DC/" id="_1C792953-80DB-4B32-99EB-
25FBE32BAF9E" name="Traffic Violation"
expressionLanguage="http://www.omg.org/spec/DMN/20180521/FEEL/"
typeLanguage="http://www.omg.org/spec/DMN/20180521/FEEL/"
namespace="https://github.com/kiegroup/drools/kie-dmn/_A4BCA8B8-CF08-433F-93B2-
A2598F19ECFF">
  <dmn:extensionElements/>
  <dmn:itemDefinition id="_63824D3F-9173-446D-A940-6A7F0FA056BB" name="tDriver"
isCollection="false">
    <dmn:itemComponent id="_9DAB5DAA-3B44-4F6D-87F2-95125FB2FEE4" name="Name"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_856BA8FA-EF7B-4DF9-A1EE-E28263CE9955" name="Age"
isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_FDC2CE03-D465-47C2-A311-98944E8CC23F" name="State"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_D6FD34C4-00DC-4C79-B1BF-BBCF6FC9B6D7" name="City"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_7110FE7E-1A38-4C39-B0EB-AEEF06BA37F4" name="Points"
isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
  </dmn:itemDefinition>
  <dmn:itemDefinition id="_40731093-0642-4588-9183-1660FC55053B" name="tViolation"
isCollection="false">
    <dmn:itemComponent id="_39E88D9F-AE53-47AD-B3DE-8AB38D4F50B3" name="Code"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_1648EA0A-2463-4B54-A12A-D743A3E3EE7B" name="Date"
isCollection="false">
      <dmn:typeRef>date</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_9F129EAA-4E71-4D99-B6D0-84EEC3AC43CC" name="Type"
isCollection="false">
      <dmn:typeRef>string</dmn:typeRef>
      <dmn:allowedValues kie:constraintType="enumeration" id="_626A8F9C-9DD1-44E0-9568-
0F6F8F8BA228">
        <dmn:text>"speed", "parking", "driving under the influence"</dmn:text>
      </dmn:allowedValues>
    </dmn:itemComponent>
    <dmn:itemComponent id="_DDD10D6E-BD38-4C79-9E2F-8155E3A4B438" name="Speed
Limit" isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
```

```
    </dmn:itemComponent>
    <dmn:itemComponent id="_229F80E4-2892-494C-B70D-683ABF2345F6" name="Actual
Speed" isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
  </dmn:itemDefinition>
  <dmn:itemDefinition id="_2D4F30EE-21A6-4A78-A524-A5C238D433AE" name="tFine"
isCollection="false">
    <dmn:itemComponent id="_B9F70BC7-1995-4F51-B949-1AB65538B405" name="Amount"
isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
    <dmn:itemComponent id="_F49085D6-8F08-4463-9A1A-EF6B57635DBD" name="Points"
isCollection="false">
      <dmn:typeRef>number</dmn:typeRef>
    </dmn:itemComponent>
  </dmn:itemDefinition>
  <dmn:inputData id="_1929CBD5-40E0-442D-B909-49CEDE0101DC" name="Violation">
    <dmn:variable id="_C16CF9B1-5FAB-48A0-95E0-5FCD661E0406" name="Violation"
typeRef="tViolation"/>
  </dmn:inputData>
  <dmn:decision id="_4055D956-1C47-479C-B3F4-BAEB61F1C929" name="Fine">
    <dmn:variable id="_8C1EAC83-F251-4D94-8A9E-B03ACF6849CD" name="Fine"
typeRef="tFine"/>
    <dmn:informationRequirement id="_800A3BBB-90A3-4D9D-BA5E-A311DED0134F">
      <dmn:requiredInput href="#_1929CBD5-40E0-442D-B909-49CEDE0101DC"/>
    </dmn:informationRequirement>
  </dmn:decision>
  <dmn:inputData id="_1F9350D7-146D-46F1-85D8-15B5B68AF22A" name="Driver">
    <dmn:variable id="_A80F16DF-0DB4-43A2-B041-32900B1A3F3D" name="Driver"
typeRef="tDriver"/>
  </dmn:inputData>
  <dmn:decision id="_8A408366-D8E9-4626-ABF3-5F69AA01F880" name="Should the driver be
suspended?">
    <dmn:question>Should the driver be suspended due to points on his license?</dmn:question>
    <dmn:allowedAnswers>"Yes", "No"</dmn:allowedAnswers>
    <dmn:variable id="_40387B66-5D00-48C8-BB90-E83EE3332C72" name="Should the driver be
suspended?" typeRef="string"/>
    <dmn:informationRequirement id="_982211B1-5246-49CD-BE85-3211F71253CF">
      <dmn:requiredInput href="#_1F9350D7-146D-46F1-85D8-15B5B68AF22A"/>
    </dmn:informationRequirement>
    <dmn:informationRequirement id="_AEC4AA5F-50C3-4FED-A0C2-261F90290731">
      <dmn:requiredDecision href="#_4055D956-1C47-479C-B3F4-BAEB61F1C929"/>
    </dmn:informationRequirement>
  </dmn:decision>
  <dmndi:DMNDI>
    <dmndi:DMNDiagram>
      <di:extension/>
      <dmndi:DMNShape id="dmnshape-_1929CBD5-40E0-442D-B909-49CEDE0101DC"
dmnElementRef="_1929CBD5-40E0-442D-B909-49CEDE0101DC" isCollapsed="false">
        <dmndi:DMNStyle>
          <dmndi:FillColor red="255" green="255" blue="255"/>
          <dmndi:StrokeColor red="0" green="0" blue="0"/>
          <dmndi:FontColor red="0" green="0" blue="0"/>
        </dmndi:DMNStyle>
        <dc:Bounds x="708" y="350" width="100" height="50"/>
```

```
        <dmndi:DMNLabel/>
      </dmndi:DMNShape>
      <dmndi:DMNShape id="dmnshape-_4055D956-1C47-479C-B3F4-BAEB61F1C929"
dmnElementRef="_4055D956-1C47-479C-B3F4-BAEB61F1C929" isCollapsed="false">
        <dmndi:DMNStyle>
          <dmndi:FillColor red="255" green="255" blue="255"/>
          <dmndi:StrokeColor red="0" green="0" blue="0"/>
          <dmndi:FontColor red="0" green="0" blue="0"/>
        </dmndi:DMNStyle>
        <dc:Bounds x="709" y="210" width="100" height="50"/>
        <dmndi:DMNLabel/>
      </dmndi:DMNShape>
      <dmndi:DMNShape id="dmnshape-_1F9350D7-146D-46F1-85D8-15B5B68AF22A"
dmnElementRef="_1F9350D7-146D-46F1-85D8-15B5B68AF22A" isCollapsed="false">
        <dmndi:DMNStyle>
          <dmndi:FillColor red="255" green="255" blue="255"/>
          <dmndi:StrokeColor red="0" green="0" blue="0"/>
          <dmndi:FontColor red="0" green="0" blue="0"/>
        </dmndi:DMNStyle>
        <dc:Bounds x="369" y="344" width="100" height="50"/>
        <dmndi:DMNLabel/>
      </dmndi:DMNShape>
      <dmndi:DMNShape id="dmnshape-_8A408366-D8E9-4626-ABF3-5F69AA01F880"
dmnElementRef="_8A408366-D8E9-4626-ABF3-5F69AA01F880" isCollapsed="false">
        <dmndi:DMNStyle>
          <dmndi:FillColor red="255" green="255" blue="255"/>
          <dmndi:StrokeColor red="0" green="0" blue="0"/>
          <dmndi:FontColor red="0" green="0" blue="0"/>
        </dmndi:DMNStyle>
        <dc:Bounds x="534" y="83" width="133" height="63"/>
        <dmndi:DMNLabel/>
      </dmndi:DMNShape>
      <dmndi:DMNEdge id="dmnedge-_800A3BBB-90A3-4D9D-BA5E-A311DED0134F"
dmnElementRef="_800A3BBB-90A3-4D9D-BA5E-A311DED0134F">
        <di:waypoint x="758" y="375"/>
        <di:waypoint x="759" y="235"/>
      </dmndi:DMNEdge>
      <dmndi:DMNEdge id="dmnedge-_982211B1-5246-49CD-BE85-3211F71253CF"
dmnElementRef="_982211B1-5246-49CD-BE85-3211F71253CF">
        <di:waypoint x="419" y="369"/>
        <di:waypoint x="600.5" y="114.5"/>
      </dmndi:DMNEdge>
      <dmndi:DMNEdge id="dmnedge-_AEC4AA5F-50C3-4FED-A0C2-261F90290731"
dmnElementRef="_AEC4AA5F-50C3-4FED-A0C2-261F90290731">
        <di:waypoint x="759" y="235"/>
        <di:waypoint x="600.5" y="114.5"/>
      </dmndi:DMNEdge>
    </dmndi:DMNDiagram>
  </dmndi:DMNDI>
```

指定されたコンテナーで指定の DMN モデルを評価します

POST /server/containers/{containerId}/dmn/models/{modelname}

REST エンドポイントの例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation

curl 要求例

```
curl -u wbadmin:wbadmin-X POST "http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation" -H  "accept: application/json" -H  "Content-Type: application/json" -d "{\"Driver\":{\"Points\":15},\"Violation\":{\"Date\":\"2021-04-08\",\"Type\":\"speed\",\"Actual Speed\":135,\"Speed Limit\":100}}"
```

入力データでの POST 要求ボディーの例

```
{
  "Driver": {
    "Points": 15
  },
  "Violation": {
    "Date": "2021-04-08",
    "Type": "speed",
    "Actual Speed": 135,
    "Speed Limit": 100
  }
}
```

応答例 (JSON)

```
{
  "Violation": {
    "Type": "speed",
    "Speed Limit": 100,
    "Actual Speed": 135,
    "Code": null,
    "Date": "2021-04-08"
  },
  "Driver": {
    "Points": 15,
    "State": null,
    "City": null,
    "Age": null,
    "Name": null
  },
  "Fine": {
    "Points": 7,
    "Amount": 1000
  },
  "Should the driver be suspended?": "Yes"
}
```

コンテナーで指定された DMN モデル内の指定のデシジョンサービスを評価します

POST /server/containers/{containerId}/dmn/models/{modelname}/{decisionServiceName}
このエンドポイントでは、要求ボディーにデシジョンサービスのすべての要件を含める必要があります。応答は、デシジョン値、元の入力値、およびシリアル化形式の他の parametric DRG コンポーネントすべてなど、デシジョンサービスの結果として返される DMN コンテキストです。たとえば、ビジネスナレッジモデルは、署名の文字列のシリアル化形式で利用できます。

デシジョンサービスが単一の出力デシジョンで設定される場合には、応答はその特定のデシジョンから返される値になります。この動作は、モデル自体でデシジョンサービスを呼び出す時に、仕様機能の API レベルで同等の値を指定します。これにより、たとえば単一ページの Web アプリケーションから DMN デシジョンサービスと対話ができます。

図21.1 単一の出力デシジョンを含む TrafficViolationDecisionService デシジョンサービスの例

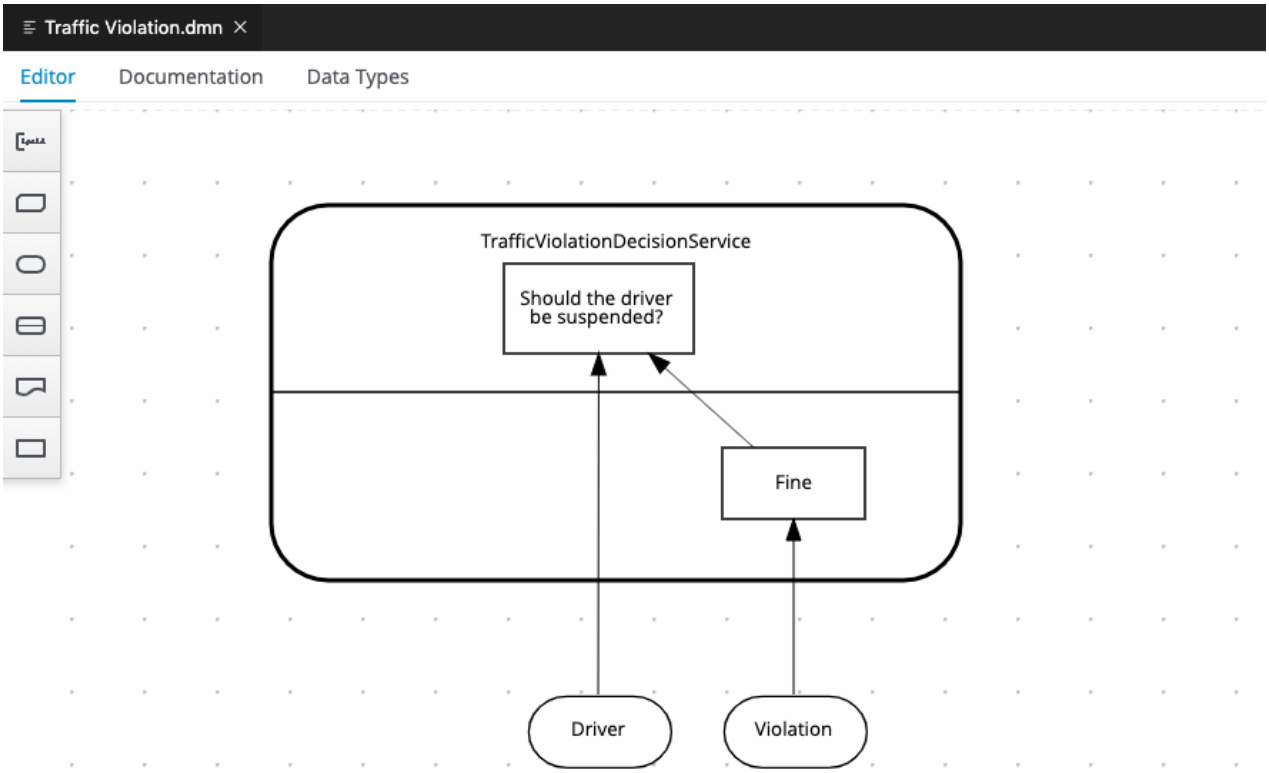
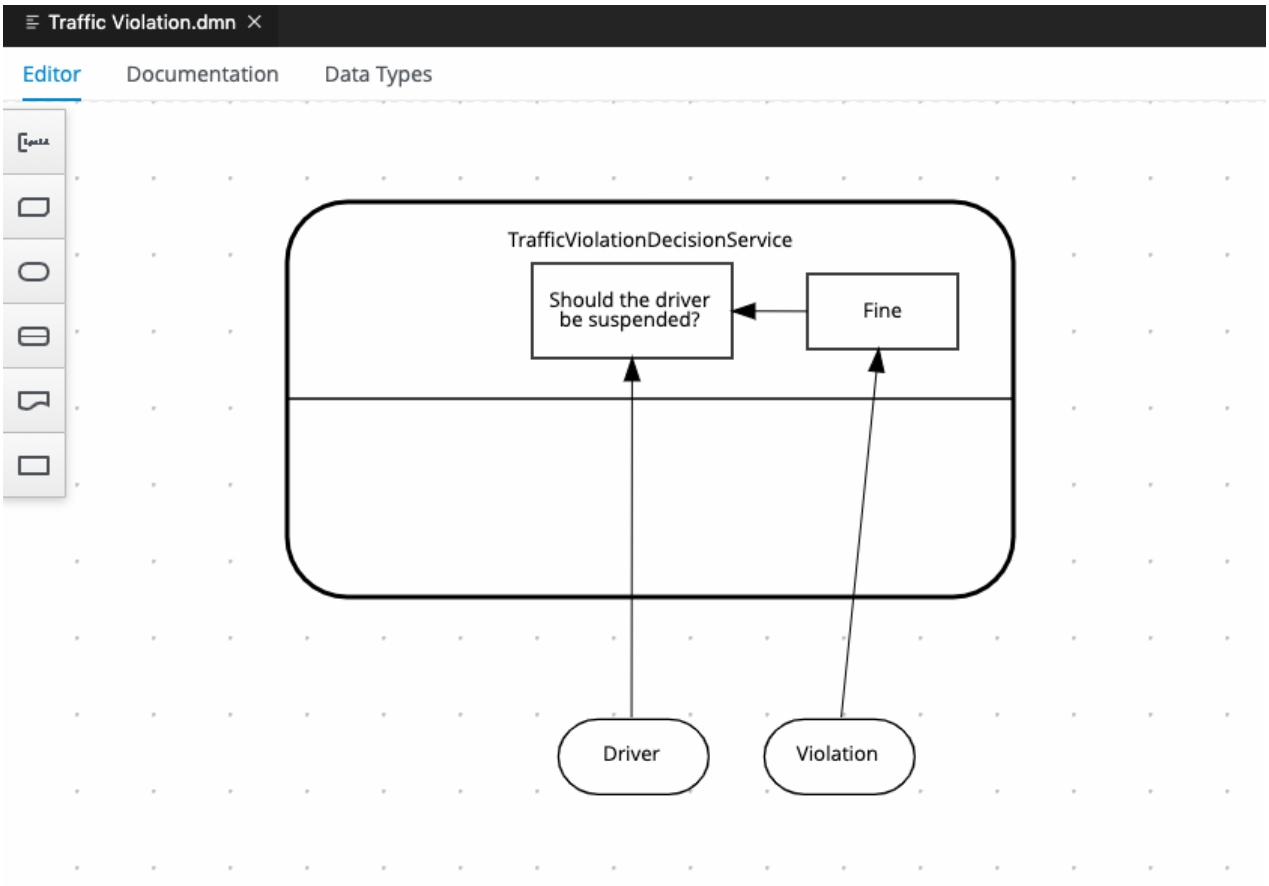


図21.2 複数の出力デシジョンを含む TrafficViolationDecisionService デシジョンサービスの例



REST エンドポイントの例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation/TrafficViolationDecisionService

入力データでの POST 要求ボディーの例

```
{
  "Driver": {
    "Points": 2
  },
  "Violation": {
    "Type": "speed",
    "Actual Speed": 120,
    "Speed Limit": 100
  }
}
```

curl 要求例

```
curl -X POST http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation/TrafficViolationDecisionService -H 'content-type: application/json' -H 'accept: application/json' -d '{"Driver": {"Points": 2}, "Violation": {"Type": "speed", "Actual Speed": 120, "Speed Limit": 100}}'
```

シングル出力デシジョンの応答例 (JSON)

```
"No"
```

-

複数出力デシジョンの応答例 (JSON)

```
{
  "Violation": {
    "Type": "speed",
    "Speed Limit": 100,
    "Actual Speed": 120
  },
  "Driver": {
    "Points": 2
  },
  "Fine": {
    "Points": 3,
    "Amount": 500
  },
  "Should the driver be suspended?": "No"
}
```

指定したコンテナーで指定の DMN モデルを評価し、DMNResult 応答を返します。

POST /server/containers/{containerId}/dmn/models/{modelname}/dmnresult

REST エンドポイントの例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation/dmnresult

入力データでの POST 要求ボディーの例

```
{
  "Driver": {
    "Points": 2
  },
  "Violation": {
    "Type": "speed",
    "Actual Speed": 120,
    "Speed Limit": 100
  }
}
```

curl 要求例

```
curl -X POST http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation/dmnresult -H 'content-type: application/json' -H 'accept: application/json' -d '{"Driver": {"Points": 2}, "Violation": {"Type": "speed", "Actual Speed": 120, "Speed Limit": 100}}'
```

応答例 (JSON)

```
{
  "namespace": "https://github.com/kiegroup/drools/kie-dmn/_A4BCA8B8-CF08-433F-93B2-A2598F19ECFF",
  "modelName": "Traffic Violation",
  "dmnContext": {
```

```
    "Violation": {
      "Type": "speed",
      "Speed Limit": 100,
      "Actual Speed": 120,
      "Code": null,
      "Date": null
    },
    "Driver": {
      "Points": 2,
      "State": null,
      "City": null,
      "Age": null,
      "Name": null
    },
    "Fine": {
      "Points": 3,
      "Amount": 500
    },
    "Should the driver be suspended?": "No"
  },
  "messages": [],
  "decisionResults": [
    {
      "decisionId": "_4055D956-1C47-479C-B3F4-BAEB61F1C929",
      "decisionName": "Fine",
      "result": {
        "Points": 3,
        "Amount": 500
      },
      "messages": [],
      "evaluationStatus": "SUCCEEDED"
    },
    {
      "decisionId": "_8A408366-D8E9-4626-ABF3-5F69AA01F880",
      "decisionName": "Should the driver be suspended?",
      "result": "No",
      "messages": [],
      "evaluationStatus": "SUCCEEDED"
    }
  ]
}
```


指定したコンテナーの DMN モデル内で指定のデシジョンサービスを評価し、DMNResult 応答を返します。

POST
/server/containers/{containerId}/dmn/models/{modelname}/{decisionServiceName}/dmnresult

REST エンドポイントの例

http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation/TrafficViolationDecisionService/dmnresult

入力データでの POST 要求ボディーの例


```
{
```

```
  "Driver": {
    "Points": 2
  },
  "Violation": {
    "Type": "speed",
    "Actual Speed": 120,
    "Speed Limit": 100
  }
}
```

curl 要求例

```
curl -X POST http://localhost:8080/kie-server/services/rest/server/containers/mykjar-project/dmn/models/Traffic Violation/TrafficViolationDecisionService/dmnresult -H 'content-type: application/json' -H 'accept: application/json' -d '{"Driver": {"Points": 2}, "Violation": {"Type": "speed", "Actual Speed": 120, "Speed Limit": 100}}'
```

応答例 (JSON)

```
{
  "namespace": "https://github.com/kiegroup/drools/kie-dmn/_A4BCA8B8-CF08-433F-93B2-A2598F19ECFF",
  "modelName": "Traffic Violation",
  "dmnContext": {
    "Violation": {
      "Type": "speed",
      "Speed Limit": 100,
      "Actual Speed": 120,
      "Code": null,
      "Date": null
    },
    "Driver": {
      "Points": 2,
      "State": null,
      "City": null,
      "Age": null,
      "Name": null
    },
    "Should the driver be suspended?": "No"
  },
  "messages": [],
  "decisionResults": [
    {
      "decisionId": "_8A408366-D8E9-4626-ABF3-5F69AA01F880",
      "decisionName": "Should the driver be suspended?",
      "result": "No",
      "messages": [],
      "evaluationStatus": "SUCCEEDED"
    }
  ]
}
```

第22章 KIE コンテナーおよびビジネスアセット用の KIE SERVER JAVA クライアント API

Red Hat Process Automation Manager は KIE Server Java クライアント API を提供し、これを使用することで Java クライアントアプリケーションから REST プロトコルを使用して KIE Server に接続できるようになります。KIE Server REST API の代わりに KIE Server Java クライアント API を使用して、Business Central ユーザーインターフェイスを使わずに Red Hat Process Automation Manager の KIE コンテナーやビジネスアセット (ビジネスルール、プロセス、ソルバーなど) を操作することができます。この API のサポートにより、Red Hat Process Automation Manager のリソースをより効率的に維持でき、Red Hat Process Automation Manager の統合と開発を最適化できるようになります。

KIE Server Java クライアント API を使用すると、KIE Server REST API でもサポートされている以下のアクションが実行可能になります。

- KIE コンテナーのデプロイまたは破棄
- KIE コンテナー情報の取得および更新
- KIE Server ステータスおよび基本的情報の確認
- ビジネスアセット情報の取得および更新
- ビジネスアセット (ルールやプロセスなど) の実行

KIE Server Java クライアント API 要求には以下のコンポーネントが必要です。

認証

KIE Server Java クライアント API は、ユーザーロール **kie-server** に HTTP の Basic 認証を必要とします。お使いの Red Hat Process Automation Manager に設定されているユーザーロールを表示するには、**~/$SERVER_HOME/standalone/configuration/application-roles.properties** と **~/application-users.properties** に移動します。
ユーザーに **kie-server** ロールを追加するには、**~/$SERVER_HOME/bin** に移動して以下のコマンドを実行します。

```
$ ./bin/jboss-cli.sh --commands="embed-server --std-out=echo,/subsystem=elytron/filesystem-realm=ApplicationRealm:add-identity(identity=<USERNAME>),/subsystem=elytron/filesystem-realm=ApplicationRealm:set-password(identity=<USERNAME>, clear={password='<PASSWORD>'}),/subsystem=elytron/filesystem-realm=ApplicationRealm:add-identity-attribute(identity=<USERNAME>, name=role, value=['kie-server'])"
```

ユーザーロールと Red Hat Process Automation Manager のインストールオプションの詳細は、**Red Hat Process Automation Manager インストールの計画** を参照してください。

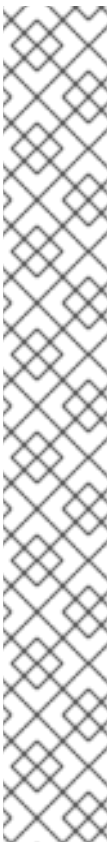
プロジェクトの依存関係

KIE Server Java クライアント API には、Java プロジェクト内の適切なクラスパスに、以下の依存関係が必要です。

```
<!-- For remote execution on KIE Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>
```

```
<!-- For runtime commands -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-compiler</artifactId>
  <scope>runtime</scope>
  <version>${rhpam.version}</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>${logback.version}</version>
</dependency>
```

Red Hat Process Automation Manager 依存関係の **<version>** は、プロジェクトで現在使用している Red Hat Process Automation Manager の Maven アーティファクトバージョンです (例: 7.59.0.Final-redhat-00006)。



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 BOM (bill of materials) 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 Decision Manager と Red Hat Process Automation Manager の両方に適用されます。BOM ファイルを追加すると、提供される Maven リポジトリーから、推移的依存関係の適切なバージョンがプロジェクトに含められます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.12.0.redhat-00008</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。

クライアント要求の設定

KIE Server Java クライアント API による Java クライアント要求はすべて、少なくとも以下のサーバー通信コンポーネントを定義する必要があります。

- **kie-server** ユーザーの認証情報
- KIE Server の場所 (例: **http://localhost:8080/kie-server/services/rest/server**)
- API 要求および応答のマーシャリングフォーマット (JSON、JAXB、または XSTREAM)

- **KieServicesConfiguration** オブジェクトおよび **KieServicesClient** オブジェクト (Java クライアント API を使用してサーバー通信を開始するためのエントリーポイントのロールを果たします)
- REST プロトコルおよびユーザーアクセスを定義する **KieServicesFactory** オブジェクト
- 使用される他のクライアントサービス (**RuleServicesClient**、**ProcessServicesClient**、**QueryServicesClient** など)

以下は、これらのコンポーネントを使用した基本的および高度なクライアント設定の例です。

基本的クライアント設定の例

```
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;

public class MyConfigurationObject {

  private static final String URL = "http://localhost:8080/kie-server/services/rest/server";
  private static final String USER = "baAdmin";
  private static final String PASSWORD = "password@1";

  private static final MarshallingFormat FORMAT = MarshallingFormat.JSON;

  private static KieServicesConfiguration conf;
  private static KieServicesClient kieServicesClient;

  public static void initialize() {
    conf = KieServicesFactory.newRestConfiguration(URL, USER, PASSWORD);

    //If you use custom classes, such as Obj.class, add them to the configuration.
    Set<Class<?>> extraClassList = new HashSet<Class<?>>();
    extraClassList.add(Obj.class);
    conf.addExtraClasses(extraClassList);

    conf.setMarshallingFormat(FORMAT);
    kieServicesClient = KieServicesFactory.newKieServicesClient(conf);
  }
}
```

追加のクライアントサービスを使用した高度なクライアント設定の例

```
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.client.CaseServicesClient;
import org.kie.server.client.DMNServicesClient;
import org.kie.server.client.DocumentServicesClient;
import org.kie.server.client.JobServicesClient;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.ProcessServicesClient;
import org.kie.server.client.QueryServicesClient;
import org.kie.server.client.RuleServicesClient;
import org.kie.server.client.SolverServicesClient;
```

```
import org.kie.server.client.UIServicesClient;
import org.kie.server.client.UserTaskServicesClient;
import org.kie.server.api.model.instance.ProcessInstance;
import org.kie.server.api.model.KieContainerResource;
import org.kie.server.api.model.ReleaseId;

public class MyAdvancedConfigurationObject {

    // REST API base URL, credentials, and marshalling format
    private static final String URL = "http://localhost:8080/kie-server/services/rest/server";
    private static final String USER = "baAdmin";
    private static final String PASSWORD = "password@1";;

    private static final MarshallingFormat FORMAT = MarshallingFormat.JSON;

    private static KieServicesConfiguration conf;

    // KIE client for common operations
    private static KieServicesClient kieServicesClient;

    // Rules client
    private static RuleServicesClient ruleClient;

    // Process automation clients
    private static CaseServicesClient caseClient;
    private static DocumentServicesClient documentClient;
    private static JobServicesClient jobClient;
    private static ProcessServicesClient processClient;
    private static QueryServicesClient queryClient;
    private static UIServicesClient uiClient;
    private static UserTaskServicesClient userTaskClient;

    // DMN client
    private static DMNServicesClient dmnClient;

    // Planning client
    private static SolverServicesClient solverClient;

    public static void main(String[] args) {
        initializeKieServerClient();
        initializeDroolsServiceClients();
        initializeJbpmServiceClients();
        initializeSolverServiceClients();
    }

    public static void initializeKieServerClient() {
        conf = KieServicesFactory.newRestConfiguration(URL, USER, PASSWORD);
        conf.setMarshallingFormat(FORMAT);
        kieServicesClient = KieServicesFactory.newKieServicesClient(conf);
    }

    public static void initializeDroolsServiceClients() {
        ruleClient = kieServicesClient.getServicesClient(RuleServicesClient.class);
        dmnClient = kieServicesClient.getServicesClient(DMNServicesClient.class);
    }
```

```
    public static void initializeJbpmServiceClients() {
        caseClient = kieServicesClient.getServicesClient(CaseServicesClient.class);
        documentClient = kieServicesClient.getServicesClient(DocumentServicesClient.class);
        jobClient = kieServicesClient.getServicesClient(JobServicesClient.class);
        processClient = kieServicesClient.getServicesClient(ProcessServicesClient.class);
        queryClient = kieServicesClient.getServicesClient(QueryServicesClient.class);
        uiClient = kieServicesClient.getServicesClient(UIServicesClient.class);
        userTaskClient = kieServicesClient.getServicesClient(UserTaskServicesClient.class);
    }

    public static void initializeSolverServiceClients() {
        solverClient = kieServicesClient.getServicesClient(SolverServicesClient.class);
    }
}
```

22.1. KIE SERVER JAVA クライアント API を使用した要求送信

KIE Server Java クライアント API を使用すると、Java クライアントアプリケーションから REST プロトコルを使用して KIE Server に接続できるようになります。KIE Server REST API の代わりに KIE Server Java クライアント API を使用して、Business Central ユーザーインターフェイスを使わずに Red Hat Process Automation Manager の KIE コンテナーやビジネスアセット (ビジネスルール、プロセス、ソルバーなど) を操作することができます。

前提条件

- KIE Server をインストールし、実行している。
- **kie-server** ユーザーロールで KIE Server にアクセスできる。
- Red Hat Process Automation Manager リソースを使った Java プロジェクトがある。

手順

1. クライアントアプリケーションで、Java プロジェクトの関連クラスパスに以下の依存関係が追加されていることを確認します。

```
<!-- For remote execution on KIE Server -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- For runtime commands -->
<dependency>
  <groupId>org.drools</groupId>
  <artifactId>drools-compiler</artifactId>
  <scope>runtime</scope>
  <version>${rhpam.version}</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>ch.qos.logback</groupId>
```

```
  <artifactId>logback-classic</artifactId>
  <version>${logback.version}</version>
</dependency>
```

2. Red Hat カスタマーポータル から **Red Hat Process Automation Manager 7.12.0 Source Distribution** をダウンロードし、**~/rhpam-7.12.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-remote/kie-server-client/src/main/java/org/kie/server/client** に移動して KIE Server Java クライアントにアクセスします。

3. **~/kie/server/client** ディレクトリーで、KIE Server の KIE コンテナーや他のアセット用のクライアントサービスにアクセスするために、**KieServicesClient** などの送信する要求用の関連 Java クライアントを特定します。

4. クライアントアプリケーションで、API 要求用の **.java** クラスを作成します。クラスには、KIE Server の場所とユーザー認証情報、**KieServicesClient** オブジェクト、**KieServicesClient** クライアントからの **createContainer** や **disposeContainer** などの実行するクライアントメソッドを含める必要があります。ご自分のユースケースに合わせて、設定詳細を調整します。

コンテナーの作成および破棄

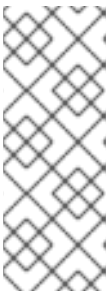
```
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.api.model.KieContainerResource;
import org.kie.server.api.model.ServiceResponse;

public class MyConfigurationObject {

  private static final String URL = "http://localhost:8080/kie-server/services/rest/server";
  private static final String USER = "baAdmin";
  private static final String PASSWORD = "password@1";

  private static final MarshallingFormat FORMAT = MarshallingFormat.JSON;

  private static KieServicesConfiguration conf;
  private static KieServicesClient kieServicesClient;

  public static void initialize() {
    conf = KieServicesFactory.newRestConfiguration(URL, USER, PASSWORD);

  public void disposeAndCreateContainer() {
     System.out.println("== Disposing and creating containers ==");

     // Retrieve list of KIE containers
     List<KieContainerResource> kieContainers =
kieServicesClient.listContainers().getResult().getContainers();
     if (kieContainers.size() == 0) {
        System.out.println("No containers available...");
        return;
     }

     // Dispose KIE container
     KieContainerResource container = kieContainers.get(0);
     String containerId = container.getContainerId();
```

```
        ServiceResponse<Void> responseDispose =
kieServicesClient.disposeContainer(containerId);
        if (responseDispose.getType() == ResponseType.FAILURE) {
            System.out.println("Error disposing " + containerId + ". Message: ");
            System.out.println(responseDispose.getMsg());
            return;
        }
        System.out.println("Success Disposing container " + containerId);
        System.out.println("Trying to recreate the container...");

        // Re-create KIE container
        ServiceResponse<KieContainerResource> createResponse =
kieServicesClient.createContainer(containerId, container);
        if(createResponse.getType() == ResponseType.FAILURE) {
            System.out.println("Error creating " + containerId + ". Message: ");
            System.out.println(responseDispose.getMsg());
            return;
        }
        System.out.println("Container recreated with success!");
        }
    }
}
```

org.kie.server.api.model.ServiceResponse<T> オブジェクトを使用してサービス応答を定義します。ここでの **T** は、返される応答のタイプを表します。**ServiceResponse** オブジェクトには以下の属性があります。

- **String message**: 応答メッセージを返します。
- **ResponseType type**: **SUCCESS** または **FAILURE** を返します。
- **T result**: 要求されたオブジェクトを返します。

この例では、コンテナーを破棄すると、**ServiceResponse** が **Void** 応答を返します。コンテナーを作成すると、**ServiceResponse** が **KieContainerResource** オブジェクトを返します。



注記

クライアントとクラスター環境内の特定の KIE Server コンテナーとの対話は、一意の **conversationID** でセキュリティーが保たれます。**conversationID** は **X-KIE-ConversationId** REST ヘッダーを使用して送信されます。コンテナーを更新する場合は、以前の **conversationID** の設定を解除します。**KieServiesClient.completeConversation()** を使用して Java API の **conversationID** を設定解除します。

5. 設定済みの **.java** クラスをプロジェクトディレクトリーから実行して、要求を出し、KIE Server の応答を確認します。
デバッグのロギングが有効になっている場合は、JSON などの設定済みマーシャリングフォーマットに従って、KIE Server が詳細を返します。

   新規 KIE コンテナーのサーバー応答の例 (ログ):

```
10:23:35.194 [main] INFO  o.k.s.a.m.MarshallerFactory - Marshaller extensions init
10:23:35.396 [main] DEBUG o.k.s.client.balancer.LoadBalancer - Load balancer
RoundRobinBalancerStrategy{availableEndpoints=[http://localhost:8080/kie-
```

```
server/services/rest/server]} selected url 'http://localhost:8080/kie-server/services/rest/server'
10:23:35.398 [main] DEBUG o.k.s.c.i.AbstractKieServicesClientImpl - About to send GET
request to 'http://localhost:8080/kie-server/services/rest/server'
10:23:35.440 [main] DEBUG o.k.s.c.i.AbstractKieServicesClientImpl - About to deserialize
content:
 '{
  "type" : "SUCCESS",
  "msg" : "Kie Server info",
  "result" : {
    "kie-server-info" : {
      "id" : "default-kieserver",
      "version" : "7.11.0.Final-redhat-00003",
      "name" : "default-kieserver",
      "location" : "http://localhost:8080/kie-server/services/rest/server",
      "capabilities" : [ "KieServer", "BRM", "BPM", "CaseMgmt", "BPM-UI", "BRP", "DMN",
"Swagger" ],
      "messages" : [ {
        "severity" : "INFO",
        "timestamp" : {
  "java.util.Date" : 1540814906533
},
        "content" : [ "Server KieServerInfo{serverId='default-kieserver', version='7.11.0.Final-
redhat-00003', name='default-kieserver', location='http://localhost:8080/kie-
server/services/rest/server', capabilities=[KieServer, BRM, BPM, CaseMgmt, BPM-UI, BRP,
DMN, Swagger], messages=null}started successfully at Mon Oct 29 08:08:26 EDT 2018" ]
      } ]
    }
  }
}'
 into type: 'class org.kie.server.api.model.ServiceResponse'
10:23:35.653 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - KieServicesClient
connected to: default-kieserver version 7.11.0.Final-redhat-00003
10:23:35.653 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Supported capabilities by
the server: [KieServer, BRM, BPM, CaseMgmt, BPM-UI, BRP, DMN, Swagger]
10:23:35.653 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Building services client for
server capability KieServer
10:23:35.653 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - No builder found for
'KieServer' capability
10:23:35.654 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Building services client for
server capability BRM
10:23:35.654 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Builder
'org.kie.server.client.helper.DroolsServicesClientBuilder@6b927fb' for capability 'BRM'
10:23:35.655 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Capability implemented by
{interface
org.kie.server.client.RuleServicesClient=org.kie.server.client.impl.RuleServicesClientImpl@4a94
ee4}
10:23:35.655 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Building services client for
server capability BPM
10:23:35.656 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Builder
'org.kie.server.client.helper.JBPMServicesClientBuilder@4cc451f2' for capability 'BPM'
10:23:35.672 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Capability implemented by
{interface
org.kie.server.client.JobServicesClient=org.kie.server.client.impl.JobServicesClientImpl@1189d
52, interface
org.kie.server.client.admin.ProcessAdminServicesClient=org.kie.server.client.admin.impl.Proces
sAdminServicesClientImpl@36bc55de, interface
```

```
org.kie.server.client.DocumentServicesClient=org.kie.server.client.impl.DocumentServicesClient
mpl@564fabc8, interface
org.kie.server.client.admin.UserTaskAdminServicesClient=org.kie.server.client.admin.impl.User
TaskAdminServicesClientImpl@16d04d3d, interface
org.kie.server.client.QueryServicesClient=org.kie.server.client.impl.QueryServicesClientImpl@4
9ec71f8, interface
org.kie.server.client.ProcessServicesClient=org.kie.server.client.impl.ProcessServicesClientImp
@1d2adfbe, interface
org.kie.server.client.UserTaskServicesClient=org.kie.server.client.impl.UserTaskServicesClientI
mpl@36902638}
10:23:35.672 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Building services client for
server capability CaseMgmt
10:23:35.672 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Builder
'org.kie.server.client.helper.CaseServicesClientBuilder@223d2c72' for capability 'CaseMgmt'
10:23:35.676 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Capability implemented by
{interface
org.kie.server.client.admin.CaseAdminServicesClient=org.kie.server.client.admin.impl.CaseAdm
nServicesClientImpl@2b662a77, interface
org.kie.server.client.CaseServicesClient=org.kie.server.client.impl.CaseServicesClientImpl@7f0
eb4b4}
10:23:35.676 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Building services client for
server capability BPM-UI
10:23:35.676 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Builder
'org.kie.server.client.helper.JBPMUIServicesClientBuilder@5c33f1a9' for capability 'BPM-UI'
10:23:35.677 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Capability implemented by
{interface
org.kie.server.client.UIServicesClient=org.kie.server.client.impl.UIServicesClientImpl@223191a
}
10:23:35.678 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Building services client for
server capability BRP
10:23:35.678 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Builder
'org.kie.server.client.helper.OptaplannerServicesClientBuilder@49139829' for capability
'BRP'
10:23:35.679 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Capability implemented by
{interface
org.kie.server.client.SolverServicesClient=org.kie.server.client.impl.SolverServicesClientImpl@7
7fbd92c}
10:23:35.679 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Building services client for
server capability DMN
10:23:35.679 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Builder
'org.kie.server.client.helper.DMNServicesClientBuilder@67c27493' for capability 'DMN'
10:23:35.680 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Capability implemented by
{interface
org.kie.server.client.DMNServicesClient=org.kie.server.client.impl.DMNServicesClientImpl@35e
2d654}
10:23:35.680 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - Building services client for
server capability Swagger
10:23:35.680 [main] DEBUG o.k.s.c.impl.KieServicesClientImpl - No builder found for
'Swagger' capability
10:23:35.681 [main] DEBUG o.k.s.client.balancer.LoadBalancer - Load balancer
RoundRobinBalancerStrategy{availableEndpoints=[http://localhost:8080/kie-
server/services/rest/server]} selected url 'http://localhost:8080/kie-server/services/rest/server'
10:23:35.701 [main] DEBUG o.k.s.c.i.AbstractKieServicesClientImpl - About to send PUT
request to 'http://localhost:8080/kie-server/services/rest/server/containers/employee-
rostering3' with payload '{
  "container-id" : null,
```

```
  "release-id" : {
    "group-id" : "employeerostering",
    "artifact-id" : "employeerostering",
    "version" : "1.0.0-SNAPSHOT"
  },
  "resolved-release-id" : null,
  "status" : null,
  "scanner" : null,
  "config-items" : [ ],
  "messages" : [ ],
  "container-alias" : null
}'
10:23:38.071 [main] DEBUG o.k.s.c.i.AbstractKieServicesClientImpl - About to deserialize
content:
 '{
  "type" : "SUCCESS",
  "msg" : "Container employee-rostering3 successfully deployed with module
employeerostering:employeerostering:1.0.0-SNAPSHOT.",
  "result" : {
    "kie-container" : {
      "container-id" : "employee-rostering3",
      "release-id" : {
        "group-id" : "employeerostering",
        "artifact-id" : "employeerostering",
        "version" : "1.0.0-SNAPSHOT"
      },
      "resolved-release-id" : {
        "group-id" : "employeerostering",
        "artifact-id" : "employeerostering",
        "version" : "1.0.0-SNAPSHOT"
      },
      "status" : "STARTED",
      "scanner" : {
        "status" : "DISPOSED",
        "poll-interval" : null
      },
      "config-items" : [ ],
      "messages" : [ {
        "severity" : "INFO",
        "timestamp" : {
  "java.util.Date" : 1540909418069
},
        "content" : [ "Container employee-rostering3 successfully created with module
employeerostering:employeerostering:1.0.0-SNAPSHOT." ]
      } ],
      "container-alias" : null
    }
  }
}'
 into type: 'class org.kie.server.api.model.ServiceResponse'
```

エラーが発生した場合は、返されたエラーメッセージを確認して、それに応じて Java 設定を調整します。

22.2. サポート対象の KIE SERVER JAVA クライアント

以下は、Red Hat Process Automation Manager の **org.kie.server.client** パッケージで利用可能な Java クライアントサービスの一部です。これらのサービスを使用して、KIE Server REST API と同様に KIE Server の関連リソースを操作できます。

- **KieServicesClient**: 他の KIE Server Java クライアントのエントリーポイントとして使用し、KIE コンテナーを操作します。
- **JobServicesClient**: ジョブ要求のスケジュール、キャンセル、再度のキュー入れ、および取得を行うために使用されます。
- **RuleServicesClient**: ルール関連の操作を実行するためにサーバーにコマンドを送信するために使用されます (ルールの実行、KIE セッションへのオブジェクト挿入など)。
- **SolverServicesClient**: ソルバーステータスや最適解の取得、またはソルバーの破棄など、Red Hat ビルドの OptaPlanner のすべての操作を実行するために使用されます。
- **ProcessServicesClient**: プロセスまたは作業アイテムの開始、シグナル送信、および停止を行うために使用されます。
- **QueryServicesClient**: プロセス、プロセスノード、およびプロセス変数のクエリーを行うために使用されます。
- **UserTaskServicesClient**: (タスクの起動、要求、キャンセルなど) ユーザータスクに関する全操作を実行し、(プロセスインスタンス ID ごと、ユーザーごとなどの) 指定したフィールドでタスクをクエリーするために使用されます。
- **UIServicesClient**: フォーム (XML または JSON) およびプロセスイメージ (SVG) の文字列表現を取得するのに使用されます。
- **ProcessAdminServicesClient**: (~/**org/kie/server/client/admin** にある) プロセスインスタンスを使用するオペレーションにインターフェイスを提供します。
- **UserTaskAdminServicesClient**: (~/**org/kie/server/client/admin** にある) ユーザータスクを使用する操作にインターフェイスを提供します。

getServicesClient メソッドは、これらのクライアントのいずれかへのアクセスを提供します。

```
RuleServicesClient rulesClient = kieServicesClient.getServicesClient(RuleServicesClient.class);
```

利用可能な KIE Server Java クライアントの完全一覧については、Red Hat カスタマーポータル から **Red Hat Process Automation Manager 7.12.0 Source Distribution**をダウンロードして、~/**rhpam-7.12.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-remote/kie-server-client/src/main/java/org/kie/server/client** に移動してください。

22.3. KIE SERVER JAVA クライアント API を使用した要求の例

以下は、KIE Server と基本的な対話を行うための KIE Server Java クライアント API 要求の例です。利用可能な KIE Server Java クライアントの完全一覧については、Red Hat カスタマーポータル から **Red Hat Process Automation Manager 7.12.0 Source Distribution** をダウンロードして、~/**rhpam-7.12.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-remote/kie-server-client/src/main/java/org/kie/server/client** に移動してください。

KIE Server の機能一覧

org.kie.server.api.model.KieServerInfo オブジェクトを使用すると、サーバー機能を特定できます。**KieServicesClient** クライアントが適切にサービスクライアントを生成するには、サーバー機能の情報が必要です。このケーパビリティーは **KieServicesConfiguration** でグローバルに指定するこ

とができます。指定しない場合は、KIE Server から自動的に取得します。

KIE Server ケーパビリティーを返す要求の例

```
public void listCapabilities() {

  KieServerInfo serverInfo = kieServicesClient.getServerInfo().getResult();
  System.out.print("Server capabilities:");

  for (String capability : serverInfo.getCapabilities()) {
    System.out.print(" " + capability);
  }

  System.out.println();
}
```

KIE Server での KIE コンテナーの一覧

KIE コンテナーは **org.kie.server.api.model.KieContainerResource** オブジェクトで表されます。リソース一覧は、**org.kie.server.api.model.KieContainerResourceList** オブジェクトで表されます。

KIE Server から KIE コンテナーを返す要求の例

```
public void listContainers() {
    KieContainerResourceList containersList = kieServicesClient.listContainers().getResult();
    List<KieContainerResource> kieContainers = containersList.getContainers();
    System.out.println("Available containers: ");
    for (KieContainerResource container : kieContainers) {
        System.out.println("\t" + container.getContainerId() + " (" + container.getReleaseId() + ")");
    }
}
```

org.kie.server.api.model.KieContainerResourceFilter クラスのインスタンスを使用して KIE コンテナーの結果をフィルターリングすることもできます。これは **org.kie.server.client.KieServicesClient.listContainers()** メソッドに渡されます。

リリース ID とステータスごとの KIE コンテナーを返す要求の例

```
public void listContainersWithFilter() {

    // Filter containers by releaseId "org.example:container:1.0.0.Final" and status FAILED
    KieContainerResourceFilter filter = new KieContainerResourceFilter.Builder()
            .releaseId("org.example", "container", "1.0.0.Final")
            .status(KieContainerStatus.FAILED)
            .build();

    // Using previously created KieServicesClient
    KieContainerResourceList containersList = kieServicesClient.listContainers(filter).getResult();
    List<KieContainerResource> kieContainers = containersList.getContainers();

    System.out.println("Available containers: ");

    for (KieContainerResource container : kieContainers) {
```

```
            System.out.println("\t" + container.getContainerId() + " (" + container.getReleaseId() + ")");
        }
    }
```

KIE Server での KIE コンテナーの作成および破棄

KieServicesClient で **createContainer** メソッドおよび **disposeContainer** メソッドを使用すると、KIE コンテナーの作成と破棄ができます。この例では、コンテナーを破棄すると、**ServiceResponse** が **Void** 応答を返します。コンテナーを作成すると、**ServiceResponse** が **KieContainerResource** オブジェクトを返します。

KIE コンテナーを破棄して再作成する要求の例

```
public void disposeAndCreateContainer() {
    System.out.println("== Disposing and creating containers ==");

    // Retrieve list of KIE containers
    List<KieContainerResource> kieContainers =
kieServicesClient.listContainers().getResult().getContainers();
    if (kieContainers.size() == 0) {
        System.out.println("No containers available...");
        return;
    }

    // Dispose KIE container
    KieContainerResource container = kieContainers.get(0);
    String containerId = container.getContainerId();
    ServiceResponse<Void> responseDispose = kieServicesClient.disposeContainer(containerId);
    if (responseDispose.getType() == ResponseType.FAILURE) {
        System.out.println("Error disposing " + containerId + ". Message: ");
        System.out.println(responseDispose.getMsg());
        return;
    }
    System.out.println("Success Disposing container " + containerId);
    System.out.println("Trying to recreate the container...");

    // Re-create KIE container
    ServiceResponse<KieContainerResource> createResponse =
kieServicesClient.createContainer(containerId, container);
    if(createResponse.getType() == ResponseType.FAILURE) {
        System.out.println("Error creating " + containerId + ". Message: ");
        System.out.println(responseDispose.getMsg());
        return;
    }
    System.out.println("Container recreated with success!");
}
```

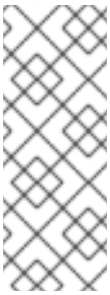
KIE Server でのランタイムコマンドの実行

Red Hat Process Automation Manager はランタイムコマンドをサポートしています。これは、KIE セッションでオブジェクトを挿入したり取り消したり、全ルールを実行するなどのアセット関連の操作のために KIE Server に送信するものです。サポートされるランタイムコマンドの全一覧は、Red Hat Process Automation Manager インスタンスの **org.drools.core.command.runtime** パッケージにあります。

コマンドの挿入には **org.kie.api.command.KieCommands** クラスを使用し、**KieCommands** クラスのインスタンス化には **org.kie.api.KieServices.get().getCommands()** を使用することもできます。複数のコマンドを追加するには、**BatchExecutionCommand** ラッパーを使用します。

オブジェクトの挿入および全ルール実行の要求の例

```
import org.kie.api.command.Command;
import org.kie.api.command.KieCommands;
import org.kie.server.api.model.ServiceResponse;
import org.kie.server.client.RuleServicesClient;
import org.kie.server.client.KieServicesClient;
import org.kie.api.KieServices;

import java.util.Arrays;

...

public void executeCommands() {

  String containerId = "hello";
  System.out.println("== Sending commands to the server ==");
  RuleServicesClient rulesClient = kieServicesClient.getServicesClient(RuleServicesClient.class);
  KieCommands commandsFactory = KieServices.Factory.get().getCommands();

  Command<?> insert = commandsFactory.newInsert("Some String OBJ");
  Command<?> fireAllRules = commandsFactory.newFireAllRules();
  Command<?> batchCommand = commandsFactory.newBatchExecution(Arrays.asList(insert,
fireAllRules));

  ServiceResponse<String> executeResponse = rulesClient.executeCommands(containerId,
batchCommand);

  if(executeResponse.getType() == ResponseType.SUCCESS) {
    System.out.println("Commands executed with success! Response: ");
    System.out.println(executeResponse.getResult());
  } else {
    System.out.println("Error executing rules. Message: ");
    System.out.println(executeResponse.getMsg());
  }
}
```



注記

クライアントとクラスター環境内の特定の KIE Server コンテナーとの対話は、一意の **conversationID** でセキュリティーが保たれます。**conversationID** は **X-KIE-ConversationId** REST ヘッダーを使用して送信されます。コンテナーを更新する場合は、以前の **conversationID** の設定を解除します。**KieServiesClient.completeConversation()** を使用して Java API の **conversationID** を設定解除します。

KIE コンテナーで利用可能なビジネスプロセスの一覧

QueryServicesClient クライアントを使用すると、利用可能なプロセス定義を一覧表示できます。**QueryServicesClient** メソッドはページネーションを使用するため、実行するクエリーの他に現在のページと 1 ページごとの結果数を提供する必要があります。この例では、クエリーはページ **0**

から始まり、最初の **1000** 件の結果を表示します。

KIE Server でのビジネスプロセス一覧要求の例

```
public void listProcesses() {
    System.out.println("== Listing Business Processes ==");
    QueryServicesClient queryClient =
kieServicesClient.getServicesClient(QueryServicesClient.class);
    List<ProcessDefinition> findProcessesByContainerId =
queryClient.findProcessesByContainerId("rewards", 0, 1000);
    for (ProcessDefinition def : findProcessesByContainerId) {
        System.out.println(def.getName() + " - " + def.getId() + " v" + def.getVersion());
    }
}
```

KIE コンテナーでのビジネスプロセスの開始

ProcessServicesClient クライアントを使用してビジネスプロセスを開始します。プロセスに必要なカスタムクラスは **addExtraClasses()** メソッドを使用して **KieServicesConfiguration** オブジェクトに追加してください。

ビジネスプロセスを開始する要求の例

```
import java.util.HashMap;
import java.util.HashSet;
import java.util.Map;
import java.util.Set;

import javax.xml.bind.JAXBContext;
import javax.xml.bind.JAXBException;
import javax.xml.bind.Marshaller;

import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.client.KieServicesClient;
import org.kie.server.client.KieServicesConfiguration;
import org.kie.server.client.KieServicesFactory;
import org.kie.server.client.ProcessServicesClient;
...

public static void startProcess() {

  //Client configuration setup
  KieServicesConfiguration config = KieServicesFactory.newRestConfiguration(SERVER_URL,
LOGIN, PASSWORD);

  //Add custom classes, such as Obj.class, to the configuration
  Set<Class<?>> extraClassList = new HashSet<Class<?>>();
  extraClassList.add(Obj.class);
  config.addExtraClasses(extraClassList);
  config.setMarshallingFormat(MarshallingFormat.JSON);

  // ProcessServicesClient setup
  KieServicesClient client = KieServicesFactory.newKieServicesClient(config);
  ProcessServicesClient processServicesClient =
client.getServicesClient(ProcessServicesClient.class);
```

```
    // Create an instance of the custom class
    Obj obj = new Obj();
    obj.setOk("ok");

    Map<String, Object> variables = new HashMap<String, Object>();
    variables.put("test", obj);

    // Start the process with custom class
    processServicesClient.startProcess(CONTAINER, processId, variables);
}
```

カスタムクエリーの実行

QueryServicesClient クライアントの **QueryDefinition** オブジェクトを使用して、KIE Server でカスタムクエリーを登録および実行します。

KIE Server でのカスタムクエリー登録要求および実行要求の例

```
// Client setup
KieServicesConfiguration conf = KieServicesFactory.newRestConfiguration(SERVER_URL,
LOGIN, PASSWORD);
KieServicesClient client = KieServicesFactory.newKieServicesClient(conf);

// Get the QueryServicesClient
QueryServicesClient queryClient = client.getServicesClient(QueryServicesClient.class);

// Build the query
QueryDefinition queryDefinition = QueryDefinition.builder().name(QUERY_NAME)
        .expression("select * from Task t")
        .source("java:jboss/datasources/ExampleDS")
        .target("TASK").build();

// Specify that two queries cannot have the same name
queryClient.unregisterQuery(QUERY_NAME);

// Register the query
queryClient.registerQuery(queryDefinition);

// Execute the query with parameters: query name, mapping type (to map the fields to an object),
page number, page size, and return type
List<TaskInstance> query = queryClient.query(QUERY_NAME,
QueryServicesClient.QUERY_MAP_TASK, 0, 100, TaskInstance.class);

// Read the result
for (TaskInstance taskInstance : query) {
    System.out.println(taskInstance);
}
```

この例では、**target** がクエリーサービスにデフォルトフィルターを適用するように指示しています。別の方法では、フィルターのパラメーターを手動で設定することもできます。**Target** クラスは以下の値をサポートしています。

```
public enum Target {
    PROCESS,
    TASK,
```

```
    BA_TASK,
    PO_TASK,
    JOBS,
    CUSTOM;
}
```

第23章 RED HAT PROCESS AUTOMATION MANAGER での KIE SERVER および KIE コンテナーコマンド

Red Hat Process Automation Manager は、サーバー情報を取得したり、コンテナーの作成や削除など、サーバー関連またはコンテナー関連の操作のために KIE Server に送信するサーバーコマンドをサポートしています。サポートされる KIE Server 設定コマンドの全一覧は、Red Hat Process Automation Manager インスタンスの **org.kie.server.api.commands** パッケージにあります。

KIE Server REST API では、**org.kie.server.api.commands** コマンドを **http://SERVER:PORT/kie-server/services/rest/server/config** への **POST** 要求のボディとして使用します。KIE Server REST API の使用に関する詳細情報は、*21章KIE コンテナーおよびビジネスアセット用の KIE Server REST API* を参照してください。

KIE Server Java クライアント API では、親 **KieServicesClient** Java クライアントで対応するメソッドを Java アプリケーションで埋め込み API 要求として使用します。KIE Server コマンドはすべて Java クライアント API で提供されるメソッドが実行するため、実際の KIE Server コマンドを Java アプリケーションに埋め込む必要はありません。KIE Server Java クライアント API の使用に関する詳細情報は、*22章KIE コンテナーおよびビジネスアセット用の KIE Server Java クライアント API* を参照してください。

23.1. KIE SERVER および KIE コンテナーのコマンドサンプル

以下は、KIE Server で KIE Server REST API または Java クライアント API のサーバー関連もしくはコンテナー関連操作に使用可能な KIE Server コマンドのサンプルです。

- **GetServerInfoCommand**
- **GetServerStateCommand**
- **CreateContainerCommand**
- **GetContainerInfoCommand**
- **ListContainersCommand**
- **CallContainerCommand**
- **DisposeContainerCommand**
- **GetScannerInfoCommand**
- **UpdateScannerCommand**
- **UpdateReleaseIdCommand**

サポートされる KIE Server 設定および管理コマンドの全一覧は、Red Hat Process Automation Manager インスタンスの **org.kie.server.api.commands** パッケージにあります。

KIE Server コマンドは個別に実行するか、バッチ REST API 要求またはバッチ Java API 要求として実行することができます。

KIE コンテナー (JSON) を作成、呼び出し、破棄するバッチ REST API 要求

```
{
  "commands": [
```

```
    {
      "create-container": {
        "container": {
          "status": "STARTED",
          "container-id": "command-script-container",
          "release-id": {
            "version": "1.0",
            "group-id": "com.redhat",
            "artifact-id": "Project1"
          }
        }
      }
    },
    {
      "call-container": {
        "payload": "{\n  \"commands\" : [ {\n    \"fire-all-rules\" : {\n      \"max\" : -1,\n      \"out-identifier\" :
null\n    }\n  } ]\n}",
        "container-id": "command-script-container"
      }
    },
    {
      "dispose-container": {
        "container-id": "command-script-container"
      }
    }
  ]
}
```

KIE コンテナーを取得、破棄、再作成するバッチ Java API 要求

```
public void disposeAndCreateContainer() {
    System.out.println("== Disposing and creating containers ==");

    // Retrieve list of KIE containers
    List<KieContainerResource> kieContainers =
kieServicesClient.listContainers().getResult().getContainers();
    if (kieContainers.size() == 0) {
        System.out.println("No containers available...");
        return;
    }

    // Dispose KIE container
    KieContainerResource container = kieContainers.get(0);
    String containerId = container.getContainerId();
    ServiceResponse<Void> responseDispose = kieServicesClient.disposeContainer(containerId);
    if (responseDispose.getType() == ResponseType.FAILURE) {
        System.out.println("Error disposing " + containerId + ". Message: ");
        System.out.println(responseDispose.getMsg());
        return;
    }
    System.out.println("Success Disposing container " + containerId);
    System.out.println("Trying to recreate the container...");

    // Re-create KIE container
    ServiceResponse<KieContainerResource> createResponse =
kieServicesClient.createContainer(containerId, container);
```

```
    if(createResponse.getType() == ResponseType.FAILURE) {
        System.out.println("Error creating " + containerId + ". Message: ");
        System.out.println(responseDispose.getMsg());
        return;
    }
    System.out.println("Container recreated with success!");
}
```

本セクションの各コマンドには、KIE Server REST API 用の REST 要求のボディ例 (JSON) と KIE Server Java クライアント API 用の **KieServicesClient** Java クライアントからの埋め込みメソッド例が含まれています。

GetServerInfoCommand

KIE Server インスタンスに関する情報を返します。

REST 要求のボディ (JSON) の例

```
{
  "commands" : [ {
    "get-server-info" : { }
  } ]
}
```

Java クライアントメソッドの例

```
KieServerInfo serverInfo = kieServicesClient.getServerInfo();
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Kie Server info",
      "result": {
        "kie-server-info": {
          "id": "default-kieserver",
          "version": "7.11.0.Final-redhat-00001",
          "name": "default-kieserver",
          "location": "http://localhost:8080/kie-server/services/rest/server",
          "capabilities": [
            "KieServer",
            "BRM",
            "BPM",
            "CaseMgmt",
            "BPM-UI",
            "BRP",
            "DMN",
            "Swagger"
          ],
          "messages": [
            {
              "severity": "INFO",
              "timestamp": {
```

```
            "java.util.Date": 1538502533321
          },
          "content": [
            "Server KieServerInfo{serverId='default-kieserver', version='7.11.0.Final-redhat-00001',
name='default-kieserver', location='http://localhost:8080/kie-server/services/rest/server',
capabilities=[KieServer, BRM, BPM, CaseMgmt, BPM-UI, BRP, DMN, Swagger],
messages=null}started successfully at Tue Oct 02 13:48:53 EDT 2018"
          ]
        }
      ]
    }
  }
 ]
}
```


GetServerStateCommand

KIE Server インスタンスの現在の状態と設定に関する情報を返します。

REST 要求のボディ (JSON) の例

```
{
  "commands" : [ {
    "get-server-state" : { }
  } ]
}
```

Java クライアントメソッドの例

```
KieServerStateInfo serverStateInfo = kieServicesClient.getServerState();
```

サーバーの応答例 (JSON)


```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Successfully loaded server state for server id default-kieserver",
      "result": {
        "kie-server-state-info": {
          "controller": [
            "http://localhost:8080/business-central/rest/controller"
          ],
          "config": {
            "config-items": [
              {
                "itemName": "org.kie.server.location",
                "itemValue": "http://localhost:8080/kie-server/services/rest/server",
                "itemType": "java.lang.String"
              },
              {
                "itemName": "org.kie.server.controller.user",
                "itemValue": "controllerUser",
                "itemType": "java.lang.String"
```

```
            },
            {
              "itemName": "org.kie.server.controller",
              "itemValue": "http://localhost:8080/business-central/rest/controller",
              "itemType": "java.lang.String"
            }
          ]
        },
        "containers": [
          {
            "container-id": "employee-rostering",
            "release-id": {
              "group-id": "employeerostering",
              "artifact-id": "employeerostering",
              "version": "1.0.0-SNAPSHOT"
            },
            "resolved-release-id": null,
            "status": "STARTED",
            "scanner": {
              "status": "STOPPED",
              "poll-interval": null
            },
            "config-items": [
              {
                "itemName": "KBase",
                "itemValue": "",
                "itemType": "BPM"
              },
              {
                "itemName": "KSession",
                "itemValue": "",
                "itemType": "BPM"
              },
              {
                "itemName": "MergeMode",
                "itemValue": "MERGE_COLLECTIONS",
                "itemType": "BPM"
              },
              {
                "itemName": "RuntimeStrategy",
                "itemValue": "SINGLETON",
                "itemType": "BPM"
              }
            ],
            "messages": [],
            "container-alias": "employeerostering"
          }
        ]
      }
    }
  }
 ]
}
```


CreateContainerCommand

KIE Server の KIE コンテナーを作成します。

表23.1コマンドの属性

Name	説明	要件
container	**container-id**、**release-id** データ (グループ ID、アーティファクト ID、バージョン)、**status**、および新規 KIE コンテナーの他のコンポーネントを含むマップ。	必須

REST 要求のボディ (JSON) の例

```
{
  "commands" : [ {
    "create-container" : {
      "container" : {
        "status" : null,
        "messages" : [ ],
        "container-id" : "command-script-container",
        "release-id" : {
          "version" : "1.0",
          "group-id" : "com.redhat",
          "artifact-id" : "Project1"
        },
        "config-items" : [ ]
      }
    }
  } ]
}
```

Java クライアントメソッドの例

```
ServiceResponse<KieContainerResource> response =
kieServicesClient.createContainer("command-script-container", resource);
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully deployed with module
com.redhat:Project1:1.0.",
      "result": {
        "kie-container": {
          "container-id": "command-script-container",
          "release-id": {
            "version" : "1.0",
            "group-id" : "com.redhat",
            "artifact-id" : "Project1"
          },
          "resolved-release-id": {
```

```
            "version" : "1.0",
            "group-id" : "com.redhat",
            "artifact-id" : "Project1"
          },
          "status": "STARTED",
          "scanner": {
            "status": "DISPOSED",
            "poll-interval": null
          },
          "config-items": [],
          "messages": [
            {
              "severity": "INFO",
              "timestamp": {
                "java.util.Date": 1538762455510
              },
              "content": [
                "Container command-script-container successfully created with module
com.redhat:Project1:1.0."
              ]
            }
          ],
          "container-alias": null
        }
      }
    }
  ]
}
```


GetContainerInfoCommand

KIE Server の指定された KIE コンテナーに関する情報を返します。

表23.2 コマンドの属性

Name	説明	要件
container-id	KIE コンテナーの ID	必須

REST 要求のボディ (JSON) の例


```
{
  "commands" : [ {
    "get-container-info" : {
      "container-id" : "command-script-container"
    }
  } ]
}
```


Java クライアントメソッドの例

```
ServiceResponse<KieContainerResource> response =
kieServicesClient.getContainerInfo("command-script-container");
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Info for container command-script-container",
      "result": {
        "kie-container": {
          "container-id": "command-script-container",
          "release-id": {
            "group-id": "com.redhat",
            "artifact-id": "Project1",
            "version": "1.0"
          },
          "resolved-release-id": {
            "group-id": "com.redhat",
            "artifact-id": "Project1",
            "version": "1.0"
          },
          "status": "STARTED",
          "scanner": {
            "status": "DISPOSED",
            "poll-interval": null
          },
          "config-items": [

          ],
          "container-alias": null
        }
      }
    }
  ]
}
```

ListContainersCommand

KIE Server インスタンスで作成された KIE コンテナー一覧を返します。

表23.3 コマンドの属性

Name	説明	要件
kie-container-filter	**release-id-filter**、**container-status-filter**、および結果のフィルターリングに使用する他の KIE コンテナープロパティーを含むオプションのマップ。	任意

REST 要求のボディ (JSON) の例

```
{
  "commands" : [ {
    "list-containers" : {
      "kie-container-filter" : {
```

```
        "release-id-filter" : { },
        "container-status-filter" : {
          "accepted-status" : ["FAILED"]
        }
      }
    }
  } ]
}
```

Java クライアントメソッドの例

```
KieContainerResourceFilter filter = new KieContainerResourceFilter.Builder()
        .status(KieContainerStatus.FAILED)
        .build();

KieContainerResourceList containersList = kieServicesClient.listContainers(filter);
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "List of created containers",
      "result": {
        "kie-containers": {
          "kie-container": [
            {
              "container-id": "command-script-container",
              "release-id": {
                "group-id": "com.redhat",
                "artifact-id": "Project1",
                "version": "1.0"
              },
              "resolved-release-id": {
                "group-id": "com.redhat",
                "artifact-id": "Project1",
                "version": "1.0"
              },
              "status": "STARTED",
              "scanner": {
                "status": "STARTED",
                "poll-interval": 5000
              },
              "config-items": [
                {
                  "itemName": "RuntimeStrategy",
                  "itemValue": "SINGLETON",
                  "itemType": "java.lang.String"
                },
                {
                  "itemName": "MergeMode",
                  "itemValue": "MERGE_COLLECTIONS",
                  "itemType": "java.lang.String"
                },
```

```
              {
                "itemName": "KBase",
                "itemValue": "",
                "itemType": "java.lang.String"
              },
              {
                "itemName": "KSession",
                "itemValue": "",
                "itemType": "java.lang.String"
              }
            ],
            "messages": [
              {
                "severity": "INFO",
                "timestamp": {
                  "java.util.Date": 1538504619749
                },
                "content": [
                  "Container command-script-container successfully created with module
com.redhat:Project1:1.0."
                ]
              }
            ],
            "container-alias": null
          }
        ]
      }
    }
  }
 ]
}
```


CallContainerCommand

KIE コンテナーを呼び出し、1 つ以上のランタイムコマンドを実行します。Red Hat Process Automation Manager ランタイムコマンドの情報は、*24章Red Hat Process Automation Manager のランタイムコマンド* を参照してください。

表23.4 コマンドの属性

Name	説明	要件
container-id	呼び出される KIE コンテナーの ID	必須
payload	KIE コンテナーで実行される **BatchExecutionCommand** ラッパー内の 1 つ以上のコマンド	必須

REST 要求のボディ (JSON) の例


```
{
  "commands" : [ {
    "call-container" : {
      "payload" : "{\n  \"lookup\" : \"defaultKieSession\",\n  \"commands\" : [ {\n    \"fire-all-rules\" : {\n
```

```
\"max\" : -1,\n      \"out-identifier\" : null\n    }\n  } ]\n}",
      "container-id" : "command-script-container"
    }
  } ]
}
```

Java クライアントメソッドの例

```
List<Command<?>> commands = new ArrayList<Command<?>>();
      BatchExecutionCommand batchExecution1 =
commandsFactory.newBatchExecution(commands, "defaultKieSession");
      commands.add(commandsFactory.newFireAllRules());

      ServiceResponse<ExecutionResults> response1 =
ruleClient.executeCommandsWithResults("command-script-container", batchExecution1);
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": "{\n  \"results\" : [ ],\n  \"facts\" : [ ]\n}"
    }
  ]
}
```

DisposeContainerCommand

KIE Server の指定された KIE コンテナーを破棄します。

表23.5 コマンドの属性

Name	説明	要件
container-id	破棄される KIE コンテナーの ID	必須

REST 要求のボディ (JSON) の例

```
{
  "commands" : [ {
    "dispose-container" : {
      "container-id" : "command-script-container"
    }
  } ]
}
```

Java クライアントメソッドの例

```
ServiceResponse<Void> response = kieServicesClient.disposeContainer("command-script-
container");
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully disposed.",
      "result": null
    }
  ]
}
```

GetScannerInfoCommand

該当する場合は、指定された KIE コンテナー内の自動更新に使用される KIE スキャナーに関する情報を返します。

表23.6 コマンドの属性

Name	説明	要件
container-id	KIE スキャナーを使用する KIE コンテナーの ID	必須

REST 要求のボディ (JSON) の例

```
{
  "commands" : [ {
    "get-scanner-info" : {
      "container-id" : "command-script-container"
    }
  } ]
}
```

Java クライアントメソッドの例

```
ServiceResponse<KieScannerResource> response =
kieServicesClient.getScannerInfo("command-script-container");
```

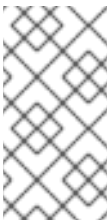
サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Scanner info successfully retrieved",
      "result": {
        "kie-scanner": {
          "status": "DISPOSED",
          "poll-interval": null
        }
      }
```

```
    }
  ]
}
```

UpdateScannerCommand

更新済み KIE コンテナーデプロイメントのポーリングを制御する KIE スキャナーを起動または停止します。



注記

ビジネスプロセスと KIE スキャナーを併用しないようにしてください。プロセスで KIE スキャナーを使用すると、予期せぬ更新が発生し、プロセスインスタンスの実行と互換性のない変更が加えられた場合に、長時間実行中のプロセスでエラーが発生する可能性があります。

表23.7 コマンドの属性

Name	説明	要件
container-id	KIE スキャナーを使用する KIE コンテナーの ID	必須
status	KIE スキャナーに設定するステータス (**STARTED**、**STOPPED**)	必須
poll-interval	ポーリングの時間 (単位: ミリ秒)	スキャナーの起動時にのみ必須

REST 要求のボディ (JSON) の例

```
{
  "commands" : [ {
    "update-scanner" : {
      "scanner" : {
        "status" : "STARTED",
        "poll-interval" : 10000
      },
      "container-id" : "command-script-container"
    }
  } ]
}
```

Java クライアントメソッドの例

```
KieScannerResource scannerResource = new KieScannerResource();
scannerResource.setPollInterval(10000);
scannerResource.setStatus(KieScannerStatus. STARTED);

ServiceResponse<KieScannerResource> response =
kieServicesClient.updateScanner("command-script-container", scannerResource);
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Kie scanner successfully created.",
      "result": {
        "kie-scanner": {
          "status": "STARTED",
          "poll-interval": 10000
        }
      }
    }
  ]
}
```

UpdateReleaseIdCommand

指定した KIE コンテナーのリリース ID データ (グループ ID、アーティファクト ID、バージョン) を更新します。

表23.8 コマンドの属性

Name	説明	要件
container-id	更新される KIE コンテナーの ID	必須
releaseId	KIE コンテナーに適用される更新済み GAV (グループ ID、アーティファクト ID、バージョン) データ	必須

REST 要求のボディ (JSON) の例

```
{
  "commands" : [ {
    "update-release-id" : {
      "releaseId" : {
        "version" : "1.1",
        "group-id" : "com.redhat",
        "artifact-id" : "Project1"
      },
      "container-id" : "command-script-container"
    }
  } ]
}
```

Java クライアントメソッドの例

```
ServiceResponse<ReleaseId> response = kieServicesClient.updateReleaseId("command-script-container", "com.redhat:Project1:1.1");
```

サーバーの応答例 (JSON)

```
{
```

```
 "response": [
  {
   "type": "SUCCESS",
   "msg": "Release id successfully updated",
   "result": {
    "release-id": {
     "group-id": "com.redhat",
     "artifact-id": "Project1",
     "version": "1.1"
    }
   }
  }
 ]
}
```

第24章 RED HAT PROCESS AUTOMATION MANAGER のランタイムコマンド

Red Hat Process Automation Manager はランタイムコマンドをサポートしています。これは、KIE セッションで全ルールを実行したり、オブジェクトを挿入したり取り消したりするなどのアセット関連の操作のために KIE Server に送信するものです。サポートされるランタイムコマンドの全一覧は、Red Hat Process Automation Manager インスタンスの **org.drools.core.command.runtime** パッケージにあります。

KIE Server REST API では、グローバルの **org.drools.core.command.runtime** コマンドまたはルール固有の **org.drools.core.command.runtime.rule** コマンドを **http://SERVER:PORT/kie-server/services/rest/server/containers/instances/{containerId}** への **POST** 要求のボディとして使用します。KIE Server REST API の使用に関する詳細情報は、*21章KIE コンテナーおよびビジネスアセット用の KIE Server REST API* を参照してください。

KIE Server Java クライアント API では、関連する Java クライアントの Java アプリケーションにこれらのコマンドを埋め込むことができます。たとえばルール関連のコマンドでは、**RuleServicesClient** Java クライアントを埋め込むコマンドに使用します。KIE Server Java クライアント API の使用に関する詳細情報は、*22章KIE コンテナーおよびビジネスアセット用の KIE Server Java クライアント API* を参照してください。

24.1. RED HAT PROCESS AUTOMATION MANAGER のランタイムコマンドのサンプル

以下は、KIE Server で KIE Server REST API または Java クライアント API のアセット関連演算に使用可能なランタイムコマンドのサンプルです。

- **BatchExecutionCommand**
- **InsertObjectCommand**
- **RetractCommand**
- **ModifyCommand**
- **GetObjectCommand**
- **GetObjectsCommand**
- **InsertElementsCommand**
- **FireAllRulesCommand**
- **StartProcessCommand**
- **SignalEventCommand**
- **CompleteWorkItemCommand**
- **AbortWorkItemCommand**
- **QueryCommand**
- **SetGlobalCommand**

- **GetGlobalCommand**

サポートされるランタイムコマンドの全一覧は、Red Hat Process Automation Manager インスタンスの **org.drools.core.command.runtime** パッケージにあります。

本セクションの各コマンドには、KIE Server REST API 用の REST 要求のボディ例 (JSON) と KIE Server Java クライアント API 用の Java クライアントの埋め込みコマンド例が含まれています。Java の例では、**name** (文字列) と **age** (整数) のフィールドがあるオブジェクト **org.drools.compiler.test.Person** を使用しています。

BatchExecutionCommand

同時に実行する複数のコマンドが含まれています。

表24.1 コマンドの属性

Name	説明	要件
commands	実行されるコマンド一覧	必須
lookup	コマンドの実行対象である KIE セッション ID を設定します。ステートレス KIE セッションの場合、この属性は必須です。ステートフル KIE セッションの場合この属性は任意で、指定されていないとデフォルトの KIE セッションが使用されます。	ステートレス KIE セッションでは必須、ステートフル KIE セッションでは任意

注記

KIE セッションの ID は、Red Hat Process Automation Manager プロジェクトの **kmodule.xml** ファイルに含まれます。Business Central で KIE セッション ID を表示するか追加して、**lookup** コマンド属性と併用するには、Business Central の関連のプロジェクトに移動し、プロジェクトの **Settings → KIE bases → KIE sessions** に移動します。KIE ベースが存在しない場合は、**Add KIE base→ KIE sessions** の順にクリックして、新規の KIE ベースと KIE セッションを定義します。

JSON リクエストボディの例

```
{
  "lookup": "ksession1",
  "commands": [ {
     "insert": {
       "object": {
         "org.drools.compiler.test.Person": {
           "name": "john",
           "age": 25
         }
       }
     }
    },
    {
      "fire-all-rules": {
        "max": 10,
        "out-identifier": "firedActivations"
      }
```

```
    }
  ]
}
```

Java コマンドの例

```
InsertObjectCommand insertCommand = new InsertObjectCommand(new Person("john", 25));
FireAllRulesCommand fireCommand = new FireAllRulesCommand();

BatchExecutionCommand batch = new
BatchExecutionCommandImpl(Arrays.asList(insertCommand, fireCommand), "ksession1");
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [
            {
              "value": 0,
              "key": "firedActivations"
            }
          ],
          "facts": []
        }
      }
    }
  ]
}
```

InsertObjectCommand

KIE セッションにオブジェクトを挿入します。

表24.2 コマンドの属性

Name	説明	要件
object	挿入するオブジェクト	必須
out-identifier	オブジェクト挿入から作成され、実行結果に追加される **FactHandle** の ID	任意
return-object	実行結果にオブジェクトを返す必要があるかどうかを決定するブール値 (デフォルト値: **true**)	任意
entry-point	挿入のエントリーポイント	任意

JSON リクエストボディの例

```
{
  "commands": [ {
     "insert": {
      "entry-point": "my stream",
      "object": {
        "org.drools.compiler.test.Person": {
          "age": 25,
          "name": "john"
        }
      },
      "out-identifier": "john",
      "return-object": false
     }
    }
  ]
}
```

Java コマンドの例

```
Command insertObjectCommand =
  CommandFactory.newInsert(new Person("john", 25), "john", false, null);

ksession.execute(insertObjectCommand);
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [],
          "facts": [
            {
              "value": {
                "org.drools.core.common.DefaultFactHandle": {
                  "external-form": "0:4:436792766:-
2127720265:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap"
                }
              },
              "key": "john"
            }
          ]
        }
      }
    }
  ]
}
```

RetractCommand

KIE セッションからオブジェクトを取り消します。

表24.3 コマンドの属性

Name	説明	要件
fact-handle	取り消すオブジェクトに関連付けられた **FactHandle**	必須

JSON リクエストボディの例

```
{
  "commands": [ {
     "retract": {
       "fact-handle": "0:4:436792766:-
2127720265:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap"
     }
   }
  ]
}
```

Java コマンドの使用例: FactHandleFromString

```
RetractCommand retractCommand = new RetractCommand();
retractCommand.setFactHandleFromString("123:234:345:456:567");
```

Java コマンドの使用例: 挿入されたオブジェクトから FactHandle の使用

```
RetractCommand retractCommand = new RetractCommand(factHandle);
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container employee-rostering successfully called.",
      "result": {
        "execution-results": {
          "results": [],
          "facts": []
        }
      }
    }
  ]
}
```

ModifyCommand

KIE セッションに以前に挿入されたオブジェクトを修正します。

表24.4 コマンドの属性

Name	説明	要件
fact-handle	修正するオブジェクトに関連付けられた **FactHandle**	必須
setters	オブジェクト修正のセッター一覧	必須

JSON リクエストボディの例

```
{
  "commands": [ {
     "modify": {
       "fact-handle": "0:4:436792766:-
2127720265:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap",
       "setters": {
         "accessor": "age",
         "value": 25
       }
     }
   }
  ]
}
```

Java コマンドの例

```
ModifyCommand modifyCommand = new ModifyCommand(factHandle);

List<Setter> setters = new ArrayList<Setter>();
setters.add(new SetterImpl("age", "25"));

modifyCommand.setSetters(setters);
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container employee-rostering successfully called.",
      "result": {
        "execution-results": {
          "results": [],
          "facts": []
        }
      }
    }
  ]
}
```

GetObjectCommand

KIE セッションからオブジェクトを取得します。

表24.5 コマンドの属性

Name	説明	要件
fact-handle	取得するオブジェクトに関連付けられた **FactHandle**	必須
out-identifier	オブジェクト挿入から作成され、実行結果に追加される **FactHandle** の ID	任意

JSON リクエストボディの例

```
{
  "commands": [ {
     "get-object": {
       "fact-handle": "0:4:436792766:-
2127720265:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap",
       "out-identifier": "john"
     }
   }
  ]
}
```

Java コマンドの例

```
GetObjectCommand getObjectCommand = new GetObjectCommand();
getObjectCommand.setFactHandleFromString("123:234:345:456:567");
getObjectCommand.setOutIdentifier("john");
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [
            {
              "value": null,
              "key": "john"
            }
          ],
          "facts": []
        }
      }
    }
  ]
}
```

GetObjectsCommand

KIE セッションからすべてのオブジェクトをコレクションとして取得します。

表24.6 コマンドの属性

Name	説明	要件
object-filter	KIE セッションから返されるオブジェクト用のフィルター	任意
out-identifier	実行結果に使用する識別子	任意

JSON リクエストボディの例

```
{
  "commands": [ {
     "get-objects": {
       "out-identifier": "objects"
     }
   }
  ]
}
```

Java コマンドの例

```
GetObjectsCommand getObjectsCommand = new GetObjectsCommand();
getObjectsCommand.setOutIdentifier("objects");
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [
            {
              "value": [
                {
                  "org.apache.xerces.dom.ElementNSImpl": "<?xml version=\"1.0\" encoding=\"UTF-
16\"?>\n<object xmlns:xsi=\"http://www.w3.org/2001/XMLSchema-instance\" xsi:type=\"person\">
<age>25</age><name>john</name>\n <\/object>"
                },
                {
                  "org.drools.compiler.test.Person": {
                    "name": "john",
                    "age": 25
                  }
                }
              ],
              "key": "objects"
            }
```

```
          ],
          "facts": []
        }
      }
    }
  ]
}
```

InsertElementsCommand

KIE セッションにオブジェクト一覧を挿入します。

表24.7 コマンドの属性

Name	説明	要件
objects	KIE セッションに挿入するオブジェクト一覧	必須
out-identifier	オブジェクト挿入から作成され、実行結果に追加される **FactHandle** の ID	任意
return-object	実行結果にオブジェクトを返す必要があるかどうかを決定するブール値。デフォルト値は **true** です。	任意
entry-point	挿入のエントリーポイント	任意

JSON リクエストボディの例

```
{
  "commands": [ {
    "insert-elements": {
        "objects": [
            {
                "containedObject": {
                    "@class": "org.drools.compiler.test.Person",
                    "age": 25,
                    "name": "john"
                }
            },
            {
                "containedObject": {
                    "@class": "Person",
                    "age": 35,
                    "name": "sarah"
                }
            }
        ]
    }
  }
]
}
```

Java コマンドの例

```
List<Object> objects = new ArrayList<Object>();
objects.add(new Person("john", 25));
objects.add(new Person("sarah", 35));

Command insertElementsCommand = CommandFactory.newInsertElements(objects);
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [],
          "facts": [
            {
              "value": {
                "org.drools.core.common.DefaultFactHandle": {
                  "external-form": "0:4:436792766:-
2127720265:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap"
                }
              },
              "key": "john"
            },
            {
              "value": {
                "org.drools.core.common.DefaultFactHandle": {
                  "external-form": "0:4:436792766:-
2127720266:4:DEFAULT:NON_TRAIT:java.util.LinkedHashMap"
                }
              },
              "key": "sarah"
            }
          ]
        }
      }
    }
  ]
}
```

FireAllRulesCommand

KIE セッションですべてのルールを実行します。

表24.8 コマンドの属性

Name	説明	要件
max	実行するルールの最大数。デフォルト値は **-1** で、実行に制限を課しません。	任意

Name	説明	要件
out-identifier	実行結果で、使用されたルール数の取得に使用される ID。	任意
agenda-filter	ルール実行に使用するアジェンダフィルター。	任意

JSON リクエストボディの例

```
{
  "commands" : [ {
    "fire-all-rules": {
        "max": 10,
        "out-identifier": "firedActivations"
    }
  } ]
}
```

Java コマンドの例

```
FireAllRulesCommand fireAllRulesCommand = new FireAllRulesCommand();
fireAllRulesCommand.setMax(10);
fireAllRulesCommand.setOutIdentifier("firedActivations");
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [
            {
              "value": 0,
              "key": "firedActivations"
            }
          ],
          "facts": []
        }
      }
    }
  ]
}
```

StartProcessCommand

プロセス ID を使用してプロセスを開始します。パラメーターと初期データを渡して挿入することもできます。

表24.9 コマンドの属性

Name	説明	要件
processId	開始するプロセスの ID	必須
parameters	プロセスのスタートアップでパラメーターを渡す **Map <String,Object>** 引数	任意
data	プロセスのスタートアップ前に KIE セッションに挿入するオブジェクト一覧	任意

JSON リクエストボディの例

```
{
  "commands": [
    {
      "start-process": {
        "processId": "myProject.myProcess",
        "data": null,
        "parameter": [],
        "out-identifier": null
      }
    }
  ]
}
```

Java コマンドの例

```
StartProcessCommand startProcessCommand = new StartProcessCommand();
startProcessCommand.setProcessId("org.drools.task.processOne");
```

サーバーの応答例 (JSON)

```
{
  "type": "SUCCESS",
  "msg": "Container stateful-session successfully called.",
  "result": {
    "execution-results": {
      "results": [],
      "facts": []
    }
  }
}
```

SignalEventCommand

KIE セッションに単一イベントを送信します。

表24.10 コマンドの属性

Name	説明	要件
event-type	着信イベントのタイプ	必須
process-instance-id	通知されるプロセスインスタンスの ID	任意
event	着信イベントのデータ	任意

JSON リクエストボディの例

```
{
  "commands": [
    {
      "signal-event": {
        "process-instance-id": 1001,
        "correlation-key": null,
        "event-type": "start",
        "event": {
          "org.kie.server.testing.Person": {
            "fullname": "john",
            "age": 25
          }
        }
      }
    }
  ]
}
```

Java コマンドの例

```
SignalEventCommand signalEventCommand = new SignalEventCommand();
signalEventCommand.setProcessInstanceId(1001);
signalEventCommand.setEventType("start");
signalEventCommand.setEvent(new Person("john", 25));
```

サーバーの応答例 (JSON)

```
{
  "type": "SUCCESS",
  "msg": "Container stateful-session successfully called.",
  "result": {
    "execution-results": {
      "results": [],
      "facts": []
    }
  }
}
```

CompleteWorkItemCommand

KIE セッションでワークアイテムを完了します。

表24.11 コマンドの属性

表24.11 コマンドの属性

Name	説明	要件
workItemId	完了するワークアイテムの ID	必須
results	ワークアイテムの結果	任意

JSON リクエストボディの例

```
{
  "commands": [ {
    "complete-work-item": {
        "id": 1001
    }
  }
]
}
```

Java コマンドの例

```
CompleteWorkItemCommand completeWorkItemCommand = new
CompleteWorkItemCommand();
completeWorkItemCommand.setWorkItemId(1001);
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container employee-rostering successfully called.",
      "result": {
        "execution-results": {
          "results": [],
          "facts": []
        }
      }
    }
  ]
}
```

AbortWorkItemCommand

ksession.getWorkItemManager().abortWorkItem(workItemId) と同じ方法で KIE セッションのワークアイテムを中止します。

表24.12 コマンドの属性

Name	説明	要件
workItemId	中止するワークアイテムの ID	必須

JSON リクエストボディの例

```
{
  "commands": [ {
     "abort-work-item": {
       "id": 1001
     }
   }
  ]
}
```

Java コマンドの例

```
AbortWorkItemCommand abortWorkItemCommand = new AbortWorkItemCommand();
abortWorkItemCommand.setWorkItemId(1001);
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container employee-rostering successfully called.",
      "result": {
        "execution-results": {
          "results": [],
          "facts": []
        }
      }
    }
  ]
}
```

QueryCommand

KIE ベースで定義されたクエリーを実行します。

表24.13 コマンドの属性

Name	説明	要件
name	クエリー名。	必須
out-identifier	クエリー結果の ID。クエリー結果を実行結果に追加する際に、この ID を使用します。	任意
arguments	クエリーパラメーターとして渡されるオブジェクト一覧。	任意

JSON リクエストボディの例

```
{
```

```
  "commands": [
    {
      "query": {
        "name": "persons",
        "arguments": [],
        "out-identifier": "persons"
      }
    }
  ]
}
```

Java コマンドの例

```
QueryCommand queryCommand = new QueryCommand();
queryCommand.setName("persons");
queryCommand.setOutIdentifier("persons");
```

サーバーの応答例 (JSON)

```
{
  "type": "SUCCESS",
  "msg": "Container stateful-session successfully called.",
  "result": {
    "execution-results": {
      "results": [
        {
          "value": {
            "org.drools.core.runtime.rule.impl.FlatQueryResults": {
              "idFactHandleMaps": {
                "type": "LIST",
                "componentType": null,
                "element": [
                  {
                    "type": "MAP",
                    "componentType": null,
                    "element": [
                      {
                        "value": {
                          "org.drools.core.common.DisconnectedFactHandle": {
                            "id": 1,
                            "identityHashCode": 1809949690,
                            "objectHashCode": 1809949690,
                            "recency": 1,
                            "object": {
                              "org.kie.server.testing.Person": {
                                "fullname": "John Doe",
                                "age": 47
                              }
                            },
                            "entryPointId": "DEFAULT",
                            "traitType": "NON_TRAIT",
                            "external-form":
"0:1:1809949690:1809949690:1:DEFAULT:NON_TRAIT:org.kie.server.testing.Person"
                          }
                        },
```

```
                    "key": "$person"
                  }
                ]
              }
            ]
          },
          "idResultMaps": {
            "type": "LIST",
            "componentType": null,
            "element": [
              {
                "type": "MAP",
                "componentType": null,
                "element": [
                  {
                    "value": {
                      "org.kie.server.testing.Person": {
                        "fullname": "John Doe",
                        "age": 47
                      }
                    },
                    "key": "$person"
                  }
                ]
              }
            ]
          },
          "identifiers": {
            "type": "SET",
            "componentType": null,
            "element": [
              "$person"
            ]
          }
        }
      },
      "key": "persons"
    }
  ],
  "facts": []
 }
}
}
```


SetGlobalCommand

オブジェクトをグローバルステートに設定します。

表24.14 コマンドの属性

Name	説明	要件
identifier	KIE ベースで定義されるグローバル変数の ID	必須
object	グローバル変数に設定されるオブジェクト	任意

Name	説明	要件
out	設定したグローバル変数を実行結果から除外するブール値	任意
out-identifier	グローバル実行結果の ID	任意

JSON リクエストボディの例

```
{
  "commands": [
    {
      "set-global": {
        "identifier": "helper",
        "object": {
          "org.kie.server.testing.Person": {
            "fullname": "kyle",
            "age": 30
          }
        },
        "out-identifier": "output"
      }
    }
  ]
}
```

Java コマンドの例

```
SetGlobalCommand setGlobalCommand = new SetGlobalCommand();
setGlobalCommand.setIdentifier("helper");
setGlobalCommand.setObject(new Person("kyle", 30));
setGlobalCommand.setOut(true);
setGlobalCommand.setOutIdentifier("output");
```

サーバーの応答例 (JSON)

```
{
  "type": "SUCCESS",
  "msg": "Container stateful-session successfully called.",
  "result": {
    "execution-results": {
      "results": [
        {
          "value": {
            "org.kie.server.testing.Person": {
              "fullname": "kyle",
              "age": 30
            }
          },
          "key": "output"
        }
      ],
```

```
      "facts": []
    }
  }
}
```

GetGlobalCommand

以前に定義したグローバルオブジェクトを取得します。

表24.15 コマンドの属性

Name	説明	要件
identifier	KIE ベースで定義されるグローバル変数の ID	必須
out-identifier	実行結果で使用される ID	任意

JSON リクエストボディの例

```
{
  "commands": [ {
    "get-global": {
      "identifier": "helper",
      "out-identifier": "helperOutput"
    }
  }
 ]
}
```

Java コマンドの例

```
GetGlobalCommand getGlobalCommand = new GetGlobalCommand();
getGlobalCommand.setIdentifier("helper");
getGlobalCommand.setOutIdentifier("helperOutput");
```

サーバーの応答例 (JSON)

```
{
  "response": [
    {
      "type": "SUCCESS",
      "msg": "Container command-script-container successfully called.",
      "result": {
        "execution-results": {
          "results": [
            {
              "value": null,
              "key": "helperOutput"
            }
          ],
          "facts": []
        }
      }
```

```
    }
  ]
}
```

第25章 KIE SERVER テンプレートおよびインスタンス用の PROCESS AUTOMATION MANAGER コントローラー REST API

Red Hat Process Automation Manager は Process Automation Manager コントローラー REST API を提供し、これを使用することで Business Central ユーザーインターフェイスを使用せずに KIE Server のテンプレート (設定) や KIE Server インスタンス (リモートサーバー)、関連する KIE コンテナー (デプロイメントユニット) を操作することができます。この API のサポートにより、Red Hat Process Automation Manager サーバーおよびリソースをより効率的に維持でき、Red Hat Process Automation Manager の統合と開発を最適化できるようになります。

Process Automation Manager コントローラー REST API を使用すると、以下のアクションが可能になります。

- KIE Server テンプレート、インスタンス、および関連する KIE コンテナーに関する情報の取得
- KIE Server テンプレートおよびインスタンスに関連付けられた KIE コンテナーの更新、起動、または停止
- KIE Server テンプレートの作成、更新、または削除
- KIE Server インスタンスの作成、更新、または削除

Process Automation Manager コントローラー REST API への要求には、以下のコンポーネントが必要です。

認証

Process Automation Manager コントローラー REST API は、コントローラーのタイプによって、以下のユーザーロールに HTTP の Basic 認証またはトークンベースの認証を必要とします。

- Business Central をインストールしていて、ビルトインの Process Automation Manager コントローラーを使用する場合は、**rest-all** のユーザーロール。
- ヘッドレス Process Automation Manager コントローラーを Business Central とは別にインストールしている場合は、**kie-server** のユーザーロール。

お使いの Red Hat Process Automation Manager に設定されているユーザーロールを表示するには、**~/$SERVER_HOME/standalone/configuration/application-roles.properties** と **~/application-users.properties** に移動します。

ユーザーに **kie-server** ロールか **rest-all** ロール、もしくはそれら両方を追加するには、**~/$SERVER_HOME/bin** に移動し、ロールを指定して以下のコマンドを実行します。

```
$ ./bin/jboss-cli.sh --commands="embed-server --std-out=echo,/subsystem=elytron/filesystem-realm=ApplicationRealm:add-identity(identity=<USERNAME>),/subsystem=elytron/filesystem-realm=ApplicationRealm:set-password(identity=<USERNAME>, clear={password='<PASSWORD>'}),/subsystem=elytron/filesystem-realm=ApplicationRealm:add-identity-attribute(identity=<USERNAME>, name=role, value=['kie-server','rest-all'])"
```

Process Automation Manager コントローラーのアクセスで **kie-server** または **rest-all** ユーザーを設定するには、**~/$SERVER_HOME/standalone/configuration/standalone-full.xml** に移動し、(該当する場合は) **org.kie.server** プロパティーのコメントを解除して、コントローラーユーザーログイン認証情報とコントローラーの位置を (必要に応じて) 追加します。

```
<property name="org.kie.server.location" value="http://localhost:8080/kie-server/services/rest/server"/>
```

```
<property name="org.kie.server.controller" value="http://localhost:8080/business-
central/rest/controller"/>
<property name="org.kie.server.controller.user" value="baAdmin"/>
<property name="org.kie.server.controller.pwd" value="password@1"/>
<property name="org.kie.server.id" value="default-kieserver"/>
```

ユーザーロールと Red Hat Process Automation Manager のインストールオプションの詳細は、Red Hat Process Automation Manager インストールの計画 を参照してください。

HTTP ヘッダー

Process Automation Manager コントローラー REST API は、API 要求に以下の HTTP ヘッダーを必要とします。

- **Accept**: 要求元のクライアントが受け付けるデータ形式:
  - **application/json** (JSON)
  - **application/xml** (XML、JAXB 用)
- **Content-Type**: **POST** または **PUT** API 要求データ向けのデータ形式:
  - **application/json** (JSON)
  - **application/xml** (XML、JAXB 用)

HTTP メソッド

Process Automation Manager コントローラー REST API は、API 要求に以下の HTTP メソッドをサポートします。

- **GET**: 指定したリソースのエンドポイントから指定した情報を取得する
- **POST**: リソースまたはリソースインスタンスを更新する
- **PUT**: リソースまたはリソースインスタンスを作成する
- **DELETE**: リソースまたはリソースインスタンスを削除する

ベース URL

Process Automation Manager コントローラー REST API リクエストのベース URL は **http://SERVER:PORT/CONTROLLER/rest/** で、Business Central のビルトイン Process Automation Manager コントローラーを使用している場合は **http://localhost:8080/business-central/rest/** のようになります。

エンドポイント

指定した KIE Server テンプレートにおける **/controller/management/servers/{serverTemplateId}** などの Process Automation Manager コントローラー REST API のエンドポイントは、Process Automation Manager コントローラー REST API のベース URL に追記する URI で、Red Hat Process Automation Manager の対応するサーバーリソースやサーバーリソースのタイプにアクセスするためのものです。

/spaces/{serverTemplateId} エンドポイントの要求 URL 例

http://localhost:8080/business-central/rest/controller/management/servers/default-kieserver

要求パラメーターおよび要求データ

Process Automation Manager コントローラー REST API 要求のなかには、特定リソースを特定またはフィルターリングし、特定のアクションを実行するために、要求 URL パスで特定のパラメーターを必要とします。URL パラメーターは、**?<PARAM>=<VALUE>&<PARAM>=<VALUE>** の形式でエンドポイントに追記します。

DELETE 要求 URL のパラメーター例

http://localhost:8080/business-central/rest/controller/server/new-kieserver-instance?location=http://localhost:8080/kie-server/services/rest/server

HTTP **POST** と **PUT** の要求は、さらに要求のボディもしくはデータのあるファイルが必要になる場合があります。

PUT 要求 URL と JSON 要求のボディデータの例

http://localhost:8080/business-central/rest/controller/management/servers/new-kieserver

```
{
  "server-id": "new-kieserver",
  "server-name": "new-kieserver",
  "container-specs": [],
  "server-config": {},
  "capabilities": [
    "RULE",
    "PROCESS",
    "PLANNING"
  ]
}
```

25.1. REST クライアントまたは CURL ユーティリティーを使用した PROCESS AUTOMATION MANAGER コントローラー REST API による要求送信

Process Automation Manager コントローラーは REST API を提供し、これを使用することで Business Central ユーザーインターフェイスを使わずに KIE Server のテンプレート (設定) や KIE Server インスタンス (リモートサーバー)、関連する KIE コンテナー (デプロイメントユニット) を操作することができます。Process Automation Manager コントローラー REST API リクエストは、REST クライアントや curl ユーティリティーを使って送信することができます。

前提条件

- KIE Server をインストールし、実行している。
- Process Automation Manager コントローラーまたはヘッドレス Process Automation Manager コントローラーがインストールされ、実行中である。
- Business Central をインストールしている場合は Process Automation Manager コントローラーにアクセスする **rest-all** ユーザーロールがある。もしくは、Business Central とは別にインストールされたヘッドレス Process Automation Manager コントローラーにアクセスする **kie-server** ユーザーロールがある。

手順

1. 要求の送信先となる関連する API エンドポイント を特定します。Process Automation Manager コントローラーから KIE Server テンプレートを取得する **[GET] /controller/management/servers** などです。

2. REST クライアントまたは curl ユーティリティーで、**controller/management/servers** への **GET** 要求に以下のコンポーネントを記入します。ご自分のユースケースに合わせて、要求詳細を調整します。
REST クライアントの場合:

   - **Authentication**: **rest-all** ロールのある Process Automation Manager コントローラーユーザーまたは **kie-server** ロールを持つヘッドレス Process Automation Manager コントローラーユーザーのユーザー名とパスワードを入力します。
   - **HTTP Headers**: 以下のヘッダーを設定します。
     - **Accept**: **application/json**
   - **HTTP method**: **GET** に設定します。
   - **URL**: Process Automation Manager コントローラー REST API ベース URL とエンドポイントを入力します。たとえば、**http://localhost:8080/business-central/rest/controller/management/servers** となります。

   curl ユーティリティーの場合:

   - **-u**: **rest-all** ロールのある Process Automation Manager コントローラーユーザーまたは **kie-server** ロールを持つヘッドレス Process Automation Manager コントローラーユーザーのユーザー名およびパスワードを入力します。
   - **-H**: 以下のヘッダーを設定します。
     - **Accept**: **application/json**
   - **-X**: **GET** に設定します。
   - **URL**: Process Automation Manager コントローラー REST API ベース URL とエンドポイントを入力します。たとえば、**http://localhost:8080/business-central/rest/controller/management/servers** となります。

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -X GET
"http://localhost:8080/business-central/rest/controller/management/servers"
```

3. 要求を実行し、Process Automation Manager コントローラーの応答を確認します。
サーバー応答の例 (JSON):

```
{
  "server-template": [
    {
      "server-id": "default-kieserver",
      "server-name": "default-kieserver",
      "container-specs": [
        {
          "container-id": "employeerostering_1.0.0-SNAPSHOT",
          "container-name": "employeerostering",
          "server-template-key": {
            "server-id": "default-kieserver",
```

```
        "server-name": "default-kieserver"
      },
      "release-id": {
        "group-id": "employeerostering",
        "artifact-id": "employeerostering",
        "version": "1.0.0-SNAPSHOT"
      },
      "configuration": {
        "RULE": {
          "org.kie.server.controller.api.model.spec.RuleConfig": {
            "pollInterval": null,
            "scannerStatus": "STOPPED"
          }
        },
        "PROCESS": {
          "org.kie.server.controller.api.model.spec.ProcessConfig": {
            "runtimeStrategy": "SINGLETON",
            "kbase": "",
            "ksession": "",
            "mergeMode": "MERGE_COLLECTIONS"
          }
        }
      },
      "status": "STARTED"
    },
    {
      "container-id": "mortgage-process_1.0.0-SNAPSHOT",
      "container-name": "mortgage-process",
      "server-template-key": {
        "server-id": "default-kieserver",
        "server-name": "default-kieserver"
      },
      "release-id": {
        "group-id": "mortgage-process",
        "artifact-id": "mortgage-process",
        "version": "1.0.0-SNAPSHOT"
      },
      "configuration": {
        "RULE": {
          "org.kie.server.controller.api.model.spec.RuleConfig": {
            "pollInterval": null,
            "scannerStatus": "STOPPED"
          }
        },
        "PROCESS": {
          "org.kie.server.controller.api.model.spec.ProcessConfig": {
            "runtimeStrategy": "PER_PROCESS_INSTANCE",
            "kbase": "",
            "ksession": "",
            "mergeMode": "MERGE_COLLECTIONS"
          }
        }
      },
      "status": "STARTED"
    }
  ],
```

```
      "server-config": {},
      "server-instances": [
        {
          "server-instance-id": "default-kieserver-instance@localhost:8080",
          "server-name": "default-kieserver-instance@localhost:8080",
          "server-template-id": "default-kieserver",
          "server-url": "http://localhost:8080/kie-server/services/rest/server"
        }
      ],
      "capabilities": [
        "RULE",
        "PROCESS",
        "PLANNING"
      ]
    }
  ]
}
```

4. REST クライアントまたは curl ユーティリティーで、**/controller/management/servers/{serverTemplateId}** への **PUT** 要求を以下のコンポーネントで送信し、新規の KIE Server テンプレートを作成します。ご自分のユースケースに合わせて、要求詳細を調整します。
REST クライアントの場合:

   - **Authentication**: **rest-all** ロールのある Process Automation Manager コントローラーユーザーまたは **kie-server** ロールを持つヘッドレス Process Automation Manager コントローラーユーザーのユーザー名とパスワードを入力します。
   - **HTTP Headers**: 以下のヘッダーを設定します。
     - **Accept**: **application/json**
     - **Content-Type**: **application/json**
   - **HTTP method**: **PUT** に設定します。
   - **URL**: Process Automation Manager コントローラー REST API ベース URL およびエンドポイントを入力します。たとえば、**http://localhost:8080/business-central/rest/controller/management/servers/new-kieserver** となります。
   - **要求のボディ**: 新規 KIE Server テンプレート用の設定を含めて JSON 要求のボディを追加します。

```
{
  "server-id": "new-kieserver",
  "server-name": "new-kieserver",
  "container-specs": [],
  "server-config": {},
  "capabilities": [
    "RULE",
    "PROCESS",
    "PLANNING"
  ]
}
```

   curl ユーティリティーの場合:

- **-u**: **rest-all** ロールのある Process Automation Manager コントローラーユーザーまたは **kie-server** ロールを持つヘッドレス Process Automation Manager コントローラーユーザーのユーザー名およびパスワードを入力します。
- **-H**: 以下のヘッダーを設定します。
  - **Accept**: **application/json**
  - **Content-Type**: **application/json**
- **-X**: **PUT** に設定します。
- **URL**: Process Automation Manager コントローラー REST API ベース URL およびエンドポイントを入力します。たとえば、**http://localhost:8080/business-central/rest/controller/management/servers/new-kieserver** となります。
- **-d**: 新規 KIE Server テンプレート用の設定を含めて JSON 要求のボディまたはファイル (**@file.json**) を追加します。

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Content-Type: application/json" -X PUT "http://localhost:8080/business-central/rest/controller/management/servers/new-kieserver" -d "{ \"server-id\": \"new-kieserver\", \"server-name\": \"new-kieserver\", \"container-specs\": [], \"server-config\": {}, \"capabilities\": [ \"RULE\", \"PROCESS\", \"PLANNING\" ]}"
```

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Content-Type: application/json" -X PUT "http://localhost:8080/business-central/rest/controller/management/servers/new-kieserver" -d @my-server-template-configs.json
```

5. 要求を実行し、Process Automation Manager コントローラーの応答が正常であることを確認します。
   要求エラーが発生した場合は、返されたエラーコードメッセージを確認して、それに応じて要求を調整します。

25.2. SWAGGER インターフェイスを使用した PROCESS AUTOMATION MANAGER コントローラー REST API による要求送信

Process Automation Manager コントローラー REST API は Swagger Web インターフェイスをサポートしています。スタンドアロンの REST クライアントや curl ユーティリティーの代わりのこれを使用すると、Business Central ユーザーインターフェイスを使わずに KIE Server の テンプレート、インスタンス、関連する KIE コンテナーを Red Hat Process Automation Manager で操作することができます。

注記

デフォルトでは、**org.kie.workbench.swagger.disabled=false** のシステムプロパティーが指定されており、Process Automation Manager の Swagger Web インターフェイスが有効になっています。Process Automation Manager で Swagger Web インターフェイスを無効にするには、このシステムプロパティーを **true** に設定してください。

前提条件

- Process Automation Manager コントローラーがインストールされ、実行中である。

- Business Central をインストールしている場合は Process Automation Manager コントローラーにアクセスする **rest-all** ユーザーロールがある。もしくは、Business Central とは別にインストールされたヘッドレス Process Automation Manager コントローラーにアクセスする **kie-server** ユーザーロールがある。

手順

1. Web ブラウザーで **http://SERVER:PORT/CONTROLLER/docs** に移動します。たとえば、**http://localhost:8080/business-central/docs** などです。**rest-all** ロールのある Process Automation Manager コントローラーユーザー、または **kie-server** ロールを持つヘッドレス Process Automation Manager コントローラーユーザーのユーザー名とパスワードでログインします。

注記

Business Central に組み込まれている Process Automation Manager コントローラーを使用している場合、Process Automation Manager コントローラーに関連付けられている Swagger ページは、Business Central REST サービスでは Business Central API として特定されます。Business Central なしでヘッドレス Process Automation Manager コントローラーを使用している場合は、ヘッドレス Process Automation Manager コントローラーに関連付けられている Swagger ページは、Controller API と特定されます。いずれの場合も、Process Automation Manager コントローラー REST API エンドポイントは、同じです。

2. Swagger ページで、要求の送信先となる関連 API エンドポイントを選択します。たとえば、**Controller :: KIE Server templates and KIE containers↪ [GET] /controller/management/servers** で KIE Server テンプレートを Process Automation Manager コントローラーから取得します。

3. 該当する場合は **Try it out** をクリックして、結果のフィルターリングに使用する任意のパラメーターを提供します。

4. **Response content type** ドロップダウンメニューで、サーバー応答のフォーマットを選択します (例: JSON フォーマットでは **application/json**)。

5. **Execute** をクリックし、KIE Server の応答を確認します。
サーバー応答の例 (JSON):


```
{
  "server-template": [
    {
      "server-id": "default-kieserver",
      "server-name": "default-kieserver",
      "container-specs": [
        {
          "container-id": "employeerostering_1.0.0-SNAPSHOT",
          "container-name": "employeerostering",
          "server-template-key": {
            "server-id": "default-kieserver",
            "server-name": "default-kieserver"
          },
          "release-id": {
            "group-id": "employeerostering",
            "artifact-id": "employeerostering",
            "version": "1.0.0-SNAPSHOT"
```

```
      },
      "configuration": {
        "RULE": {
          "org.kie.server.controller.api.model.spec.RuleConfig": {
            "pollInterval": null,
            "scannerStatus": "STOPPED"
          }
        },
        "PROCESS": {
          "org.kie.server.controller.api.model.spec.ProcessConfig": {
            "runtimeStrategy": "SINGLETON",
            "kbase": "",
            "ksession": "",
            "mergeMode": "MERGE_COLLECTIONS"
          }
        }
      },
      "status": "STARTED"
    },
    {
      "container-id": "mortgage-process_1.0.0-SNAPSHOT",
      "container-name": "mortgage-process",
      "server-template-key": {
        "server-id": "default-kieserver",
        "server-name": "default-kieserver"
      },
      "release-id": {
        "group-id": "mortgage-process",
        "artifact-id": "mortgage-process",
        "version": "1.0.0-SNAPSHOT"
      },
      "configuration": {
        "RULE": {
          "org.kie.server.controller.api.model.spec.RuleConfig": {
            "pollInterval": null,
            "scannerStatus": "STOPPED"
          }
        },
        "PROCESS": {
          "org.kie.server.controller.api.model.spec.ProcessConfig": {
            "runtimeStrategy": "PER_PROCESS_INSTANCE",
            "kbase": "",
            "ksession": "",
            "mergeMode": "MERGE_COLLECTIONS"
          }
        }
      },
      "status": "STARTED"
    }
  ],
  "server-config": {},
  "server-instances": [
    {
      "server-instance-id": "default-kieserver-instance@localhost:8080",
      "server-name": "default-kieserver-instance@localhost:8080",
      "server-template-id": "default-kieserver",
```

```
          "server-url": "http://localhost:8080/kie-server/services/rest/server"
        }
       ],
       "capabilities": [
        "RULE",
        "PROCESS",
        "PLANNING"
       ]
      }
     ]
}
```

6. Swagger ページで **Controller :: KIE Server templates and KIE containers↬ [GET] /controller/management/servers/{serverTemplateId}** エンドポイントに移動し、新たな KIE Server テンプレートを作成するための別の要求を送信します。ご自分のユースケースに合わせて、要求詳細を調整します。

7. **Try it out** をクリックして、以下の要求のコンポーネントを入力します。

   - **serverTemplateId**: 新規 KIE Server テンプレートの ID を入力します (例: **new-kieserver**)。
   - **body**: **Parameter content type** を任意の要求のボディ形式 (JSON の場合は **application/json** など) に設定し、要求のボディに新規 KIE Server テンプレートの設定を追加します。

```
{
 "server-id": "new-kieserver",
 "server-name": "new-kieserver",
 "container-specs": [],
 "server-config": {},
 "capabilities": [
  "RULE",
  "PROCESS",
  "PLANNING"
 ]
}
```

8. **Response content type** ドロップダウンメニューで、サーバー応答のフォーマットを選択します (例: JSON フォーマットでは **application/json**)。

9. **Execute** をクリックし、Process Automation Manager コントローラーの応答が正常であることを確認します。
   要求エラーが発生した場合は、返されたエラーコードメッセージを確認して、それに応じて要求を調整します。

25.3. サポート対象の PROCESS AUTOMATION MANAGER コントローラー REST API エンドポイント

Process Automation Manager コントローラー REST API はエンドポイントを提供し、これを使用することで KIE Server の テンプレート (設定) や KIE Server インスタンス (リモートサーバー)、関連する KIE コンテナー (デプロイメントユニット) を操作することができます。Process Automation Manager コントローラー REST API のベース URL は、**http://SERVER:PORT/CONTROLLER/rest/** です。Business Central がインストール済みでビルトインの Process Automation Manager コントローラーを

使用する場合の **rest-all** ユーザーロール、もしくは Business Central とは別にヘッドレス Process Automation Manager コントローラーをインストール済みの場合の **kie-server** ユーザーロールでは、すべてのリクエストでは HTTP Basic 認証もしくはトークンベースの認証が必要です。

Process Automation Manager コントローラー REST API エンドポイントの完全一覧と説明は、以下のリソースを参照してください。

- jBPM ドキュメントページ (静的) の Controller REST API
- **http://SERVER:PORT/CONTROLLER/docs** (動的。稼働中の Process Automation Manager コントローラーが必要) ページの Process Automation Manager コントローラー REST API 用 Swagger UI

注記

デフォルトでは、**org.kie.workbench.swagger.disabled=false** のシステムプロパティーが指定されており、Process Automation Manager の Swagger Web インターフェイスが有効になっています。Process Automation Manager で Swagger Web インターフェイスを無効にするには、このシステムプロパティーを **true** に設定してください。

Business Central に組み込まれている Process Automation Manager コントローラーを使用している場合、Process Automation Manager コントローラーに関連付けられている Swagger ページは、Business Central REST サービスでは Business Central API として特定されます。Business Central なしでヘッドレス Process Automation Manager コントローラーを使用している場合は、ヘッドレス Process Automation Manager コントローラーに関連付けられている Swagger ページは、Controller API と特定されます。いずれの場合も、Process Automation Manager コントローラー REST API エンドポイントは、同じです。

第26章 KIE SERVER テンプレートおよびインスタンス用の PROCESS AUTOMATION MANAGER コントローラー JAVA クライアント API

Red Hat Process Automation Manager は Process Automation Manager コントローラー Java クライアント API を提供し、これを使用することで Java クライアントアプリケーションから REST もしくは WebSocket プロトコルを使用して Process Automation Manager コントローラーに接続できるようになります。Process Automation Manager コントローラー REST API の代わりに Process Automation Manager コントローラー Java クライアント API を使用して、Business Central ユーザーインターフェイスを使わずに Red Hat Process Automation Manager で KIE Server テンプレート (設定)、KIE Server のインスタンス (リモートサーバー)、関連する KIE コンテナー (デプロイメントユニット) を操作することができます。この API のサポートにより、Red Hat Process Automation Manager サーバーおよびリソースをより効率的に維持でき、Red Hat Process Automation Manager の統合と開発を最適化できるようになります。

Process Automation Manager コントローラー Java クライアント API を使用すると、Process Automation Manager コントローラー REST API でもサポートされている以下のアクションが実行可能になります。

- KIE Server テンプレート、インスタンス、および関連する KIE コンテナーに関する情報の取得
- KIE Server テンプレートおよびインスタンスに関連付けられた KIE コンテナーの更新、起動、または停止
- KIE Server テンプレートの作成、更新、または削除
- KIE Server インスタンスの作成、更新、または削除

Process Automation Manager コントローラー Java クライアント API 要求には以下のコンポーネントが必要です。

認証

Process Automation Manager コントローラー Java クライアント API は、コントローラーのタイプによって、以下のユーザーロールに HTTP の Basic 認証を必要とします。

- Business Central をインストールしていて、ビルトインの Process Automation Manager コントローラーを使用する場合は、**rest-all** のユーザーロール。
- ヘッドレス Process Automation Manager コントローラーを Business Central とは別にインストールしている場合は、**kie-server** のユーザーロール。

お使いの Red Hat Process Automation Manager に設定されているユーザーロールを表示するには、**~/$SERVER_HOME/standalone/configuration/application-roles.properties** と **~/application-users.properties** に移動します。

ユーザーに **kie-server** ロールか **rest-all** ロール、もしくはそれら両方を追加するには (Keystore がすでに設定されていると仮定)、**~/$SERVER_HOME/bin** に移動し、ロールを指定して以下のコマンドを実行します。

```
$ ./bin/jboss-cli.sh --commands="embed-server --std-out=echo,/subsystem=elytron/filesystem-realm=ApplicationRealm:add-identity(identity=<USERNAME>),/subsystem=elytron/filesystem-realm=ApplicationRealm:set-password(identity=<USERNAME>, clear={password='<PASSWORD>'}),/subsystem=elytron/filesystem-realm=ApplicationRealm:add-identity-attribute(identity=<USERNAME>, name=role, value=['rest-all','kie-server'])"
```

キーストアが設定されていない場合は、以下のコマンドを実行してキーストアを作成します。

```
$ keytool -importpassword -keystore
$SERVER_HOME/standalone/configuration/kie_keystore.jceks -keypass
<SECRETKEYPASSWORD> -alias kieserver -storepass <SECRETSTOREPASSWORD> -
storetype JCEKS
```

また、~/**$SERVER_HOME/standalone/configuration/standalone-full.xml** に以下のプロパティーを追加します。

```
    <property name="kie.keystore.keyStoreURL"
value="file:///data/jboss/rhpam780/standalone/configuration/kie_keystore.jceks"/>
    <property name="kie.keystore.keyStorePwd" value="<SECRETSTOREPASSWORD>"/>
    <property name="kie.keystore.key.server.alias" value="kieserver"/>
    <property name="kie.keystore.key.server.pwd" value="<SECRETKEYPASSWORD>"/>
    <property name="kie.keystore.key.ctrl.alias" value="kieserver"/>
    <property name="kie.keystore.key.ctrl.pwd" value="<SECRETKEYPASSWORD>"/>
```

Process Automation Manager コントローラーのアクセスで **kie-server** または **rest-all** ユーザーを設定するには、~/**$SERVER_HOME/standalone/configuration/standalone-full.xml** に移動し、(該当する場合は) **org.kie.server** プロパティーのコメントを解除して、コントローラーユーザーログイン認証情報とコントローラーの位置を (必要に応じて) 追加します。

```
<property name="org.kie.server.location" value="http://localhost:8080/kie-
server/services/rest/server"/>
<property name="org.kie.server.controller" value="http://localhost:8080/business-
central/rest/controller"/>
<property name="org.kie.server.controller.user" value="<USERNAME>"/>
<property name="org.kie.server.id" value="default-kieserver"/>
```

ユーザーロールと Red Hat Process Automation Manager のインストールオプションの詳細は、**Red Hat Process Automation Manager インストールの計画** を参照してください。

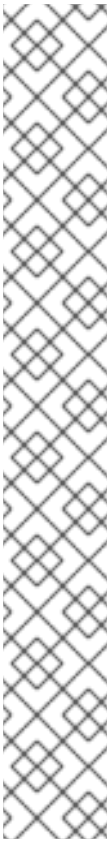
プロジェクトの依存関係

Process Automation Manager コントローラー Java クライアント API は、Java プロジェクトの関連するクラスパスで、以下の依存関係を必要とします。

```
<!-- For remote execution on controller -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-controller-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- For REST client -->
<dependency>
  <groupId>org.jboss.resteasy</groupId>
  <artifactId>resteasy-client</artifactId>
  <version>${resteasy.version}</version>
</dependency>

<!-- For WebSocket client -->
<dependency>
  <groupId>io.undertow</groupId>
```

```
  <artifactId>undertow-websockets-jsr</artifactId>
  <version>${undertow.version}</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>${logback.version}</version>
</dependency>
```

Red Hat Process Automation Manager 依存関係の **<version>** は、プロジェクトで現在使用している Red Hat Process Automation Manager の Maven アーティファクトバージョンです (例: 7.59.0.Final-redhat-00006)。



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 BOM (bill of materials) 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 Decision Manager と Red Hat Process Automation Manager の両方に適用されます。BOM ファイルを追加すると、提供される Maven リポジトリーから、推移的依存関係の適切なバージョンがプロジェクトに含められます。

BOM 依存関係の例:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.12.0.redhat-00008</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。

クライアント要求の設定

Process Automation Manager コントローラー Java クライアント API による Java クライアント要求はすべて、少なくとも以下のコントローラー通信コンポーネントを定義する必要があります。

- Business Central をインストールしている場合は **rest-all** ユーザーの認証情報、またはヘッドレス Process Automation Manager コントローラーを Business Central とは別にインストールしている場合は **kie-server** のユーザーの認証情報。
- REST もしくは WebSocket プロトコル用 Process Automation Manager コントローラーの場所。
  - REST URL の例: **http://localhost:8080/business-central/rest/controller**
  - WebSocket URL の例: **ws://localhost:8080/headless-controller/websocket/controller**
- API 要求および応答のマーシャリングフォーマット (JSON または JAXB)

- **KieServerControllerClient** オブジェクト。これは、Java クライアント API を使用してサーバー通信を開始するためのエントリーポイントのロールを果たします。
- REST プロトコルまたは WebSocket プロトコルおよびユーザーアクセスを定義する **KieServerControllerClientFactory**。
- 使用される Process Automation Manager コントローラークライアントサービス (**listServerTemplates**、**getServerTemplate**、**getServerInstances** など)。

以下は、これらのコンポーネントを使用した REST および WebSocket クライアントの設定例です。

REST によるクライアント設定例

```
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.controller.api.model.spec.ServerTemplateList;
import org.kie.server.controller.client.KieServerControllerClient;
import org.kie.server.controller.client.KieServerControllerClientFactory;

public class ListServerTemplatesExample {

    private static final String URL = "http://localhost:8080/business-central/rest/controller";
    private static final String USER = "baAdmin";
    private static final String PASSWORD = "password@1";

    private static final MarshallingFormat FORMAT = MarshallingFormat.JSON;

    public static void main(String[] args) {
        KieServerControllerClient client = KieServerControllerClientFactory.newRestClient(URL,
                                                                                           USER,
                                                                                           PASSWORD);

        final ServerTemplateList serverTemplateList = client.listServerTemplates();
        System.out.println(String.format("Found %s server template(s) at controller url: %s",
                                         serverTemplateList.getServerTemplates().length,
                                         URL));
    }
}
```

WebSocket によるクライアント設定例

```
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.controller.api.model.spec.ServerTemplateList;
import org.kie.server.controller.client.KieServerControllerClient;
import org.kie.server.controller.client.KieServerControllerClientFactory;

public class ListServerTemplatesExample {

    private static final String URL = "ws://localhost:8080/my-controller/websocket/controller";
    private static final String USER = "baAdmin";
    private static final String PASSWORD = "password@1";

    private static final MarshallingFormat FORMAT = MarshallingFormat.JSON;

    public static void main(String[] args) {
        KieServerControllerClient client =
```

```
KieServerControllerClientFactory.newWebSocketClient(URL,
                                                    USER,
                                                    PASSWORD);

        final ServerTemplateList serverTemplateList = client.listServerTemplates();
        System.out.println(String.format("Found %s server template(s) at controller url: %s",
                                         serverTemplateList.getServerTemplates().length,
                                         URL));
    }
}
```

26.1. PROCESS AUTOMATION MANAGER コントローラー JAVA クライアント API を使用した要求送信

Process Automation Manager コントローラー Java クライアント API を使用すると、Java クライアントアプリケーションから REST もしくは WebSocket プロトコルを使用して Process Automation Manager コントローラー に接続できるようになります。Process Automation Manager コントローラー REST API の代わりに Process Automation Manager コントローラー Java クライアント API を使用して、Business Central ユーザーインターフェイスを使わずに Red Hat Process Automation Manager で KIE Server テンプレート (設定)、KIE Server のインスタンス (リモートサーバー)、関連する KIE コンテナー (デプロイメントユニット) を操作することができます。

前提条件

- KIE Server をインストールし、実行している。
- Process Automation Manager コントローラーまたはヘッドレス Process Automation Manager コントローラーがインストールされ、実行中である。
- Business Central をインストールしている場合は Process Automation Manager コントローラーにアクセスする **rest-all** ユーザーロールがある。もしくは、Business Central とは別にインストールされたヘッドレス Process Automation Manager コントローラーにアクセスする **kie-server** ユーザーロールがある。
- Red Hat Process Automation Manager リソースを使った Java プロジェクトがある。

手順

1. クライアントアプリケーションで、Java プロジェクトの関連クラスパスに以下の依存関係が追加されていることを確認します。

```
<!-- For remote execution on controller -->
<dependency>
  <groupId>org.kie.server</groupId>
  <artifactId>kie-server-controller-client</artifactId>
  <version>${rhpam.version}</version>
</dependency>

<!-- For REST client -->
<dependency>
  <groupId>org.jboss.resteasy</groupId>
  <artifactId>resteasy-client</artifactId>
  <version>${resteasy.version}</version>
</dependency>
```

```
<!-- For WebSocket client -->
<dependency>
  <groupId>io.undertow</groupId>
  <artifactId>undertow-websockets-jsr</artifactId>
  <version>${undertow.version}</version>
</dependency>

<!-- For debug logging (optional) -->
<dependency>
  <groupId>ch.qos.logback</groupId>
  <artifactId>logback-classic</artifactId>
  <version>${logback.version}</version>
</dependency>
```

2. Red Hat カスタマーポータル から **Red Hat Process Automation Manager 7.12.0 Source Distribution** をダウンロードし、**~/rhpam-7.12.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-controller/kie-server-controller-client/src/main/java/org/kie/server/controller/client** に移動して Process Automation Manager コントローラー Java クライアントにアクセスします。

3. **~/kie/server/controller/client** ディレクトリーで、KIE Server テンプレートや KIE コンテナーが REST プロトコルでクライアントサービスにアクセスするために、**RestKieServerControllerClient** 実装などの送信する要求用の関連 Java クライアントを特定します。

4. クライアントアプリケーションで、API 要求用の **.java** クラスを作成します。クラスには、Process Automation Manager コントローラーの場所とユーザー認証情報、**KieServerControllerClient** オブジェクト、実行するクライアントメソッド (**RestKieServerControllerClient** 実装からの **createServerTemplate** や **createContainer** など) を含める必要があります。ご自分のユースケースに合わせて、設定詳細を調整します。

KIE Server テンプレートおよび KIE コンテナーの作成と対話

```
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.KieContainerStatus;
import org.kie.server.api.model.KieScannerStatus;
import org.kie.server.api.model.ReleaseId;
import org.kie.server.controller.api.model.spec.*;
import org.kie.server.controller.client.KieServerControllerClient;
import org.kie.server.controller.client.KieServerControllerClientFactory;

public class RestTemplateContainerExample {

  private static final String URL = "http://localhost:8080/business-central/rest/controller";
  private static final String USER = "baAdmin";
  private static final String PASSWORD = "password@1";

    private static KieServerControllerClient client;

    public static void main(String[] args) {
        KieServerControllerClient client = KieServerControllerClientFactory.newRestClient(URL,
```

```
                                                    USER,
                                                    PASSWORD,
                                                    MarshallingFormat.JSON);
        // Create server template and KIE container, start and stop KIE container, and delete
server template
        ServerTemplate serverTemplate = createServerTemplate();
        ContainerSpec container = createContainer(serverTemplate);
        client.startContainer(container);
        client.stopContainer(container);
        client.deleteServerTemplate(serverTemplate.getId());
    }

    // Re-create and configure server template
    protected static ServerTemplate createServerTemplate() {
        ServerTemplate serverTemplate = new ServerTemplate();
        serverTemplate.setId("example-client-id");
        serverTemplate.setName("example-client-name");
        serverTemplate.setCapabilities(Arrays.asList(Capability.PROCESS.name(),
                                        Capability.RULE.name(),
                                        Capability.PLANNING.name()));

        client.saveServerTemplate(serverTemplate);

        return serverTemplate;
    }

    // Re-create and configure KIE containers
    protected static ContainerSpec createContainer(ServerTemplate serverTemplate){
        Map<Capability, ContainerConfig> containerConfigMap = new HashMap();

        ProcessConfig processConfig = new ProcessConfig("PER_PROCESS_INSTANCE",
"kieBase", "kieSession", "MERGE_COLLECTION");
        containerConfigMap.put(Capability.PROCESS, processConfig);

        RuleConfig ruleConfig = new RuleConfig(500l, KieScannerStatus.SCANNING);
        containerConfigMap.put(Capability.RULE, ruleConfig);

        ReleaseId releaseId = new ReleaseId("org.kie.server.testing", "stateless-session-kjar",
"1.0.0-SNAPSHOT");

        ContainerSpec containerSpec = new ContainerSpec("example-container-id", "example-
client-name", serverTemplate, releaseId, KieContainerStatus.STOPPED,
containerConfigMap);
        client.saveContainerSpec(serverTemplate.getId(), containerSpec);

        return containerSpec;
    }
}
```

5. 設定済み **.java** クラスをプロジェクトディレクトリーから実行して要求を実行し、Process Automation Manager コントローラーの応答を確認します。
デバッグのロギングが有効になっている場合は、JSON などの設定済みマーシャリングフォーマットに従って、KIE Server が詳細を返します。エラーが発生した場合は、返されたエラーメッセージを確認して、それに応じて Java 設定を調整します。

26.2. サポート対象の PROCESS AUTOMATION MANAGER コントローラー JAVA クライアント

以下は、Red Hat Process Automation Manager ディストリビューションの **org.kie.server.controller.client** パッケージで利用可能な Java クライアントサービスの一部です。これらのサービスを使用して、Process Automation Manager コントローラー REST API と同様に Process Automation Manager コントローラーの関連リソースを操作できます。

- **KieServerControllerClient**: Process Automation Manager コントローラーとの通信のエントリーポイントとして使用します。
- **RestKieServerControllerClient**: REST プロトコルでの KIE Server テンプレートと KIE コンテナーとの対話に使用される実装 (~/**org/kie/server/controller/client/rest** に格納)。
- **WebSocketKieServerControllerClient**: WebSocket プロトコルでの KIE Server テンプレートと KIE コンテナーとの対話に使用される実装 (~/**org/kie/server/controller/client/websocket** に格納)。

利用可能な Process Automation Manager コントローラーの Java クライアントの完全一覧については、Red Hat カスタマーポータル から **Red Hat Process Automation Manager 7.12.0 Source Distribution** をダウンロードし、~/**rhpam-7.12.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-controller/kie-server-controller-client/src/main/java/org/kie/server/controller/client** に移動します。

26.3. PROCESS AUTOMATION MANAGER コントローラー JAVA クライアント API を使った要求例

以下は、Process Automation Manager コントローラーと基本的な対話を行う Process Automation Manager コントローラーの Java クライアント API 要求例です。利用可能な Process Automation Manager コントローラーの Java クライアントの完全一覧については、Red Hat カスタマーポータル から **Red Hat Process Automation Manager 7.12.0 Source Distribution**をダウンロードし、~/**rhpam-7.12.0-sources/src/droolsjbpm-integration-$VERSION/kie-server-parent/kie-server-controller/kie-server-controller-client/src/main/java/org/kie/server/controller/client** に移動します。

KIE Server テンプレートおよび KIE コンテナーの作成と対話

REST または WebSocket の Process Automation Manager コントローラークライアントで **ServerTemplate** サービスおよび **ContainerSpec** サービスを使用すると、KIE Server テンプレートと KIE コンテナーの作成、破棄、更新が可能で、さらに KIE コンテナーの起動と停止もできます。以下に例を示します。

KIE Server テンプレートおよび KIE コンテナーによる作成と対話要求の例

```
import java.util.Arrays;
import java.util.HashMap;
import java.util.Map;

import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.api.model.KieContainerStatus;
import org.kie.server.api.model.KieScannerStatus;
import org.kie.server.api.model.ReleaseId;
import org.kie.server.controller.api.model.spec.*;
import org.kie.server.controller.client.KieServerControllerClient;
import org.kie.server.controller.client.KieServerControllerClientFactory;
```

```
public class RestTemplateContainerExample {

  private static final String URL = "http://localhost:8080/business-central/rest/controller";
  private static final String USER = "baAdmin";
  private static final String PASSWORD = "password@1";

    private static KieServerControllerClient client;

    public static void main(String[] args) {
        KieServerControllerClient client = KieServerControllerClientFactory.newRestClient(URL,
                                                                                           USER,
                                                                                           PASSWORD,
                                                                                           MarshallingFormat.JSON);
        // Create server template and KIE container, start and stop KIE container, and delete server
template
        ServerTemplate serverTemplate = createServerTemplate();
        ContainerSpec container = createContainer(serverTemplate);
        client.startContainer(container);
        client.stopContainer(container);
        client.deleteServerTemplate(serverTemplate.getId());
    }

    // Re-create and configure server template
    protected static ServerTemplate createServerTemplate() {
        ServerTemplate serverTemplate = new ServerTemplate();
        serverTemplate.setId("example-client-id");
        serverTemplate.setName("example-client-name");
        serverTemplate.setCapabilities(Arrays.asList(Capability.PROCESS.name(),
                                       Capability.RULE.name(),
                                       Capability.PLANNING.name()));

        client.saveServerTemplate(serverTemplate);

        return serverTemplate;
    }

    // Re-create and configure KIE containers
    protected static ContainerSpec createContainer(ServerTemplate serverTemplate){
        Map<Capability, ContainerConfig> containerConfigMap = new HashMap();

        ProcessConfig processConfig = new ProcessConfig("PER_PROCESS_INSTANCE",
"kieBase", "kieSession", "MERGE_COLLECTION");
        containerConfigMap.put(Capability.PROCESS, processConfig);

        RuleConfig ruleConfig = new RuleConfig(500l, KieScannerStatus.SCANNING);
        containerConfigMap.put(Capability.RULE, ruleConfig);

        ReleaseId releaseId = new ReleaseId("org.kie.server.testing", "stateless-session-kjar",
"1.0.0-SNAPSHOT");

        ContainerSpec containerSpec = new ContainerSpec("example-container-id", "example-client-
name", serverTemplate, releaseId, KieContainerStatus.STOPPED, containerConfigMap);
        client.saveContainerSpec(serverTemplate.getId(), containerSpec);
```

```
        return containerSpec;
    }
}
```

KIE Server テンプレートの一覧および接続タイムアウトの指定 (REST)

Process Automation Manager コントローラー Java クライアント API リクエストに REST プロトコルを使用すると、独自の **javax.ws.rs.core.Configuration** 仕様で接続タイムアウトなどの基本的な REST クライアント API を変更することができます。

サーバーテンプレートを返し、接続タイムアウトを指定する REST 要求の例

```
import java.util.concurrent.TimeUnit;
import javax.ws.rs.core.Configuration;
import org.jboss.resteasy.client.jaxrs.ResteasyClientBuilder;

import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.controller.api.model.spec.ServerTemplateList;
import org.kie.server.controller.client.KieServerControllerClient;
import org.kie.server.controller.client.KieServerControllerClientFactory;

public class RESTTimeoutExample {

  private static final String URL = "http://localhost:8080/business-central/rest/controller";
  private static final String USER = "baAdmin";
  private static final String PASSWORD = "password@1";

  public static void main(String[] args) {

    // Specify connection timeout
    final Configuration configuration =
          new ResteasyClientBuilder()
                .establishConnectionTimeout(10,
                                            TimeUnit.SECONDS)
                .socketTimeout(60,
                               TimeUnit.SECONDS)
                 .getConfiguration();
      KieServerControllerClient client = KieServerControllerClientFactory.newRestClient(URL,
                                                                                         USER,
                                                                                         PASSWORD,
                                                                                         MarshallingFormat.JSON,
                                                                                         configuration);

      // Retrieve list of server templates
      final ServerTemplateList serverTemplateList = client.listServerTemplates();
      System.out.println(String.format("Found %s server template(s) at controller url: %s",
                                       serverTemplateList.getServerTemplates().length,
                                       URL));
    }
}
```

KIE Server テンプレートの一覧およびイベント通知の指定 (WebSocket)

Process Automation Manager コントローラー Java クライアント API リクエストに WebSocket プロトコルを使用すると、クライアント API の接続先である特定の Process Automation Manager コントローラーで発生した変更に基づいてイベントが通知されるようにすることができます。たとえ

ば、KIE Server テンプレートもしくはインスタンスが接続する、または Process Automation Manager コントローラー内で更新されると、通知を受け取ることができます。

サーバーテンプレートを返し、イベント通知を指定する WebSocket 要求の例

```
import org.kie.server.api.marshalling.MarshallingFormat;
import org.kie.server.controller.api.model.events.*;
import org.kie.server.controller.api.model.spec.ServerTemplateList;
import org.kie.server.controller.client.KieServerControllerClient;
import org.kie.server.controller.client.KieServerControllerClientFactory;
import org.kie.server.controller.client.event.EventHandler;

public class WebSocketEventsExample {

    private static final String URL = "ws://localhost:8080/my-controller/websocket/controller";
    private static final String USER = "baAdmin";
    private static final String PASSWORD = "password@1";

    public static void main(String[] args) {
        KieServerControllerClient client =
KieServerControllerClientFactory.newWebSocketClient(URL,
                                                            USER,
                                                            PASSWORD,
                                                            MarshallingFormat.JSON,
                                                            new TestEventHandler());

        // Retrieve list of server templates
        final ServerTemplateList serverTemplateList = client.listServerTemplates();
        System.out.println(String.format("Found %s server template(s) at controller url: %s",
                                         serverTemplateList.getServerTemplates().length,
                                         URL));
        try {
            Thread.sleep(60 * 1000);
        } catch (Exception e) {
            e.printStackTrace();
        }
    }

    // Set up event notifications
    static class TestEventHandler implements EventHandler {

        @Override
        public void onServerInstanceConnected(ServerInstanceConnected
serverInstanceConnected) {
            System.out.println("serverInstanceConnected = " + serverInstanceConnected);
        }

        @Override
        public void onServerInstanceDeleted(ServerInstanceDeleted serverInstanceDeleted) {
            System.out.println("serverInstanceDeleted = " + serverInstanceDeleted);
        }

        @Override
        public void onServerInstanceDisconnected(ServerInstanceDisconnected
serverInstanceDisconnected) {
            System.out.println("serverInstanceDisconnected = " + serverInstanceDisconnected);
```

```
        }

        @Override
        public void onServerTemplateDeleted(ServerTemplateDeleted serverTemplateDeleted) {
            System.out.println("serverTemplateDeleted = " + serverTemplateDeleted);
        }

        @Override
        public void onServerTemplateUpdated(ServerTemplateUpdated serverTemplateUpdated) {
            System.out.println("serverTemplateUpdated = " + serverTemplateUpdated);
        }

        @Override
        public void onServerInstanceUpdated(ServerInstanceUpdated serverInstanceUpdated) {
            System.out.println("serverInstanceUpdated = " + serverInstanceUpdated);
        }

        @Override
        public void onContainerSpecUpdated(ContainerSpecUpdated containerSpecUpdated) {
            System.out.println("onContainerSpecUpdated = " + containerSpecUpdated);
        }
    }
}
```

第27章 BUSINESS CENTRAL PROCESS 向けの BPMN プロセス FLUENT API

Red Hat Process Automation Manager は、ファクトリーを使用してビジネスプロセスを作成できる BPMN プロセスの Fluent API を提供します。また、プロセスの Fluent API を使用して作成したビジネスプロセスを手動で検証することもできます。プロセスの Fluent API は **org.kie.api.fluent** パッケージで定義されます。

したがって、BPMN2 XML 標準を使用する代わりに、プロセスの Fluent API を使用して数行でビジネスプロセスを作成することができます。

27.1. BPMN プロセスの FLUENT API を使用した要求の例

以下の例は、ビジネスプロセスとの基本的な対話に対する BPMN プロセスの Fluent API 要求を示しています。その他の例については、Red Hat カスタマーポータル から **Red Hat Process Automation Manager 7.12.0 Source Distribution** をダウンロードし、**~/rhpam-7.12.0-sources/src/droolsjbpm-knowledge-$VERSION/kie-api/src/main/java/org/kie/api/fluent** に移動します。

Business Central のビジネスプロセスの作成および操作

以下の例は、スクリプトタスク、例外ハンドラー、および変数を使用した基本的なビジネスプロセスを示しています。

Business Central のビジネスプロセスを作成して操作する要求の例

```
Process process =
        // Create process builder
        factory.processBuilder(processId)
            // package and name
            .packageName("org.jbpm")
            .name("My process")
            // start node
            .startNode(1).name("Start").done()
            // Add variable of type string
            .variable(var("pepe", String.class))
            // Add exception handler
            .exceptionHandler(IllegalArgumentException.class, Dialect.JAVA,
"System.out.println(\"Exception\");")
            // script node in Java language that prints "action"
            .actionNode(2).name("Action")
            .action(Dialect.JAVA,
                "System.out.println(\"Action\");").done()
            // end node
            .endNode(3).name("End").done()
            // connections
            .connection(1,
                    2)
            .connection(2,
                    3)
            .build();
```

この例では、**ProcessBuilderFactory** 参照を取得し、**processBuilder(String processId)** メソッドを使用すると、指定されたプロセス ID に関連する **ProcessBuilder** インスタンスが作成されます。**ProcessBuilder** インスタンスでは、Fluent API を使用して作成されたプロセスの定義を構築できます。

ビジネスプロセスは、以下の 3 つのコンポーネントで設定されています。

- ヘッダー: プロセス、インポート、変数の名前などのグローバル要素で設定されるヘッダーセクション。
  この例では、ヘッダーにはプロセスの名前およびバージョンとパッケージ名が含まれます。

- ノード: プロセスの一部である異なるすべてのノードを含むノードセクション。
  上記の例では、**startNode()** メソッド、**actionNode()** メソッド、および **endNode()** メソッドを呼び出すことで、ノードがプロセスに追加されます。これらのメソッドは、ノードのプロパティーを設定できる特定の **NodeBuilder** を返します。コードが特定のノードの設定が完了すると、**done()** メソッドは **NodeContainerBuilder** を返し、必要に応じてノードを追加します。

- 接続: connection セクションはノードをリンクし、フローチャートを作成します。
  上記の例では、すべてのノードを追加すると、ノード間の接続を作成して接続する必要があります。ノードをリンクする **connection()** メソッドを呼び出すことができます。

最後に、**build()** メソッドを呼び出して、生成されたプロセス定義を取得できます。また、**build()** メソッドはプロセス定義を検証し、プロセス定義が有効でない場合は例外が発生します。

27.2. ビジネスプロセスを実行する要求の例

有効なプロセス定義インスタンスを作成したら、パブリック KIE API と内部 KIE API の組み合わせを使用してこれを実行できます。プロセスを実行するには、**KieBase** の作成に使用する **Resource** を作成します。**KieBase** を使用して、プロセスを実行するために **KieSession** を作成できます。

以下の例は **ProcessBuilderFactory.toBytes** プロセスを使用して **ByteArrayResource** リソースを作成します。

プロセスを実行する要求の例

```
// Build resource from Process
KieResources resources = ServiceRegistry.getInstance().get(KieResources.class);
Resource res = resources
                .newByteArrayResource(factory.toBytes(process))
                .setSourcePath("/tmp/processFactory.bpmn2"); // source path or target path must be
set to be added into kbase
// Build kie base from this resource using KIE API
KieServices ks = KieServices.Factory.get();
KieRepository kr = ks.getRepository();
KieFileSystem kfs = ks.newKieFileSystem();
kfs.write(res);
KieBuilder kb = ks.newKieBuilder(kfs);
kb.buildAll(); // kieModule is automatically deployed to KieRepository if successfully built.
KieContainer kContainer = ks.newKieContainer(kr.getDefaultReleaseId());
KieBase kbase = kContainer.getKieBase();
// Create kie session using KieBase
KieSessionConfiguration conf = ...;
Environment env = ....;
KieSession ksession = kbase.newKieSession(conf,env);
// execute process using same process Id that is used to obtain ProcessBuilder instance
ksession.startProcess(processId)
```

第28章 BUSINESS CENTRAL スペースおよびプロジェクト用のナレッジストア REST API

Red Hat Process Automation Manager はナレッジストア REST API を提供し、これを使用することで Business Central ユーザーインターフェイスを使わずに Red Hat Process Automation Manager のプロジェクトやスペースを操作することができます。ナレッジストアは、Red Hat Process Automation Manager のアセット用のアーティファクトリーポジトリーです。この API のサポートにより、Business Central プロジェクトとスペースの活用と、それらのメンテナーンスの自動化が可能になります。

ナレッジストア REST API を使用すると、以下のアクションが可能になります。

- 全プロジェクトおよびスペースに関する情報の取得
- プロジェクトおよびスペースの作成、更新、削除
- プロジェクトのビルド、デプロイ、およびテスト
- 以前のナレッジストア REST API 要求または **jobs** についての情報の取得

ナレッジストア REST API 要求には以下のコンポーネントが必要です。

認証

ナレッジストア REST API は、ユーザーロール **rest-all** に HTTP の Basic 認証またはトークンベースの認証を必要とします。お使いの Red Hat Process Automation Manager に設定されているユーザーロールを表示するには、**~/$SERVER_HOME/standalone/configuration/application-roles.properties** と **~/application-users.properties** に移動します。
ユーザーに **rest-all** ロールを追加するには、**~/$SERVER_HOME/bin** に移動して以下のコマンドを実行します。

```
$ ./bin/jboss-cli.sh --commands="embed-server --std-out=echo,/subsystem=elytron/filesystem-realm=ApplicationRealm:add-identity(identity=<USERNAME>),/subsystem=elytron/filesystem-realm=ApplicationRealm:set-password(identity=<USERNAME>, clear={password='<PASSWORD>'}),/subsystem=elytron/filesystem-realm=ApplicationRealm:add-identity-attribute(identity=<USERNAME>, name=role, value=['rest-all'])"
```

ユーザーロールと Red Hat Process Automation Manager のインストールオプションの詳細は、Red Hat Process Automation Manager インストールの計画 を参照してください。

HTTP ヘッダー

ナレッジストア REST API は、API 要求に以下の HTTP ヘッダーを必要とします。

- **Accept**: 要求元のクライアントが受け付けるデータ形式:
  - **application/json** (JSON)
- **Content-Type**: **POST** または **PUT** API 要求データ向けのデータ形式:
  - **application/json** (JSON)

HTTP メソッド

ナレッジストア REST API は、API 要求に以下の HTTP メソッドを必要とします。

- **GET**: 指定したリソースのエンドポイントから指定した情報を取得する
- **POST**: リソースを作成または更新する

- **PUT**: リソースを更新する
- **DELETE**: リソースを削除する

ベース URL

ナレッジストア REST API リクエストのベース URL は **http://SERVER:PORT/business-central/rest/** で、たとえば **http://localhost:8080/business-central/rest/** となります。

注記

ナレッジストアの REST API のベース URL と Business Central にビルトインの Process Automation Manager コントローラーのものは、両方とも Business Central REST サービスの一部とみなされるため同じになります。

エンドポイント

特定のスペースにおける **/spaces/{spaceName}** など、ナレッジストア REST API のエンドポイントは、ナレッジストア REST API ベース URL に追記する URI で、Red Hat Process Automation Manager の対応するリソースやリソースタイプにアクセスするためのものです。

/spaces/{spaceName} エンドポイントの要求 URL 例

http://localhost:8080/business-central/rest/spaces/MySpace

要求データ

ナレッジストア REST API の HTTP **POST** 要求は、データに JSON 要求ボディが必要になる場合があります。

POST 要求 URL と JSON 要求のボディデータの例

http://localhost:8080/business-central/rest/spaces/MySpace/projects

```
{
  "name": "Employee_Rostering",
  "groupId": "employeerostering",
  "version": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill."
}
```

28.1. REST クライアントまたは CURL ユーティリティーを使用した ナレッジストア REST API による要求送信

ナレッジストア REST API を使用すると、Business Central ユーザーインターフェイスを使用せずに Red Hat Process Automation Manager のプロジェクトやスペースを操作することができます。ナレッジストア REST API 要求は、REST クライアントまたは curl ユーティリティーを使用して送信できます。

前提条件

- Business Central をインストールし、実行している。
- **rest-all** ユーザーロールで Process Server にアクセスできる。

手順

1. 要求の送信先となる関連する API エンドポイント を特定します。Business Central からスペースを取得する **[GET] /spaces** などです。

2. REST クライアントまたは curl ユーティリティーで、**/spaces** への **GET** 要求に以下のコンポーネントを入力します。ご自分のユースケースに合わせて、要求詳細を調整します。
   REST クライアントの場合:

   - **Authentication**: **rest-all** ロールを持つ Business Central ユーザーのユーザー名とパスワードを入力します。
   - **HTTP Headers**: 以下のヘッダーを設定します。
     - **Accept**: **application/json**
   - **HTTP method**: **GET** に設定します。
   - **URL**: Process Server REST API ベース URL とエンドポイントを入力します。たとえば、**http://localhost:8080/business-central/rest/spaces** となります。

   curl ユーティリティーの場合:

   - **-u**: **rest-all** ロールを持つ Business Central ユーザーのユーザー名とパスワードを入力します。
   - **-H**: 以下のヘッダーを設定します。
     - **Accept**: **application/json**
   - **-X**: **GET** に設定します。
   - **URL**: Process Server REST API ベース URL とエンドポイントを入力します。たとえば、**http://localhost:8080/business-central/rest/spaces** となります。

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -X GET
"http://localhost:8080/business-central/rest/spaces"
```

3. 要求を実行し、KIE Server の応答を確認します。
   サーバー応答の例 (JSON):


```
[
  {
    "name": "MySpace",
    "description": null,
    "projects": [
      {
        "name": "Employee_Rostering",
        "spaceName": "MySpace",
        "groupId": "employeerostering",
        "version": "1.0.0-SNAPSHOT",
        "description": "Employee rostering problem optimisation using Planner. Assigns
employees to shifts based on their skill.",
        "publicURIs": [
          {
            "protocol": "git",
```

```
          "uri": "git://localhost:9418/MySpace/example-Employee_Rostering"
        },
        {
          "protocol": "ssh",
          "uri": "ssh://localhost:8001/MySpace/example-Employee_Rostering"
        }
      ]
    },
    {
      "name": "Mortgage_Process",
      "spaceName": "MySpace",
      "groupId": "mortgage-process",
      "version": "1.0.0-SNAPSHOT",
      "description": "Getting started loan approval process in BPMN2, decision table, business
rules, and forms.",
      "publicURIs": [
        {
          "protocol": "git",
          "uri": "git://localhost:9418/MySpace/example-Mortgage_Process"
        },
        {
          "protocol": "ssh",
          "uri": "ssh://localhost:8001/MySpace/example-Mortgage_Process"
        }
      ]
    }
  ],
  "owner": "admin",
  "defaultGroupId": "com.myspace"
 },
 {
  "name": "MySpace2",
  "description": null,
  "projects": [
    {
      "name": "IT_Orders",
      "spaceName": "MySpace",
      "groupId": "itorders",
      "version": "1.0.0-SNAPSHOT",
      "description": "Case Management IT Orders project",
      "publicURIs": [
        {
          "protocol": "git",
          "uri": "git://localhost:9418/MySpace/example-IT_Orders-1"
        },
        {
          "protocol": "ssh",
          "uri": "ssh://localhost:8001/MySpace/example-IT_Orders-1"
        }
      ]
    }
  ],
  "owner": "admin",
  "defaultGroupId": "com.myspace"
 }
]
```

4. REST クライアントまたは curl ユーティリティーで、**/spaces/{spaceName}/projects** への **POST** 要求を以下のコンポーネントで送信し、スペース内でプロジェクトを作成します。ご自分のユースケースに合わせて、要求詳細を調整します。
REST クライアントの場合:

- **Authentication**: **rest-all** ロールを持つ Business Central ユーザーのユーザー名とパスワードを入力します。
- **HTTP Headers**: 以下のヘッダーを設定します。
  - **Accept**: **application/json**
  - **accept-Language**: **en-US**
  - **Content-Type**: **application/json**
- **HTTP method**: **POST** に設定します。
- **URL**: Process Server REST API ベース URL とエンドポイントを入力します。たとえば、**http://localhost:8080/business-central/rest/spaces/MySpace/projects** となります。
- **要求のボディ**: 新規プロジェクト用の ID データのある JSON 要求ボディを追加します。

```
{
  "name": "Employee_Rostering",
  "groupId": "employeerostering",
  "version": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill."
}
```

curl ユーティリティーの場合:

- **-u**: **rest-all** ロールを持つ Business Central ユーザーのユーザー名とパスワードを入力します。
- **-H**: 以下のヘッダーを設定します。
  - **Accept**: **application/json**
  - **accept-Language**: **en-US** (定義されていない場合は JVM のデフォルトのロケールが反映されます)
  - **Content-Type**: **application/json**
- **-X**: **POST** に設定します。
- **URL**: Process Server REST API ベース URL とエンドポイントを入力します。たとえば、**http://localhost:8080/business-central/rest/spaces/MySpace/projects** となります。
- **-d**: 新規プロジェクト用の ID データのある JSON 要求のボディまたはファイル (**@file.json**) を追加します。

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Accept-Language: en-US" -H "Content-Type: application/json" -X POST "http://localhost:8080/business-central/rest/spaces/MySpace/projects" -d "{ \"name\": \"Employee_Rostering\", \"groupId\":
```

```
\"employeerostering\", \"version\": \"1.0.0-SNAPSHOT\", \"description\": \"Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill.\"}"
```

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Accept-Language: en-US" -H "Content-Type: application/json" -X POST "http://localhost:8080/business-central/rest/spaces/MySpace/projects" -d @my-project.json
```

5. 要求を実行し、KIE Server の応答を確認します。
サーバー応答の例 (JSON):

```
{
  "jobId": "1541017411591-6",
  "status": "APPROVED",
  "spaceName": "MySpace",
  "projectName": "Employee_Rostering",
  "projectGroupId": "employeerostering",
  "projectVersion": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill."
}
```

要求エラーが発生した場合は、返されたエラーコードメッセージを確認して、それに応じて要求を調整します。

28.2. サポートされるナレッジストア REST API エンドポイント

ナレッジストア REST API は、Red Hat Process Automation Manager 内のスペースおよびプロジェクトを管理し、以前の内 REST API リクエストまたは **jobs** に関する情報を取得するエンドポイントを提供します。

28.2.1. スペース

ナレッジストア REST API は Business Central のスペースを管理するための以下のエンドポイントをサポートします。ナレッジストア REST API のベース URL は **http://SERVER:PORT/business-central/rest/** です。ユーザーロール **rest-all** では、すべての要求で HTTP の Basic 認証またはトークンベースの認証が必要です。

[GET] /spaces

Business Central のすべてのスペースを返します。

サーバーの応答例 (JSON)

```
[
  {
    "name": "MySpace",
    "description": null,
    "projects": [
      {
        "name": "Employee_Rostering",
        "spaceName": "MySpace",
        "groupId": "employeerostering",
        "version": "1.0.0-SNAPSHOT",
        "description": "Employee rostering problem optimisation using Planner. Assigns employees to
```

```
shifts based on their skill.",
        "publicURIs": [
          {
            "protocol": "git",
            "uri": "git://localhost:9418/MySpace/example-Employee_Rostering"
          },
          {
            "protocol": "ssh",
            "uri": "ssh://localhost:8001/MySpace/example-Employee_Rostering"
          }
        ]
      },
      {
        "name": "Mortgage_Process",
        "spaceName": "MySpace",
        "groupId": "mortgage-process",
        "version": "1.0.0-SNAPSHOT",
        "description": "Getting started loan approval process in BPMN2, decision table, business
rules, and forms.",
        "publicURIs": [
          {
            "protocol": "git",
            "uri": "git://localhost:9418/MySpace/example-Mortgage_Process"
          },
          {
            "protocol": "ssh",
            "uri": "ssh://localhost:8001/MySpace/example-Mortgage_Process"
          }
        ]
      }
    ],
    "owner": "admin",
    "defaultGroupId": "com.myspace"
  },
  {
    "name": "MySpace2",
    "description": null,
    "projects": [
      {
        "name": "IT_Orders",
        "spaceName": "MySpace",
        "groupId": "itorders",
        "version": "1.0.0-SNAPSHOT",
        "description": "Case Management IT Orders project",
        "publicURIs": [
          {
            "protocol": "git",
            "uri": "git://localhost:9418/MySpace/example-IT_Orders-1"
          },
          {
            "protocol": "ssh",
            "uri": "ssh://localhost:8001/MySpace/example-IT_Orders-1"
          }
        ]
      }
    ],
```

```
    "owner": "admin",
    "defaultGroupId": "com.myspace"
  }
]
```


[GET] /spaces/{spaceName}

指定したスペースに関する情報を返します。

表28.1 要求パラメーター

Name	説明	タイプ	要件
spaceName	取得するスペースの名前	文字列	必須

サーバーの応答例 (JSON)


```
{
  "name": "MySpace",
  "description": null,
  "projects": [
    {
      "name": "Mortgage_Process",
      "spaceName": "MySpace",
      "groupId": "mortgage-process",
      "version": "1.0.0-SNAPSHOT",
      "description": "Getting started loan approval process in BPMN2, decision table, business rules, and forms.",
      "publicURIs": [
        {
          "protocol": "git",
          "uri": "git://localhost:9418/MySpace/example-Mortgage_Process"
        },
        {
          "protocol": "ssh",
          "uri": "ssh://localhost:8001/MySpace/example-Mortgage_Process"
        }
      ]
    },
    {
      "name": "Employee_Rostering",
      "spaceName": "MySpace",
      "groupId": "employeerostering",
      "version": "1.0.0-SNAPSHOT",
      "description": "Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill.",
      "publicURIs": [
        {
          "protocol": "git",
          "uri": "git://localhost:9418/MySpace/example-Employee_Rostering"
        },
        {
          "protocol": "ssh",
```

```
          "uri": "ssh://localhost:8001/MySpace/example-Employee_Rostering"
        }
      ]
    },
    {
      "name": "Evaluation_Process",
      "spaceName": "MySpace",
      "groupId": "evaluation",
      "version": "1.0.0-SNAPSHOT",
      "description": "Getting started Business Process for evaluating employees",
      "publicURIs": [
        {
          "protocol": "git",
          "uri": "git://localhost:9418/MySpace/example-Evaluation_Process"
        },
        {
          "protocol": "ssh",
          "uri": "ssh://localhost:8001/MySpace/example-Evaluation_Process"
        }
      ]
    },
    {
      "name": "IT_Orders",
      "spaceName": "MySpace",
      "groupId": "itorders",
      "version": "1.0.0-SNAPSHOT",
      "description": "Case Management IT Orders project",
      "publicURIs": [
        {
          "protocol": "git",
          "uri": "git://localhost:9418/MySpace/example-IT_Orders"
        },
        {
          "protocol": "ssh",
          "uri": "ssh://localhost:8001/MySpace/example-IT_Orders"
        }
      ]
    }
  ],
  "owner": "admin",
  "defaultGroupId": "com.myspace"
}
```


[POST] /spaces

Business Central でスペースを作成します。

表28.2 要求パラメーター

Name	説明	タイプ	要件
ボディ	新規スペースの **name**、**description**、**owner**、**defaultGroupId**、およびその他のコンポーネント	要求ボディ	必須

要求ボディ (JSON) 例

```
{
  "name": "NewSpace",
  "description": "My new space.",
  "owner": "admin",
  "defaultGroupId": "com.newspace"
}
```

サーバーの応答例 (JSON)

```
{
  "jobId": "1541016978154-3",
  "status": "APPROVED",
  "spaceName": "NewSpace",
  "owner": "admin",
  "defaultGroupId": "com.newspace",
  "description": "My new space."
}
```

[PUT] /spaces

Business Central のスペースの **description**、**owner**、および **defaultGroupId** を更新します。

要求ボディ (JSON) 例

```
{
  "name": "MySpace",
  "description": "This is updated description",
  "owner": "admin",
  "defaultGroupId": "com.updatedGroupId"
}
```

サーバーの応答例 (JSON)

```
{
  "jobId": "1592214574454-1",
  "status": "APPROVED",
  "spaceName": "MySpace",
  "owner": "admin",
  "defaultGroupId": "com.updatedGroupId",
  "description": "This is updated description"
}
```

[DELETE] /spaces/{spaceName}

Business Central から指定したスペースを削除します。

表28.3 要求パラメーター

Name	説明	タイプ	要件
spaceName	削除するスペースの名前	文字列	必須

サーバーの応答例 (JSON)

```
{
  "jobId": "1541127032997-8",
  "status": "APPROVED",
  "spaceName": "MySpace",
  "owner": "admin",
  "description": "My deleted space.",
  "repositories": null
}
```

28.2.2. プロジェクト

ナレッジストア REST API は Business Central のプロジェクトを管理、ビルド、デプロイするための以下のエンドポイントをサポートします。ナレッジストア REST API のベース URL は **http://SERVER:PORT/business-central/rest/** です。ユーザーロール **rest-all** では、すべての要求で HTTP の Basic 認証またはトークンベースの認証が必要です。

[GET] /spaces/{spaceName}/projects

指定したスペースにあるプロジェクトを返します。

表28.4 要求パラメーター

Name	説明	タイプ	要件
spaceName	取得するプロジェクトのスペース名	文字列	必須

サーバーの応答例 (JSON)

```
[
  {
    "name": "Mortgage_Process",
    "spaceName": "MySpace",
    "groupId": "mortgage-process",
    "version": "1.0.0-SNAPSHOT",
    "description": "Getting started loan approval process in BPMN2, decision table, business rules, and forms.",
    "publicURIs": [
      {
        "protocol": "git",
        "uri": "git://localhost:9418/MySpace/example-Mortgage_Process"
      },
      {
        "protocol": "ssh",
```

```
      "uri": "ssh://localhost:8001/MySpace/example-Mortgage_Process"
    }
   ]
  },
  {
   "name": "Employee_Rostering",
   "spaceName": "MySpace",
   "groupId": "employeerostering",
   "version": "1.0.0-SNAPSHOT",
   "description": "Employee rostering problem optimisation using Planner. Assigns employees to
shifts based on their skill.",
   "publicURIs": [
    {
     "protocol": "git",
     "uri": "git://localhost:9418/MySpace/example-Employee_Rostering"
    },
    {
     "protocol": "ssh",
     "uri": "ssh://localhost:8001/MySpace/example-Employee_Rostering"
    }
   ]
  },
  {
   "name": "Evaluation_Process",
   "spaceName": "MySpace",
   "groupId": "evaluation",
   "version": "1.0.0-SNAPSHOT",
   "description": "Getting started Business Process for evaluating employees",
   "publicURIs": [
    {
     "protocol": "git",
     "uri": "git://localhost:9418/MySpace/example-Evaluation_Process"
    },
    {
     "protocol": "ssh",
     "uri": "ssh://localhost:8001/MySpace/example-Evaluation_Process"
    }
   ]
  },
  {
   "name": "IT_Orders",
   "spaceName": "MySpace",
   "groupId": "itorders",
   "version": "1.0.0-SNAPSHOT",
   "description": "Case Management IT Orders project",
   "publicURIs": [
    {
     "protocol": "git",
     "uri": "git://localhost:9418/MySpace/example-IT_Orders"
    },
    {
     "protocol": "ssh",
     "uri": "ssh://localhost:8001/MySpace/example-IT_Orders"
    }
```

```
    ]
  }
]
```

[GET] /spaces/{spaceName}/projects/{projectName}

指定したスペースにある指定したプロジェクト関する情報を返します。

表28.5 要求パラメーター

Name	説明	タイプ	要件
spaceName	プロジェクトのあるスペースの名前	文字列	必須
projectName	取得するプロジェクトの名前	文字列	必須

サーバーの応答例 (JSON)

```
{
  "name": "Employee_Rostering",
  "spaceName": "MySpace",
  "groupId": "employeerostering",
  "version": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using Planner. Assigns employees to
shifts based on their skill.",
  "publicURIs": [
    {
      "protocol": "git",
      "uri": "git://localhost:9418/MySpace/example-Employee_Rostering"
    },
    {
      "protocol": "ssh",
      "uri": "ssh://localhost:8001/MySpace/example-Employee_Rostering"
    }
  ]
}
```

[POST] /spaces/{spaceName}/projects

指定したスペースにプロジェクトを作成します。

表28.6 要求パラメーター

Name	説明	タイプ	要件
spaceName	新規プロジェクトが作成されるスペースの名前	文字列	必須
ボディ	新規プロジェクトの **name**、**groupId**、**version**、**description**、およびその他のコンポーネント	要求ボディ	必須

要求ボディ (JSON) 例

```
{
  "name": "Employee_Rostering",
  "groupId": "employeerostering",
  "version": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using Planner. Assigns employees to
shifts based on their skill."
}
```

サーバーの応答例 (JSON)

```
{
  "jobId": "1541017411591-6",
  "status": "APPROVED",
  "spaceName": "MySpace",
  "projectName": "Employee_Rostering",
  "projectGroupId": "employeerostering",
  "projectVersion": "1.0.0-SNAPSHOT",
  "description": "Employee rostering problem optimisation using Planner. Assigns employees to
shifts based on their skill."
}
```

[DELETE] /spaces/{spaceName}/projects/{projectName}

指定したスペースから指定したプロジェクトを削除します。

表28.7 要求パラメーター

Name	説明	タイプ	要件
spaceName	プロジェクトのあるスペースの名前	文字列	必須
projectName	削除するプロジェクトの名前	文字列	必須

サーバーの応答例 (JSON)

```
{
  "jobId": "1541128617727-10",
  "status": "APPROVED",
  "projectName": "Employee_Rostering",
  "spaceName": "MySpace"
}
```

[POST] /spaces/{spaceName}/git/clone

指定した Git アドレスから指定したスペースにプロジェクトのクローンを作成します。

表28.8 要求パラメーター

Name	説明	タイプ	要件
spaceName	プロジェクトのクローンを作成するスペース名	文字列	必須
ボディ	クローンするプロジェクトの **name**、**description**、Git リポジトリーの **userName**、**password**、および **gitURL**	要求ボディ	必須

要求ボディ (JSON) 例

```
{
  "name": "Employee_Rostering",
  "description": "Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill.",
  "userName": "baAdmin",
  "password": "password@1",
  "gitURL": "git://localhost:9418/MySpace/example-Employee_Rostering"
}
```

サーバーの応答例 (JSON)

```
{
  "jobId": "1541129488547-13",
  "status": "APPROVED",
  "cloneProjectRequest": {
    "name": "Employee_Rostering",
    "description": "Employee rostering problem optimisation using Planner. Assigns employees to shifts based on their skill.",
    "userName": "baAdmin",
    "password": "password@1",
    "gitURL": "git://localhost:9418/MySpace/example-Employee_Rostering"
  },
  "spaceName": "MySpace2"
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/maven/compile

指定したスペースで指定したプロジェクトをコンパイルします (**mvn compile** と同等)。

表28.9 要求パラメーター

Name	説明	タイプ	要件
spaceName	プロジェクトのあるスペースの名前	文字列	必須

Name	説明	タイプ	要件
projectName	コンパイルするプロジェクトの名前	文字列	必須

サーバーの応答例 (JSON)

```
{
  "jobId": "1541128617727-10",
  "status": "APPROVED",
  "projectName": "Employee_Rostering",
  "spaceName": "MySpace"
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/maven/test

指定したスペースで指定したプロジェクトをテストします (**mvn test** と同等)。

表28.10 要求パラメーター

Name	説明	タイプ	要件
spaceName	プロジェクトのあるスペースの名前	文字列	必須
projectName	テストするプロジェクトの名前	文字列	必須

サーバーの応答例 (JSON)

```
{
  "jobId": "1541132591595-19",
  "status": "APPROVED",
  "projectName": "Employee_Rostering",
  "spaceName": "MySpace"
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/maven/install

指定したスペースで指定したプロジェクトをインストールします (**mvn install** と同等)。

表28.11 要求パラメーター

Name	説明	タイプ	要件
spaceName	プロジェクトのあるスペースの名前	文字列	必須
projectName	インストールするプロジェクトの名前	文字列	必須

サーバーの応答例 (JSON)

```
{
  "jobId": "1541132668987-20",
  "status": "APPROVED",
  "projectName": "Employee_Rostering",
  "spaceName": "MySpace"
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/maven/deploy

指定したスペースで指定したプロジェクトをデプロイします (**mvn deploy** と同等)。

表28.12 要求パラメーター

Name	説明	タイプ	要件
spaceName	プロジェクトのあるスペースの名前	文字列	必須
projectName	デプロイするプロジェクトの名前	文字列	必須

サーバーの応答例 (JSON)

```
{
  "jobId": "1541132816435-21",
  "status": "APPROVED",
  "projectName": "Employee_Rostering",
  "spaceName": "MySpace"
}
```

28.2.3. ジョブ (API 要求)

ナレッジストア REST API の **POST** と **DELETE** 要求はすべて、返される要求詳細のほかに、各要求に関連付けられたジョブ ID を返します。ジョブ ID を使用すると、要求ステータスを確認したり、送信された要求を削除することができます。

ナレッジストア REST API 要求もしくは **ジョブ** には、以下のステータスがあります。

表28.13 ジョブステータス (API 要求ステータス)

ステータス	説明
ACCEPTED	ジョブが受け入れられ、処理中である。
BAD_REQUEST	要求に無効なコンテンツが含まれ、受け入れられなかった。
RESOURCE_NOT_EXIST	要求されたリソース (パス) が存在しない。

ステータス	説明
DUPLICATE_RESOURCE	リソースがすでに存在する。
SERVER_ERROR	KIE Server でエラーが発生した。
SUCCESS	要求が正常に完了した。
FAIL	要求が失敗した。
APPROVED	要求が承認された。
DENIED	要求が拒否された。
GONE	以下のいずれかの理由で要求のジョブ ID が見つからなかった。 ● 要求が明示的に削除された。 ● 要求が完了してステータスキャッシュから削除されている。キャッシュが最大容量に達すると、要求はステータスキャッシュから削除されます。 ● 要求が元々存在しなかった。

ナレッジストア REST API は、送信済み API 要求の取得または削除用の以下のエンドポイントをサポートします。ナレッジストア REST API のベース URL は **http://SERVER:PORT/business-central/rest/** です。ユーザーロール **rest-all** では、すべての要求で HTTP の Basic 認証またはトークンベースの認証が必要です。

[GET] /jobs/{jobId}

指定されたジョブのステータスを返します (以前に送信された API 要求)。

表28.14 要求パラメーター

Name	説明	タイプ	要件
jobId	取得するジョブの ID (例: **1541010216919-1**)	文字列	必須

サーバーの応答例 (JSON)

```
{
  "status": "SUCCESS",
  "jobId": "1541010216919-1",
  "result": null,
  "lastModified": 1541010218352,
  "detailedResult": [
    "level:INFO, path:null, text:Build of module 'Mortgage_Process' (requested by system)
```

```
completed.\n Build: SUCCESSFUL"
  ]
}
```

[DELETE] /jobs/{jobId}

指定したジョブ (以前に送信された API 要求) を削除します。ジョブがまだ処理されていない場合、この要求はジョブをジョブキューから削除します。実行中のジョブがキャンセルされたり停止されたりすることはありません。

表28.15 要求パラメーター

Name	説明	タイプ	要件
jobId	削除するジョブの ID (例: **1541010216919-1**)	文字列	必須

サーバーの応答例 (JSON)

```
{
  "status": "GONE",
  "jobId": "1541010216919-1",
  "result": null,
  "lastModified": 1541132054916,
  "detailedResult": [
    "level:INFO, path:null, text:Build of module 'Mortgage_Process' (requested by system)
completed.\n Build: SUCCESSFUL"
  ]
}
```

28.2.4. ブランチ

ナレッジストア REST API は Business Central のブランチを管理するための以下のエンドポイントをサポートします。ナレッジストア REST API のベース URL は **http://SERVER:PORT/business-central/rest/** です。ユーザーロール **rest-all** では、すべての要求で HTTP の Basic 認証またはトークンベースの認証が必要です。

[GET] /spaces/{spaceName}/projects/{projectName}/branches

指定のプロジェクトおよびスペースにあるブランチをすべて返します。

表28.16 要求パラメーター

Name	説明	タイプ	要件
spaceName	取得するプロジェクトのスペース名	文字列	必須
projectName	ブランチを取得するためのプロジェクト名	文字列	必須

サーバーの応答例 (JSON)

```
[
  {
    "name":"master"
  }
]
```

[POST] /spaces/{spaceName}/projects/{projectName}/branches

指定のプロジェクトおよびスペースに、指定したブランチを追加します。

表28.17 要求パラメーター

Name	説明	タイプ	要件
spaceName	プロジェクトのあるスペースの名前	文字列	必須
projectName	新しくブランチを作成する必要があるプロジェクトの名前	文字列	必須
ボディ	プロジェクトの **newBranchName** と **baseBranchName**	要求ボディ	必須

要求ボディ (JSON) 例

```
{
  "newBranchName":  "branch01",
  "baseBranchName": "master"
}
```

サーバーの応答例 (JSON)

```
{
   "jobId":          "1576175811141-3",
   "status":         "APPROVED",
   "spaceName":      "Space123",
   "projectName":    "ProjABC",
   "newBranchName":  "b1",
   "baseBranchName": "master",
   "userIdentifier": "bc"
}
```

[DELETE] /spaces/{spaceName}/projects/{projectName}/branches/{branchName}

指定のプロジェクトおよびスペースから、指定したブランチを削除します。

表28.18 要求パラメーター

Name	説明	タイプ	要件
spaceName	プロジェクトのあるスペースの名前	文字列	必須

Name	説明	タイプ	要件
projectName	ブランチが配置されているプロジェクトの名前	文字列	必須
branchName	削除するブランチの名前	文字列	必須

サーバーの応答例 (JSON)

```
{
   "jobId":         "1576175811421-5",
   "status":        "APPROVED",
   "spaceName":     "Space123",
   "projectName":   "ProjABC",
   "branchName":    "b1",
   "userIdentifier": "bc"
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/branches/{branchName}/maven/compile

指定のプロジェクトおよびスペースで、指定したブランチをコンパイルします。**branchName** が指定されていない場合、要求は Master ブランチに適用されます。

表28.19 要求パラメーター

Name	説明	タイプ	要件
spaceName	プロジェクトのあるスペースの名前	文字列	必須
projectName	ブランチが配置されているプロジェクトの名前	文字列	必須
branchName	コンパイルするブランチの名前	文字列	必須

サーバーの応答例 (JSON)

```
{
   "jobId":      "1576175811233-4",
   "status":     "APPROVED",
   "spaceName":  "Space123",
   "projectName": "ProjABC",
   "branchName":  "b1",
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/branches/{branchName}/maven/install

指定のプロジェクトおよびスペースに、指定したブランチをインストールします。**branchName** が指定されていない場合、要求は Master ブランチに適用されます。

表28.20 要求パラメーター

Name	説明	タイプ	要件
spaceName	プロジェクトのあるスペースの名前	文字列	必須
projectName	ブランチが配置されているプロジェクトの名前	文字列	必須
branchName	インストールするブランチの名前	文字列	必須

サーバーの応答例 (JSON)

```
{
   "jobId":       "1576175811233-4",
   "status":      "APPROVED",
   "spaceName":   "Space123",
   "projectName": "ProjABC",
   "branchName":  "b1",
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/branches/{branchName}/maven/test

指定のプロジェクトおよびスペースで、指定したブランチをテストします。**branchName** が指定されていない場合、要求は Master ブランチに適用されます。

表28.21 要求パラメーター

Name	説明	タイプ	要件
spaceName	プロジェクトが配置されているスペースの名前	文字列	必須
projectName	ブランチが配置されているプロジェクトの名前	文字列	必須
branchName	テストするブランチの名前	文字列	必須

サーバーの応答例 (JSON)

```
{
   "jobId":       "1576175811233-4",
   "status":      "APPROVED",
   "spaceName":   "Space123",
```

```
    "projectName": "ProjABC",
    "branchName":  "b1",
}
```

[POST] /spaces/{spaceName}/projects/{projectName}/branches/{branchName}/maven/deploy

指定のプロジェクトおよびスペースで、指定したブランチをデプロイします。**branchName** が指定されていない場合、要求は Master ブランチに適用されます。

表28.22 要求パラメーター

Name	説明	タイプ	要件
spaceName	プロジェクトのあるスペースの名前	文字列	必須
projectName	ブランチが配置されているプロジェクトの名前	文字列	必須
branchName	デプロイするブランチの名前	文字列	必須

サーバーの応答例 (JSON)

```
{
    "jobId":       "1576175811233-4",
    "status":      "APPROVED",
    "spaceName":   "Space123",
    "projectName": "ProjABC",
    "branchName":  "b1",
}
```

第29章 BUSINESS CENTRAL のグループ、ロール、およびユーザーの SECURITY MANAGEMENT REST API

Red Hat Process Automation Manager は、Business Central ユーザーインターフェイスを使用せずに Red Hat Process Automation Manager でグループ、ロール、およびユーザーを管理する時に使用可能な Security Management REST API を提供します。この API のサポートにより、Business Central グループ、ロール、ユーザー、および付与したパーミッションの管理の容易化、自動化が可能です。

Security Management REST API を使用すると、以下のアクションが可能になります。

- すべてのグループ、ロール、ユーザー、およびそれぞれに付与された権限に関する情報を取得する
- グループとユーザーを作成、更新、または削除する
- グループ、ロール、およびユーザーに付与された権限を更新する
- ユーザーに割り当てられたグループとロールに関する情報を取得する

Security Management REST API 要求には以下のコンポーネントが必要です。

認証

Security Management REST API は、ユーザーロール **admin** に HTTP の Basic 認証またはトークンベースの認証を必要とします。お使いの Red Hat Process Automation Manager に設定されているユーザーロールを表示するには、~/**$SERVER_HOME/standalone/configuration/application-roles.properties** と ~/**application-users.properties** に移動します。
ユーザーに **admin** ロールを追加するには、~/**$SERVER_HOME/bin** に移動して以下のコマンドを実行します。

```
$ ./bin/jboss-cli.sh --commands="embed-server --std-out=echo,/subsystem=elytron/filesystem-realm=ApplicationRealm:add-identity(identity=<USERNAME>),/subsystem=elytron/filesystem-realm=ApplicationRealm:set-password(identity=<USERNAME>, clear={password='<PASSWORD>'}),/subsystem=elytron/filesystem-realm=ApplicationRealm:add-identity-attribute(identity=<USERNAME>, name=role, value=['admin'])"
```

ユーザーロールと Red Hat Process Automation Manager のインストールオプションの詳細は、Red Hat Process Automation Manager インストールの計画 を参照してください。

HTTP ヘッダー

Security Management REST API は、API 要求に以下の HTTP ヘッダーを必要とします。

- **Accept**: 要求元のクライアントが受け付けるデータ形式:
  - **application/json** (JSON)
- **Content-Type**: **POST** または **PUT** API 要求データ向けのデータ形式:
  - **application/json** (JSON)

HTTP メソッド

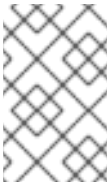
Security Management REST API は、API 要求に以下の HTTP メソッドを必要とします。

- **GET**: 指定したリソースのエンドポイントから指定した情報を取得する
- **POST**: リソースを作成または更新する

- **PUT**: リソースを更新する
- **DELETE**: リソースを削除する

ベース URL

Security Management REST API 要求のベース URL は **http://SERVER:PORT/business-central/rest/** で、たとえば **http://localhost:8080/business-central/rest/** となります。



注記

Business Central に組み込まれた Security Management、ナレッジストア、Process Automation Manager コントローラーの REST API ベース URL は、Business Central REST サービスとみなされるため、すべて同じです。

エンドポイント

特定のユーザーの **/users/{userName}** など、Security Management REST API のエンドポイントは、Security Management REST API ベース URL に追記する URI で、Red Hat Process Automation Manager 内で、対応するリソースやリソースタイプにアクセスする際に使用します。

/users/{userName} エンドポイントの要求 URL 例

http://localhost:8080/business-central/rest/users/newUser

要求データ

Security Management REST API の HTTP **POST** 要求は、データに JSON 要求のボディが必要になる場合があります。

POST 要求 URL と JSON 要求のボディデータの例

http://localhost:8080/business-central/rest/users/newUser/groups

```
[
  "newGroup"
]
```

29.1. REST クライアントまたは CURL ユーティリティーを使用した SECURITY MANAGEMENT REST API による要求送信

Security Management REST API を使用すると、Business Central ユーザーインターフェイスを使わずに Red Hat Process Automation Manager のグループ、ロール、およびユーザーを操作することができます。Security Management REST API 要求は、REST クライアントまたは curl ユーティリティーを使用して送信できます。

前提条件

- Business Central をインストールし、実行している。
- **admin** ユーザーロールで Business Central にアクセスできる。

手順

1. **[GET] /groups** など、要求の送信先に適した API エンドポイント を特定し、Business Central からグループを取得します。

2. REST クライアントまたは curl ユーティリティーで、**/groups** への **GET** 要求に以下のコンポーネントを記入します。ご自分のユースケースに合わせて、要求詳細を調整します。
REST クライアントの場合:

   - **Authentication**: **admin** ロールを持つ Business Central ユーザーのユーザー名とパスワードを入力します。
   - **HTTP Headers**: 以下のヘッダーを設定します。
     - **Accept**: **application/json**
   - **HTTP method**: **GET** に設定します。
   - **URL**: Security Management REST API ベース URL とエンドポイントを入力します。たとえば、**http://localhost:8080/business-central/rest/groups** となります。

   curl ユーティリティーの場合:

   - **-u**: **admin** ロールを持つ Business Central ユーザーのユーザー名とパスワードを入力します。
   - **-H**: 以下のヘッダーを設定します。
     - **Accept**: **application/json**
   - **-X**: **GET** に設定します。
   - **URL**: Security Management REST API ベース URL とエンドポイントを入力します。たとえば、**http://localhost:8080/business-central/rest/groups** となります。

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -X GET
"http://localhost:8080/business-central/rest/groups"
```

3. 要求を実行し、KIE Server の応答を確認します。
サーバー応答の例 (JSON):

```
[
  {
 "group1"
 },
 {
  "group2"
 }
]
```

4. REST クライアントまたは curl ユーティリティーで、**/users/{userName}/groups** への **POST** 要求を以下のコンポーネントで送信し、ユーザーに割り当てられたグループを更新します。ご自分のユースケースに合わせて、要求詳細を調整します。
REST クライアントの場合:

   - **Authentication**: **admin** ロールを持つ Business Central ユーザーのユーザー名とパスワードを入力します。

- **HTTP Headers**: 以下のヘッダーを設定します。
  - **Accept**: **application/json**
  - **Content-Type**: **application/json**
- **HTTP method**: **POST** に設定します。
- **URL**: Security Management REST API ベース URL とエンドポイントを入力します。たとえば、**http://localhost:8080/business-central/rest/users/newUser/groups** となります。
- **要求のボディ**: 新規グループ用の ID データを含む JSON 要求のボディを追加します。

```
[
  "newGroup"
]
```

curl ユーティリティーの場合:

- **-u**: **admin** ロールを持つ Business Central ユーザーのユーザー名とパスワードを入力します。
- **-H**: 以下のヘッダーを設定します。
  - **Accept**: **application/json**
  - **Content-Type**: **application/json**
- **-X**: **POST** に設定します。
- **URL**: Security Management REST API ベース URL とエンドポイントを入力します。たとえば、**http://localhost:8080/business-central/rest/users/newUser/groups** となります。
- **-d**: 新規グループ用の ID データを含む JSON 要求のボディまたはファイル (**@file.json**) を追加します。

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Content-Type:
application/json" -X POST "http://localhost:8080/business-central/rest/users/newUser/groups"
-d "["newGroup"]"
```

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Content-Type:
application/json" -X POST "http://localhost:8080/business-central/rest/users/newUser/groups"
-d @user-groups.json
```

5. 要求を実行し、KIE Server の応答を確認します。
サーバー応答の例 (JSON):

```
{
  "status": "OK",
  "message": "Groups [newGroup] are assigned successfully to user wbadmin"
}
```

要求エラーが発生した場合は、返されたエラーコードメッセージを確認して、それに応じて要求を調整します。

29.2. サポート対象の SECURITY MANAGEMENT REST API エンドポイント

Security Management REST API は、Business Central でグループ、ロール、ユーザー、およびパーミッションを管理するためのエンドポイントを提供します。これには、管理者が Business Central の **Security Management** ページでも実行できるセキュリティーおよびパーミッション管理タスクが含まれます。

29.2.1. Groups

Security Management REST API は Business Central のグループを管理するための以下のエンドポイントをサポートします。Security Management REST API のベース URL は **http://SERVER:PORT/business-central/rest/** です。ユーザーロール **admin** では、すべての要求で HTTP Basic 認証またはトークンベースの認証が必要です。

[GET] /groups

Business Central のすべてのグループを返します。

サーバーの応答例 (JSON)

```
[
  {
 "group1"
 },
 {
  "group2"
 }
]
```

[POST] /groups

Business Central でグループを作成します。グループには、少なくともユーザーを 1 つ割り当てる必要があります。

表29.1 要求パラメーター

Name	説明	タイプ	要件
ボディ	新しいグループに割り当てられたグループとユーザーの名前	要求ボディ	必須

要求ボディ (JSON) 例

```
{
  "name": "groupName",
  "users": [
    "userNames"
  ]
}
```

サーバーの応答例 (JSON)

```
{
  "status": "OK",
  "message": "Group newGroup is created successfully."
}
```

[DELETE] /groups/{groupName}

Business Central から指定のグループを削除します。

表29.2 要求パラメーター

Name	説明	タイプ	要件
groupName	削除するグループの名前	文字列	必須

サーバーの応答例 (JSON)

```
{
  "status": "OK",
  "message": "Group newGroup is deleted successfully."
}
```

29.2.2. ロール

Security Management REST API は Business Central のロールを管理するための以下のエンドポイントをサポートします。Security Management REST API のベース URL は **http://SERVER:PORT/business-central/rest/** です。ユーザーロール **admin** では、すべての要求で HTTP Basic 認証またはトークンベースの認証が必要です。

[GET] /roles

Business Central のすべてのロールを返します。

サーバーの応答例 (JSON)

```
[
  {
    "name": "process-admin"
  },
  {
    "name": "manager"
  },
  {
    "name": "admin"
  }
]
```

29.2.3. ユーザー

Security Management REST API は Business Central のユーザーを管理するための以下のエンドポイン

トをサポートします。Security Management REST API のベース URL は **http://SERVER:PORT/business-central/rest/** です。ユーザーロール **admin** では、すべての要求で HTTP Basic 認証またはトークンベースの認証が必要です。

[GET] /users

Business Central のすべてのユーザーを返します。

サーバーの応答例 (JSON)

```
[
  "newUser",
  "user1",
  "user2",
]
```

[GET] /users/{userName}/groups

指定のユーザーに割り当てられている全グループを返します。

表29.3 要求パラメーター

Name	説明	タイプ	要件
userName	割り当てられたグループを取得するユーザーの名前	文字列	必須

サーバーの応答例 (JSON)

```
[
  {
  "group1"
 },
 {
  "group2"
 }
]
```

[GET] /users/{userName}/roles

指定のユーザーに割り当てられている全ロールを返します。

表29.4 要求パラメーター

Name	説明	タイプ	要件
userName	割り当てられたロールを取得するユーザーの名前	文字列	必須

サーバーの応答例 (JSON)

```
[
  {
```

```
    "name": "process-admin"
  },
  {
    "name": "manager"
  },
  {
    "name": "admin"
  }
]
```

[POST] /users

ロールとグループを指定して、指定のユーザーを作成します。

要求ボディ (JSON) 例

```
{
  "name": "newUser",
  "roles": [
    "admin",
    "developer"
  ],
  "groups": [
    "group1",
    "group2"
  ]
}
```

サーバーの応答例 (JSON)

```
{
  "status": "OK",
  "message": "User newUser is created successfully."
}
```

[Post] /users/{userName}/changePassword

指定のユーザーのパスワードを変更します。

表29.5 要求パラメーター

Name	説明	タイプ	要件
userName	パスワードを変更するユーザーの名前	文字列	必須

要求のコマンド例

```
curl -u 'baAdmin:password@1' -H "Accept: application/json" -H "Content-Type: application/json" -X POST "http://localhost:8080/business-central/rest/users/newUser/changePassword" -d newpassword
```

サーバーの応答例 (JSON)

```
{
  "status": "OK",
  "message": "Password for newUser has been updated successfully."
}
```

[DELETE] /users/{userName}

Business Central から指定のユーザーを削除します。

表29.6 要求パラメーター

Name	説明	タイプ	要件
userName	削除するユーザーの名前	文字列	必須

サーバーの応答例 (JSON)

```
{
  "status": "OK",
  "message": "User newUser is deleted successfully."
}
```

[POST] /users/{userName}/groups

指定のユーザーに割り当てられている既存のグループを、新規グループで上書きします。

表29.7 要求パラメーター

Name	説明	タイプ	要件
userName	グループを更新するユーザーの名前	文字列	必須

要求ボディ (JSON) 例

```
[
  "newGroup"
]
```

サーバーの応答例 (JSON)

```
{
  "status": "OK",
  "message": "Groups [newGroup] are assigned successfully to user wbadmin"
}
```

[POST] /users/{userName}/roles

指定のユーザーに割り当てられている既存のロールを、新規ロールで上書きします。

表29.8 要求パラメーター

Name	説明	タイプ	要件
userName	ロールを更新するユーザーの名前	文字列	必須

要求ボディ (JSON) 例

```
[
  "admin"
]
```

サーバーの応答例 (JSON)

```
{
  "status": "OK",
  "message": "Roles [admin] are assigned successfully to user wbadmin"
}
```

29.2.4. パーミッション

Security Management REST API は Business Central のグループ、ロール、およびユーザーに割り当てられたパーミッションを管理するための以下のエンドポイントをサポートします。Security Management REST API のベース URL は **http://SERVER:PORT/business-central/rest/** です。ユーザーロール **admin** では、すべての要求で HTTP Basic 認証またはトークンベースの認証が必要です。

[GET] /groups/{groupName}/permissions

指定のグループに付与されている全パーミッションを返します。

表29.9 要求パラメーター

Name	説明	タイプ	要件
groupName	パーミッションを取得するグループの名前	文字列	必須

サーバーの応答例 (JSON)

```
{
 "homePage": "HomePerspective",
 "priority": -10,
 "project": {
  "read": {
   "access": false,
   "exceptions": []
  },

 },
 "spaces": {
  "read": {
   "access": true,
   "exceptions": [
```

```
    "MySpace"
   ]
  },
 },
 "editor": {
  "read": {
   "access": false,
   "exceptions": [
    "GuidedDecisionTreeEditorPresenter"
   ]
  },
  "create": null,
  "update": null,
  "delete": null,
  "build": null
 },
 "pages": {
  "read": {
   "access": true,
   "exceptions": []
  },
  "build": null
 },
 "workbench": {
  "editDataObject": false,
  "plannerAvailable": false,
  "editGlobalPreferences": false,
  "editProfilePreferences": false,
  "accessDataTransfer": false,
  "jarDownload": true,
  "editGuidedDecisionTableColumns": true
 }
}
```


[GET] /roles/{roleName}/permissions

指定のロールに付与されている全パーミッションを返します。

表29.10 要求パラメーター

Name	説明	タイプ	要件
roleName	パーミッションを取得するロールの名前	文字列	必須

サーバーの応答例 (JSON)


```
{
 "homePage": "HomePerspective",
 "priority": -10,
 "project": {
  "read": {
   "access": false,
   "exceptions": []
  },
```

```
 },
 "spaces": {
  "read": {
   "access": true,
   "exceptions": [
    "MySpace"
   ]
  },
 },
 "editor": {
  "read": {
   "access": false,
   "exceptions": [
    "GuidedDecisionTreeEditorPresenter"
   ]
  },
  "create": null,
  "update": null,
  "delete": null,
  "build": null
 },
 "pages": {
  "read": {
   "access": true,
   "exceptions": []
  },
  "build": null
 },
 "workbench": {
  "editDataObject": false,
  "plannerAvailable": false,
  "editGlobalPreferences": false,
  "editProfilePreferences": false,
  "accessDataTransfer": false,
  "jarDownload": true,
  "editGuidedDecisionTableColumns": true
 }
}
```


[GET] /users/{userName}/permissions

指定のユーザーに付与されている全パーミッションを返します。

表29.11 要求パラメーター

Name	説明	タイプ	要件
userName	パーミッションを取得するユーザーの名前	文字列	必須

サーバーの応答例 (JSON)


```
{
 "homePage": null,
```

```
 "priority": null,
 "project": {
  "read": {
   "access": false,
   "exceptions": []
  },

 },
 "spaces": {
  "read": {
   "access": true,
   "exceptions": [
    "MySpace"
   ]
  },
 },
 "editor": {
  "read": {
   "access": false,
   "exceptions": [
    "GuidedDecisionTreeEditorPresenter"
   ]
  },
  "create": null,
  "update": null,
  "delete": null,
  "build": null
 },
 "pages": {
  "read": {
   "access": true,
   "exceptions": []
  },
  "build": null
 },
 "workbench": {
  "editDataObject": false,
  "plannerAvailable": false,
  "editGlobalPreferences": false,
  "editProfilePreferences": false,
  "accessDataTransfer": false,
  "jarDownload": true,
  "editGuidedDecisionTableColumns": true
 }
}
```


[Post] /groups/{groupName}/permissions

指定したグループのパーミッションを更新します。

表29.12 要求パラメーター

Name	説明	タイプ	要件
groupName	パーミッションを更新するグループの名前	文字列	必須

要求ボディ (JSON) 例

```
{
  "homepage": "HomePerspective",
  "priority": 10,
  "pages": {
    "create": true,
    "read": false,
    "delete": false,
    "update": false,
    "exceptions": [
      {
        "name": "HomePerspective",
        "permissions": {
          "read": true
        }
      }
    ]
  },
  "project": {
    "create": true,
    "read": true,
    "delete": false,
    "update": false,
    "Build": false
  },
  "spaces": {
    "create": true,
    "read": true,
    "delete": false,
    "update": false
  },
  "editor": {
    "read": true
  },
  "workbench": {
    "editDataObject": true,
    "plannerAvailable": true,
    "editGlobalPreferences": true,
    "editProfilePreferences": true,
    "accessDataTransfer": true,
    "jarDownload": true,
    "editGuidedDecisionTableColumns": true
  }
}
```

サーバーの応答例 (JSON)

```
{
  "status": "OK",
  "message": "Group newGroup permissions are updated successfully."
}
```

[Post] /roles/{roleName}/permissions

指定のロールのパーミッションを更新します。

表29.13 要求パラメーター

Name	説明	タイプ	要件
roleName	パーミッションを更新するロールの名前	文字列	必須

要求ボディ (JSON) 例

```
{
 "homepage": "HomePerspective",
 "priority": 10,
 "pages": {
  "create": true,
  "read": false,
  "delete": false,
  "update": false,
  "exceptions": [{
   "name": "HomePerspective",
   "permissions": {
    "read": true
   }
  }]
 },
 "project": {
  "create": true,
  "read": true,
  "delete": false,
  "update": false,
  "Build": false
 },
 "spaces": {
  "create": true,
  "read": true,
  "delete": false,
  "update": false
 },
 "editor": {
  "read": true
 },
 "workbench": {
  "editDataObject": true,
  "plannerAvailable": true,
  "editGlobalPreferences": true,
  "editProfilePreferences": true,
  "accessDataTransfer": true,
```

```
  "jarDownload": true,
  "editGuidedDecisionTableColumns": true
 }
}
```

サーバーの応答例 (JSON)

```
{
  "status": "OK",
  "message": "Role newRole permissions are updated successfully."
}
```

29.2.4.1. Business Central でサポートされているパーミッション

Red Hat Process Automation Manager で使用できるパーミッションは次のとおりです。管理者はこれらのパーミッションを使用して、Business Central のグループ、ロール、またはユーザーに特定のアクションを許可します。

優先順位

優先順位は、複数のロールまたはグループを割り当てられたユーザーの優先度を定義する整数です。新規グループの優先順位のデフォルト値は **-100** です。Business Central で優先順位として整数値を設定できます。優先順位は、以下のルールを使用して解決されます。

表29.14 優先順位値のテーブル

整数値	優先順位
-5 未満	非常に低い
-5 から 0	低
0 相当	普通
0 から 5	高
5 以上	非常に高い

ホームページ

ホームページは、ユーザーのデフォルトのランディングページを示します。

ワークベンチ

ワークベンチは、以下の定義済みのパーミッションで設定されます。

```
{
  "editDataObject": true,
  "plannerAvailable": true,
  "editGlobalPreferences": true,
  "editProfilePreferences": true,
  "accessDataTransfer": true,
```

```
  "jarDownload": true,
  "editGuidedDecisionTableColumns": true
}
```

ページ、エディター、スペース、およびプロジェクト

以下は、リソースタイプに基にしたパーミッションで使用可能な値です。

- **PAGES: read,create,update,delete**
- **EDITOR: read**
- **SPACES: read,create,update,delete**
- **PROJECT: read,create,update,delete,build**

以下のコードを使用して、パーミッション **Pages**、**Editor**、**Spaces**、および **Projects** に例外を追加できます。

```
{
  "pages": {
    "read": false,
    "exceptions": [
      {
        "resourceName": "ProcessInstances",
        "permissions": {
          "read": false
        }
      },
      {
        "resourceName": "ProcessDefinitions",
        "permissions": {
          "read": false
        }
      }
    ]
  }
}
```

name 属性は、例外として追加するリソースの ID です。以下の REST API エンドポイントを使用して、使用可能な ID 一覧を取得します。REST API ベースの URL は **http://SERVER:PORT/business-central/rest/** です。

- **[GET] /perspectives**: Business Central での全ページのパースペクティブ名を返します。
- **[GET] /editors**: Business Central の全エディターを返します。
- **[GET] /spaces**: Business Central の全スペースを返します。
- **[GET] /spaces/{spaceName}/projects**: 指定のスペースのプロジェクトを返します。

ページのサーバー応答の例 (JSON)

```
"pages": {
 "create": true,
 "read": false,
```

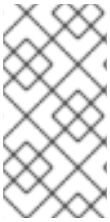
```
 "exceptions": [
    {
  "name": "HomePerspective",
  "permissions": {
   "read": true
  }
 }
]
}
```

第30章 KIE セッションやタスクサービス向けの EJB API

Red Hat Process Automation Manager には、組み込みのユースケースで使用してリモートで、アプリケーションから **KieSession** オブジェクトおよび **TaskService** オブジェクトにアクセスできるように Enterprise JavaBeans (EJB) API が含まれています。EJB API を使用すると、Red Hat Process Automation Manager のプロセスエンジンと、リモートのカスタマーアプリケーションの間で、密接にトランザクションを統合できます。

KIE Server には EJB のサポートはありませんが、KIE Server をリモートの REST または JMS で操作するように、プロセスエンジンのリモートプロトコルとして EJB を使用できます。

EJB インターフェイスの実装は、フレームワークや、コンテナーに依存しない単一の API で、フレームワーク固有のコードと併用できます。EJB サービスは、Red Hat Process Automation Manager の **org.jbpm.services.api** パッケージおよび **org.jbpm.services.ejb** パッケージで公開されます。この実装は、**RuleService** クラスをサポートしませんが、**ProcessService** クラスは **InsertCommand** や **FireAllRulesCommand** など、さまざまなルール関連のコマンドを使用できる **execute** メソッドを公開します。



注記

コンテキストと依存関係の挿入 (CDI: Contexts and Dependency Injection) も、Red Hat Process Automation Manager の **org.jbpm.services.cdi** パッケージでサポートされます。ただし、EJB の統合で競合を回避するために、EJB と CDI は併用しないでください。

30.1. サポート対象の EJB サービス

Red Hat Process Automation Manager で利用可能な Enterprise JavaBeans (EJB) サービスの完全一覧については、Red Hat カスタマーポータル から **Red Hat Process Automation Manager 7.12.0 Maven Repository** をダウンロードして、**~/jboss-rhba-7.12.0.GA-maven-repository/maven-repository/org/jbpm/jbpm-services-ejb-*** に移動してください。

jBPM サービスに対して EJB インターフェイスを提供するアーティファクトは、次のパッケージに含まれています。

- **org.jbpm.services.ejb.api**: EJB インターフェイスの jBPM サービス API の拡張を含みます。
- **org.jbpm.services.ejb.impl**: コアサービス実装の上の階層にある EJB ラッパーを含みます。
- **org.jbpm.services.ejb.client**: EJB リモートクライアント実装を含みます。Red Hat JBoss EAP でのみサポートされます。

org.jbpm.services.ejb.api パッケージには、リモート EJB クライアントで使用可能な、以下のサービスインターフェイスが含まれます。

- **DefinitionServiceEJBRemote**: このインターフェイスを使用して、プロセス (ID、名前、バージョン)、プロセス変数 (変数と型)、定義済みの再利用可能なサブプロセス、ドメイン固有のサービス、ユーザータスク、およびユーザータスクの入出力に関する情報を収集します。
- **DeploymentServiceEJBRemote**: このインターフェイスを使用してデプロイメントとデプロイメント解除を開始します。このインターフェイスには、**deploy** メソッド、**undeploy** メソッド、**getRuntimeManager** メソッド、**getDeployedUnits** メソッド、**isDeployed** メソッド、**activate** メソッド、**deactivate** メソッド、および **getDeployedUnit** メソッドが含まれます。**DeploymentUnit** のインスタンスで **deploy** メソッドを呼び出すと、**RuntimeManager** インスタンスをビルドして、ユニットをランタイムエンジンにデプロイします。デプロイメント

に成功すると、**DeployedUnit** のインスタンスが作成され、将来の使用のためにキャッシュされます。(これらの方法を使用するには、Maven リポジトリーでプロジェクトのアーティファクトをインストールする必要があります。)

- **ProcessServiceEJBRemote**: このインターフェイスを使用して、1 つまたは複数のプロセスおよびワークアイテムのライフサイクルを制御します。
- **RuntimeDataServiceEJBRemote**: このインターフェイスを使用して、プロセスインスタンス、プロセス定義、ノードインスタンスの情報、変数情報など、ランタイム関連のデータを取得します。このインターフェイスには、所有者、ステータス、および時間をもとにタスク情報を収集する便利な方法が複数含まれています。
- **UserTaskServiceEJBRemote**: このインターフェイスを使用して、ユーザータスクのライフサイクルを制御します。このインターフェイスには、**activate**、**start**、**stop**、**execute** など、ユーザータスクを操作する便利なメソッドが複数含まれます。
- **QueryServiceEJBRemote**: このインターフェイスを使用して、詳細なクエリーに対応します。
- **ProcessInstanceMigrationServiceEJBRemote**: このインターフェイスを使用して、プロセス定義の新規バージョンがデプロイされると、プロセスインスタンスを移行します。

同じ KIE Server で EJB アプリケーションおよび Business Central を実行する場合は、**org.jbpm.deploy.sync.int** システムプロパティーを設定して、指定の間隔で EJB と Business Central 間の情報を同期できます。サービスが同期を完了したら、REST 操作を使用して更新された情報にアクセスできます。

注記

Red Hat Process Automation Manager の EJB サービスは、組み込みユースケース向けに設計されています。EJB アプリケーションと Business Central を同じ KIE Server で実行する場合は、EJB アプリケーションのクラスパスに **kie-services** パッケージも追加する必要があります。

30.2. EJB サービスの WAR ファイルのデプロイ

Enterprise JavaBeans (EJB) インターフェイスを使用して、Red Hat Process Automation Manager ディストリビューションの一部として使用する EJB サービスの WAR ファイルを作成して、デプロイできます。

手順

1. 次の例のような起動 Java クラスを使用して、ヒューマンタスクのコールバックを登録します。

```
@Singleton
@Startup
public class StartupBean {

  @PostConstruct
  public void init()
  { System.setProperty("org.jbpm.ht.callback", "jaas"); }

}
```

2. EJB プロジェクトをビルドして、プロジェクト設定に合わせて WAR ファイルを生成します。

3. 生成されたファイルを Red Hat Process Automation Manager が実行している Red Hat JBoss EAP インスタンスにデプロイします。
ランタイムセッションに、**Singleton** ストラテジーは使用しないようにしてください。**Singleton** ストラテジーを使用すると、アプリケーションが下層のファイルシステムから同じ **ksession** インスタンスを複数回読み込み、楽観ロックの例外が発生する可能性があります。

   Red Hat Process Automation Manager が実行中のインスタンスと分離して、Red Hat JBoss EAP インスタンスに EJB WAR ファイルをデプロイする場合は、リモート EJB を呼び出して、セキュリティーコンテキストを伝搬するように、アプリケーションまたはアプリケーションサーバーを設定します。

   Hibernate を使用して Red Hat Process Automation Manager のデータベーススキーマを作成する場合は、Business Central の **persistence.xml** ファイルを更新して、**hibernate.hbm2ddl.auto** プロパティーの値を **create** ではなく、**update** に設定します。

4. 以下の例のように、基本的な Web アプリケーションを作成し、EJB サービスを挿入して、デプロイメントをローカルでテストします。

```
@EJB(lookup = "ejb:/sample-war-ejb-
app/ProcessServiceEJBImpl!org.jbpm.services.ejb.api.ProcessServiceEJBRemote")
private ProcessServiceEJBRemote processService;

@EJB(lookup = "ejb:/sample-war-ejb-
app/UserTaskServiceEJBImpl!org.jbpm.services.ejb.api.UserTaskServiceEJBRemote")
private UserTaskServiceEJBRemote userTaskService;

@EJB(lookup = "ejb:/sample-war-ejb-
app/RuntimeDataServiceEJBImpl!org.jbpm.services.ejb.api.RuntimeDataServiceEJBRemote")

private RuntimeDataServiceEJBRemote runtimeDataService;
```

Red Hat JBoss EAP での EJB アプリケーションの開発およびデプロイに関する詳細は、Developing EJB Applications を参照してください。

第31章 関連情報

- KIE Server の管理とモニタリング
- 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: 2023 年 2 月 1 日

付録B お問い合わせ先

Red Hat Process Automation Manager のドキュメントチーム: brms-docs@redhat.com