Red Hat Process Automation Manager 7.1

Packaging and deploying a Red Hat Process Automation Manager project

Last Updated: 2018-10-10

Red Hat Process Automation Manager 7.1 Packaging and deploying a Red Hat Process Automation Manager project

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

Legal Notice

Abstract

This document describes how to package and deploy a project in Red Hat Process Automation Manager 7.1

Table of Contents

PREFACE

As a business rules developer, you must build and deploy a developed Red Hat Process Automation Manager project to a Process Server in order to begin using the services you have created in Red Hat Process Automation Manager. You can develop and deploy a project from Business Central, from an independent Maven project, from a Java application, or using a combination of various platforms. For example, you can develop a project in Business Central and deploy it using the Process Server REST API, or develop a project in Maven configured with Business Central and deploy it using Business Central.

Prerequisite

The project to be deployed has been developed and tested. For projects in Business Central, consider using test scenarios to test the assets in your project. For example, see *Testing a decision service using test scenarios*.

CHAPTER 1. RED HAT PROCESS AUTOMATION MANAGER PROJECT PACKAGING

Red Hat Process Automation Manager projects contain the business assets that you develop in Red Hat Process Automation Manager. Each project in Red Hat Process Automation Manager is packaged as a Knowledge JAR (KJAR) file with configuration files such as a Maven project object model file (**`pom.xml`**), which contains build, environment, and other information about the project, and a KIE module descriptor file (**`kmodule.xml`**), which contains the KIE base and KIE session configurations for the assets in the project. You deploy the packaged KJAR file to a Process Server that runs the decision services, process applications, and other deployable assets (collectively referred to as *services*) from that KJAR file. These services are consumed at run time through an instantiated KIE container, or *deployment unit*. Project KJAR files are stored in a Maven repository and identified by three values: **`GroupId`**, **`ArtifactId`**, and **`Version`** (GAV). The **`Version`** value must be unique for every new version that might need to be deployed. To identify an artifact (including a KJAR file), you need all three GAV values.

Projects in Business Central are packaged automatically when you build and deploy the projects. For projects outside of Business Central, such as independent Maven projects or projects within a Java application, you must configure the KIE module descriptor settings in an appended **`kmodule.xml`** file or directly in your Java application in order to build and deploy the projects.

CHAPTER 2. PROJECT DEPLOYMENT IN BUSINESS CENTRAL

You can use Business Central to develop your business assets and services and to manage Process Servers configured for project deployment. When your project is developed, you can build the project in Business Central and deploy it automatically to the Process Server. To enable automatic deployment, Business Central includes a built-in Maven repository. From Business Central, you can start, stop, or remove the deployment units (KIE containers) that contain the services and their project versions that you have built and deployed.

You can also connect several Process Servers to the same Business Central instance and group them into different server configurations (in **Menu** → **Deploy** → **Execution Servers**). Servers belonging to the same server configuration run the same services, but you can deploy different projects or different versions of projects on different configurations.

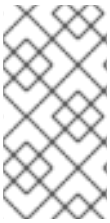
For example, you could have test servers in the **`Test`** configuration and production servers in a **`Production`** configuration. As you develop business assets and services in a project, you deploy the project on the **`Test`** server configuration and then, when a version of the project is sufficiently tested, you can deploy it on the **`Production`** server configuration. In this case, to keep developing the project, change the version in the project settings. Then the new version and the old version are seen as different artifacts in the built-in Maven repository. You can deploy the new version on the **`Test`** server configuration and keep running the old version on the **`Production`** server configuration. This deployment process is simple but has significant limitations. Notably, there is not enough access control: a developer can deploy a project directly into a production environment.

IMPORTANT

You cannot move a Process Server into a different server configuration using Business Central. You must change the configuration file of the server to change the server configuration name for it.

2.1. CONFIGURING A PROCESS SERVER TO CONNECT TO BUSINESS CENTRAL

If a Process Server is not already configured in your Red Hat Process Automation Manager environment, or if you require additional Process Servers in your Red Hat Process Automation Manager environment, you must configure a Process Server to connect to Business Central.



NOTE

If you are deploying Process Server on Red Hat OpenShift Container Platform, see *Deploying a Red Hat Process Automation Manager managed server environment on Red Hat OpenShift Container Platform* for instructions about configuring it to connect to Business Central.

Prerequisite

Process Server is installed. For installation options, see *Planning a Red Hat Process Automation Manager installation*.

Procedure

1. In your Red Hat Process Automation Manager installation directory, navigate to the **`standalone-full.xml`** file. For example, if you use a Red Hat JBoss EAP installation for Red Hat Process Automation Manager, go to

$EAP_HOME/standalone/configuration/standalone-full.xml.

2. Open **standalone-full.xml** and under the **<system-properties>** tag, set the following properties:

   - **org.kie.server.controller.user:** The user name of a user who can log in to the Business Central.
   - **org.kie.server.controller.pwd:** The password of the user who can log in to the Business Central.
   - **org.kie.server.controller:** The URL for connecting to the API of Business Central. Normally, the URL is **http://<centralhost>:<centralport>/business-central/rest/controller**, where **<centralhost>** and **<centralport>** are the host name and port for Business Central. If Business Central is deployed on OpenShift, remove **business-central/** from the URL.
   - **org.kie.server.location:** The URL for connecting to the API of Process Server. Normally, the URL is **http://<serverhost>:<serverport>/kie-server/services/rest/server**, where **<serverhost>** and **<serverport>** are the host name and port for Process Server.
   - **org.kie.server.id:** The name of a server configuration. If this server configuration does not exist in Business Central, it is created automatically when Process Server connects to Business Central.

   Example:

```
<property name="org.kie.server.controller.user"
value="central_user"/>
<property name="org.kie.server.controller.password"
value="central_password"/>
<property name="org.kie.server.controller"
value="http://central.example.com:8080/business-
central/rest/controller"/>
<property name="org.kie.server.location"
value="http://kieserver.example.com:8080/kie-
server/services/rest/server"/>
<property name="org.kie.server.id" value="production-servers"/>
```

3. Start or restart the Process Server.

2.2. CONFIGURING AN EXTERNAL MAVEN REPOSITORY FOR BUSINESS CENTRAL AND PROCESS SERVER

You can configure Business Central and Process Server to use an external Maven repository, such as Nexus, instead of the built-in repository. In this case, when you build projects in Business Central, all built project KJAR files are pushed into this external repository. You can progress these files through the repository as necessary to implement your integration process, and deploy the KJAR files using Business Central or the Process Server REST API.

Prerequisite

Business Central and Process Server are installed. For installation options, see *Planning a Red Hat Process Automation Manager installation*.

Procedure

1. Create a Maven **`settings.xml`** file with connection and access details for your external repository. For details about the **`settings.xml`** file, see the Maven Settings Reference.
2. Save the file in a known location, for example, **`/opt/custom-config/settings.xml`**.
3. In your Red Hat Process Automation Manager installation directory, navigate to the **`standalone-full.xml`** file. For example, if you use a Red Hat JBoss EAP installation for Red Hat Process Automation Manager, go to **`$EAP_HOME/standalone/configuration/standalone-full.xml`**.
4. Open **`standalone-full.xml`** and under the **`<system-properties>`** tag, set the **`kie.maven.settings.custom`** property to the full path name of the **`settings.xml`** file. For example:

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

5. Start or restart Business Central and Process Server.

2.3. BUILDING AND DEPLOYING A PROJECT IN BUSINESS CENTRAL

After your project is developed, you can build the project in Business Central and deploy it to the configured Process Server. Projects in Business Central are packaged automatically as KJARs with all necessary components when you build and deploy the projects.

Procedure

1. In Business Central, go to **Menu** → **Design** → **Projects** and click the project name.
2. In the upper-right corner, click **Deploy** to build and deploy the project.

NOTE

To compile the project without deploying it to Process Server, click **Build**.

If only one Process Server is connected to Business Central, or if all connected Process Servers are in the same server configuration, the services in the project are started automatically in a deployment unit (KIE container).

If multiple server configurations are available, a deployment dialog is displayed in Business Central, prompting you to specify server and deployment details.

3. If the deployment dialog appears, verify or set the following values:
   - **Deployment Unit Id** / **Deployment Unit Alias:** Verify the name and alias of the deployment unit (KIE container) running the service within the Process Server. You normally do not need to change these settings.
   - **Server Configuration:** Select the server configuration for deploying this project. You can later deploy it to other configured servers without rebuilding the project.

- **Start Deployment Unit?:** Verify that this box is selected to start the deployment unit (KIE container). If you clear this box, the service is deployed onto the server but not started.

2.4. DEPLOYMENT UNITS IN BUSINESS CENTRAL

The services in a project are consumed at run time through an instantiated KIE container, or *deployment unit*, on a configured Process Server. When you build and deploy a project in Business Central, the deployment unit is created automatically in the configured server. You can start, stop, or remove deployment units in Business Central as needed. You can also create additional deployment units from previously built projects and start them on existing or new Process Servers configured in Business Central.

2.4.1. Creating a deployment unit in Business Central

One or more deployment units should already exist as part of your Red Hat Process Automation Manager configuration, but if not, you can create a deployment unit from a project that was previously built in Business Central.

Prerequisite

The project or projects for which you are creating the new deployment unit has been built in Business Central.

Procedure

1. In Business Central, go to **Menu** → **Deploy** → **Execution servers**.
2. Under **Server Configurations**, select an existing configuration or click **New Server Configuration** to create a new configuration.
3. Under **Deployment Units**, click **Add Deployment Unit**.
4. In the table within the window, select a GAV and click **Select** next to the GAV to populate the deployment unit data fields.
5. Select the **Start Deployment Unit?** box to start the service immediately, or clear the box to start it later.
6. Click **Finish**.
   The new deployment unit for the service is created and placed on the Process Servers that are configured for this server configuration. If you have selected **Start Deployment Unit?**, the service is started.

2.4.2. Starting, stopping, and removing deployment units in Business Central

When a deployment unit is started, the services in the deployment unit are available for use. If only one Process Server is connected to Business Central, or if all connected Process Servers are in the same server configuration, services are started in a deployment unit automatically when a project is deployed. If multiple server configurations are available, you are prompted upon deployment to specify server and deployment details and to start the deployment unit. However, at any time you can manually start, stop, or remove deployment units in Business Central to manage your deployed services as needed.

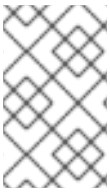
Procedure

1. In Business Central, go to **Menu** → **Deploy** → **Execution servers**.

2. Under **Server Configurations**, select a configuration.
3. Under **Deployment Units**, select a deployment unit.
4. Click **Start**, **Stop**, or **Remove** in the upper-right corner. To remove a running deployment unit, stop it and then remove it.

2.5. EDITING THE GAV VALUES FOR A PROJECT IN BUSINESS CENTRAL

The **`GroupId`**, **`ArtifactId`**, and **`Version`** (GAV) values identify a project in a Maven repository. When Business Central and Process Server are on the same file system and use the same Maven repository, the project is automatically updated in the repository each time you build a new version of your project. However, if Business Central and Process Server are on separate file systems and use separate local Maven repositories, you must update a project GAV value, usually the version, for any new versions of the project to ensure that the project is seen as a different artifact alongside the old version.



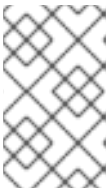
NOTE

For development purposes only, you can add the **`SNAPSHOT`** suffix in the project version to instruct Maven to get a new snapshot update according to the Maven policy. Do not use the **`SNAPSHOT`** suffix for a production environment.

You can set the GAV values in the project **Settings** screen.

Procedure

1. In Business Central, go to **Menu** → **Design** → **Projects** and click the project name.
2. Click the project **Settings** tab.
3. In **General Settings**, modify the **Group ID**, **Artifact ID**, or **Version** fields as necessary. If you have deployed the project and are developing a new version, usually you need to increase the version number.



NOTE

For development purposes only, you can add the **`SNAPSHOT`** suffix in the project version to instruct Maven to get a new snapshot update according to the Maven policy. Do not use the **`SNAPSHOT`** suffix for a production environment.

4. Click **Save** to finish.

2.6. DUPLICATE GAV DETECTION IN BUSINESS CENTRAL

In Business Central, all Maven repositories are checked for any duplicated **`GroupId`**, **`ArtifactId`**, and **`Version`** (GAV) values in a project. If a GAV duplicate exists, the performed operation is canceled.

Duplicate GAV detection is executed every time you perform the following operations:

- Save a project definition for the project.
- Save the **`pom.xml`** file.

- Install, build, or deploy a project.

The following Maven repositories are checked for duplicate GAVs:

- Repositories specified in the **<repositories>** and **<distributionManagement>** elements of the **pom.xml** file.
- Repositories specified in the Maven **settings.xml** configuration file.

2.6.1. Managing duplicate GAV detection settings in Business Central

Business Central users with the **admin** role can modify the list of repositories that are checked for duplicate **GroupId**, **ArtifactId**, and **Version** (GAV) values for a project.

Procedure

1. In Business Central, go to **Menu** → **Design** → **Projects** and click the project name.
2. Click the project **Settings** tab and then click **Validation** to open the list of repositories.
3. Select or clear any of the listed repository options to enable or disable duplicate GAV detection. In the future, duplicate GAVs will be reported for only the repositories you have enabled for validation.

NOTE

To disable this feature, set the **org.guvnor.project.gav.check.disabled** system property to **true** for Business Central at system startup:

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

CHAPTER 3. PROJECT DEPLOYMENT WITHOUT BUSINESS CENTRAL

As an alternative to developing and deploying projects in the Business Central interface, you can use independent Maven projects or your own Java applications to develop Red Hat Process Automation Manager projects and deploy them in KIE containers (deployment units) to a configured Process Server. You can then use the Process Server REST API to start, stop, or remove the KIE containers that contain the services and their project versions that you have built and deployed. This flexibility enables you to continue to use your existing application work flow to develop business assets using Red Hat Process Automation Manager features.

Projects in Business Central are packaged automatically when you build and deploy the projects. For projects outside of Business Central, such as independent Maven projects or projects within a Java application, you must configure the KIE module descriptor settings in an appended **kmodule.xml** file or directly in your Java application in order to build and deploy the projects.

3.1. CONFIGURING A KIE MODULE DESCRIPTOR FILE

A KIE module is a Maven project or module with an additional metadata file **META-INF/kmodule.xml**. All Red Hat Process Automation Manager projects require a **kmodule.xml** file in order to be properly packaged and deployed. This **kmodule.xml** file is a KIE module descriptor that defines the KIE base and KIE session configurations for the assets in a project. A KIE base is a repository that contains all rules, processes, and other business assets in Red Hat Process Automation Manager but does not contain any runtime data. A KIE session stores and executes runtime data and is created from a KIE base or directly from a KIE container if you have defined the KIE session in the **kmodule.xml** file.

If you create projects outside of Business Central, such as independent Maven projects or projects within a Java application, you must configure the KIE module descriptor settings in an appended **kmodule.xml** file or directly in your Java application in order to build and deploy the projects.

Procedure

1. In the **~/resources/META-INF** directory of your project, create a **kmodule.xml** metadata file with at least the following content:

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

   This empty **kmodule.xml** file is sufficient to produce a single default KIE base that includes all files found under your project **resources** path. The default KIE base also includes a single default KIE session that is triggered when you create a KIE container in your application at build time.

   The following example is a more advanced **kmodule.xml** file:

```
<?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>
```

This example defines two KIE bases. Specific **packages** of rule assets are included with both KIE bases. When you specify packages in this way, you must organize your rule files in a folder structure that reflects the specified packages. Two KIE sessions are instantiated from the **KBase1** KIE base, and one KIE session from **KBase2**. The KIE session from **KBase2** is a **stateless** KIE session, which means that data from a previous invocation of the KIE session (the previous session state) is discarded between session invocations. That KIE session also specifies a file (or a console) logger, a **WorkItemHandler**, and listeners of the three supported types shown: **ruleRuntimeEventListener**, **agendaEventListener** and **processEventListener**. The **<configuration>** element defines optional properties that you can use to further customize your **kmodule.xml** file.

As an alternative to manually appending a **kmodule.xml** file to your project, you can use a **KieModuleModel** instance within your Java application to programatically create a **kmodule.xml** file that defines the KIE base and a KIE session, and then add all resources in your project to the KIE virtual file system **KieFileSystem**.

Creating kmodule.xml programmatically and adding it toKieFileSystem

```
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. After you configure the **kmodule.xml** file either manually or programmatically in your project, retrieve the KIE bases and KIE sessions from the KIE container to verify the configurations:

```
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");
```

If **KieBase** or **KieSession** have been configured as **default="true"** in the **kmodule.xml** file, as in the previous **kmodule.xml** example, you can retrieve them from the KIE container without passing any names:

```
KieContainer kContainer = ...

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

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

For more information about the **kmodule.xml** file, download the **Red Hat Process Automation Manager [VERSION] Source Distribution** ZIP file from the Red Hat Customer Portal and see the **kmodule.xsd** XML schema located at **$FILE_HOME/rhpam-$VERSION-sources/kie-api-parent-$VERSION/kie-api/src/main/resources/org/kie/api/**.

3.1.1. KIE module configuration properties

The optional **<configuration>** element in the KIE module descriptor file (**kmodule.xml**) of your project defines property **key** and **value** pairs that you can use to further customize your **kmodule.xml** file.

Example configuration property in a `kmodule.xml` file

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

The following are the **`<configuration>`** property keys and values supported in the KIE module descriptor file (**`kmodule.xml`**) for your project:

drools.dialect.default

Sets the default Drools dialect.
Supported values: **`java`**, **`mvel`**

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

drools.accumulate.function.$FUNCTION

Links a class that implements an accumulate function to a specified function name, which allows you to add custom accumulate functions into the process engine.

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

drools.evaluator.$EVALUATION

Links a class that implements an evaluator definition to a specified evaluator name so that you can add custom evaluators into the process engine. An evaluator is similar to a custom operator.

```
<property key="drools.evaluator.soundslike"

value="org.drools.core.base.evaluators.SoundslikeEvaluatorsDefinition"/>
```

drools.dump.dir

Sets a path to the Red Hat Process Automation Manager **`dump/log`** directory.

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

drools.defaultPackageName

Sets a default package for the business assets in your project.

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

drools.parser.processStringEscapes

Sets the String escape function. If this property is set to **false**, the **\n** character will not be interpreted as the newline character.
Supported values: **true** (default), **false**

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

drools.kbuilder.severity.$DUPLICATE

Sets a severity for instances of duplicate rules, processes, or functions reported when a KIE base is built. For example, if you set **duplicateRule** to **ERROR**, then an error is generated for any duplicated rules detected when the KIE base is built.
Supported key suffixes: **duplicateRule**, **duplicateProcess**, **duplicateFunction**

Supported values: **INFO**, **WARNING**, **ERROR**

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

drools.propertySpecific

Sets the property reactivity of the process engine.
Supported values: **DISABLED**, **ALLOWED**, **ALWAYS**

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

drools.lang.level

Sets the DRL language level.
Supported values: **DRL5**, **DRL6**, **DRL6_STRICT** (default)

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

3.1.2. KIE base attributes supported in KIE modules

A KIE base is a repository that you define in the KIE module descriptor file (**kmodule.xml**) for your project and contains all rules, processes, and other business assets in Red Hat Process Automation Manager. When you define KIE bases in the **kmodule.xml** file, you can specify certain attributes and values to further customize your KIE base configuration.

Example KIE base configuration in a `kmodule.xml` file

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

The following are the **kbase** attributes and values supported in the KIE module descriptor file (**kmodule.xml**) for your project:

Table 3.1. KIE base attributes supported in KIE modules

Attribute	Supported values	Description
name	Any name	Defines the name that retrieves **KieBase** from **KieContainer**. *This attribute is mandatory*.
includes	Comma-separated list of other KIE base objects in the KIE module	Defines other KIE base objects and artifacts to be included in this KIE base. A KIE base can be contained in multiple KIE modules if you declare it as a dependency in the **pom.xml** file of the modules.
packages	Comma-separated list of packages to include in the KIE base Default: **all**	Defines packages of artifacts (such as rules and processes) to be included in this KIE base. By default, all artifacts in the **~/resources** directory are included into a KIE base. This attribute enables you to limit the number of compiled artifacts. Only the packages belonging to the list specified in this attribute are compiled.
default	**true**, **false** Default: **false**	Determines whether a KIE base is the default KIE base for a module so that it can be created from the KIE container without passing any name. Each module can have only one default KIE base.
equalsBehavior	**identity**, **equality** Default: **identity**	Defines the behavior of Red Hat Process Automation Manager when a new fact is inserted into the working memory. If set to **identity**, a new **FactHandle** is always created unless the same object is already present in the working memory. If set to **equality**, a new **FactHandle** is created only if the newly inserted object is not equal, according to its **equals()** method, to an existing fact.
eventProcessingMode	**cloud**, **stream** Default: **cloud**	Determines how events are processed in the KIE base. If this property is set to **cloud**, the KIE base treats events as normal facts. If this property is set to **stream**, temporal reasoning on events is allowed.

Attribute	Supported values	Description
declarativeAgenda	**disabled**, **enabled** Default: **disabled**	Determines whether the declarative agenda is enabled or not.

3.1.3. KIE session attributes supported in KIE modules

A KIE session stores and executes runtime data and is created from a KIE base or directly from a KIE container if you have defined the KIE session in the KIE module descriptor file (**kmodule.xml**) for your project. When you define KIE bases and KIE sessions in the **kmodule.xml** file, you can specify certain attributes and values to further customize your KIE session configuration.

Example KIE session configuration in a `kmodule.xml` file

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

The following are the **ksession** attributes and values supported in the KIE module descriptor file (**kmodule.xml**) for your project:

Table 3.2. KIE session attributes supported in KIE modules

Attribute	Supported values	Description
name	Any name	Defines the name that retrieves **KieSession** from **KieContainer**. *This attribute is mandatory*.
type	**stateful**, **stateless** Default: **stateful**	Determines whether data is retained (**stateful**) or discarded (**stateless**) between invocations of the KIE session. A session set to **stateful** enables you to iteratively work with the working memory, while a session set to **stateless** is typically used for one-off execution of assets. A **stateless** session stores a knowledge state that is changed every time a new fact is added, updated, or deleted, and every time a rule is executed. An execution in a **stateless** session has no information about previous actions, such rule executions.

Attribute	Supported values	Description
default	**true**, **false** Default: **false**	Determines whether a KIE session is the default session for a module so that it can be created from the KIE container without passing any name. Each module can have only one default KIE session.
clockType	**realtime**, **pseudo** Default: **realtime**	Determines whether event time stamps are assigned by the system clock or by a pseudo clock controlled by the application. This clock is especially useful for unit testing on temporal rules.
beliefSystem	**simple**, **jtms**, **defeasible** Default: **simple**	Defines the type of belief system used by the KIE session. A belief system deduces the truth from knowledge (facts). For example, if a new fact is inserted based on another fact which is later removed from the process engine, the system can determine that the newly inserted fact should be removed as well.

3.2. PACKAGING AND DEPLOYING A RED HAT PROCESS AUTOMATION MANAGER PROJECT IN MAVEN

If you want to deploy a Maven project outside of Business Central to a configured Process Server, you can edit the project **pom.xml** file to package your project as a KJAR file and add a **kmodule.xml** file with the KIE base and KIE session configurations for the assets in your project.

Prerequisites

- You have a Mavenized project that contains Red Hat Process Automation Manager business assets.
- Process Server is installed and **kie-server** user access is configured. For installation options, see *Planning a Red Hat Process Automation Manager installation*.

Procedure

1. In the **pom.xml** file of your Maven project, set the packaging type to **kjar** and add the **kie-maven-plugin** build component:

```
<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>
```

The **kjar** packaging type activates the **kie-maven-plugin** component to validate and pre-compile artifact resources. The **<version>** is the Maven artifact version for Red Hat Process Automation Manager currently used in your project (for example, 7.11.0.Final-redhat-00002). These settings are required to properly package the Maven project for deployment.

NOTE

Instead of specifying a Red Hat Process Automation Manager **<version>** for individual dependencies, consider adding the Red Hat Business Automation bill of materials (BOM) dependency to your project **pom.xml** file. The Red Hat Business Automation BOM applies to both Red Hat Decision Manager and Red Hat Process Automation Manager. When you add the BOM files, the correct versions of transitive dependencies from the provided Maven repositories are included in the project.

Example BOM dependency:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.1.0.GA-redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

For more information about the Red Hat Business Automation BOM, see What is the mapping between Red Hat Process Automation Manager and the Maven library version?.

2. Add the following dependencies to the **pom.xml** file to generate an executable rule model from your rule assets:

   - **drools-canonical-model**: Enables an executable canonical representation of a rule set model that is independent from Red Hat Process Automation Manager
   - **drools-model-compiler**: Compiles the executable model into Red Hat Process Automation Manager internal data structures so that it can be executed by the process engine

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

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

Executable rule models are embeddable models that provide a Java-based representation of a rule set for execution at build time. The executable model is a more efficient alternative to the standard asset packaging in Red Hat Process Automation Manager and enables KIE containers and KIE bases to be created more quickly, especially when you have large lists of DRL (Drools Rule Language) files and other Red Hat Process Automation Manager assets.

For more information about executable rule models, see "Executable rule models" in *Designing a decision service using DRL rules*.

3. In the **~/resources** directory of your Maven project, create a **META-INF/kmodule.xml** metadata file with at least the following content:

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

This **kmodule.xml** file is a KIE module descriptor that is required for all Red Hat Process Automation Manager projects. You can use the KIE module to define one or more KIE bases and one or more KIE sessions from each KIE base.

For more information about **kmodule.xml** configuration, see Section 3.1, “Configuring a KIE module descriptor file”.

4. In the relevant resource in your Maven project, configure a **.java** class to create a KIE container and a KIE session to load the KIE base:

```
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();

}
```

In this example, the KIE container reads the files to be built from the class path for a **testApp** project. The **KieServices** API enables you to access all KIE building and runtime configurations.

You can also create the KIE container by passing the project **ReleaseId** to the **KieServices** API. The **ReleaseId** is generated from the **GroupId**, **ArtifactId**, and **Version** (GAV) values in the project **pom.xml** file.

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

public void testApp() {

  // Identify the project in the local repository:
```

```
    ReleaseId rid = new ReleaseId();
    rid.setGroupId("com.sample");
    rid.setArtifactId("my-app");
    rid.setVersion("1.0.0");

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

}
```

5. In a command terminal, navigate to your Maven project directory and run the following command to build the project from an executable model:

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

The **-DgenerateModel=<VALUE>** property enables the project to be built as a model-based KJAR instead of a DRL-based KJAR, so that rule assets are built in an executable rule model.

Replace **<VALUE>** with one of three values:

- **YES**: Generates the executable model corresponding to the DRL files in the original project and excludes the DRL files from the generated KJAR.
- **WITHDRL**: Generates the executable model corresponding to the DRL files in the original project and also adds the DRL files to the generated KJAR for documentation purposes (the KIE base is built from the executable model regardless).
- **NO**: Does not generate the executable model.

Example build command:

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

If the build fails, address any problems described in the command line error messages and try again to validate the files until the build is successful.

6. After you successfully build and test the project locally, deploy the project to the remote Maven repository:

```
mvn deploy
```

3.3. PACKAGING AND DEPLOYING A RED HAT PROCESS AUTOMATION MANAGER PROJECT IN A JAVA APPLICATION

If you want to deploy a project from within your own Java application to a configured Process Server, you can use a **KieModuleModel** instance to programatically create a **kmodule.xml** file that defines the KIE base and a KIE session, and then add all resources in your project to the KIE virtual file system **KieFileSystem**.

Prerequisites

- You have a Java application that contains Red Hat Process Automation Manager business assets.
- Process Server is installed and **`kie-server`** user access is configured. For installation options, see *Planning a Red Hat Process Automation Manager installation*.

Procedure

1. In your client application, add the following dependencies to the relevant classpath of your Java project to generate an executable rule model from your rule assets:
   - **`drools-canonical-model`**: Enables an executable canonical representation of a rule set model that is independent from Red Hat Process Automation Manager
   - **`drools-model-compiler`**: Compiles the executable model into Red Hat Process Automation Manager internal data structures so that it can be executed by the process engine

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

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

   Executable rule models are embeddable models that provide a Java-based representation of a rule set for execution at build time. The executable model is a more efficient alternative to the standard asset packaging in Red Hat Process Automation Manager and enables KIE containers and KIE bases to be created more quickly, especially when you have large lists of DRL (Drools Rule Language) files and other Red Hat Process Automation Manager assets.

   For more information about executable rule models, see "Executable rule models" in *Designing a decision service using DRL rules*.

   The **`<version>`** is the Maven artifact version for Red Hat Process Automation Manager currently used in your project (for example, 7.11.0.Final-redhat-00002).

NOTE

Instead of specifying a Red Hat Process Automation Manager **<version>** for individual dependencies, consider adding the Red Hat Business Automation bill of materials (BOM) dependency to your project **pom.xml** file. The Red Hat Business Automation BOM applies to both Red Hat Decision Manager and Red Hat Process Automation Manager. When you add the BOM files, the correct versions of transitive dependencies from the provided Maven repositories are included in the project.

Example BOM dependency:

```
<dependency>
  <groupId>com.redhat.ba</groupId>
  <artifactId>ba-platform-bom</artifactId>
  <version>7.1.0.GA-redhat-00002</version>
  <scope>import</scope>
  <type>pom</type>
</dependency>
```

For more information about the Red Hat Business Automation BOM, see What is the mapping between Red Hat Process Automation Manager and the Maven library version?.

2. Use the **KieServices** API to create a **KieModuleModel** instance with the desired KIE base and KIE session. The **KieServices** API enables you to access all KIE building and runtime configurations. The **KieModuleModel** instance generates the **kmodule.xml** file for your project.
   For more information about **kmodule.xml** configuration, see Section 3.1, “Configuring a KIE module descriptor file”.

3. Convert your **KieModuleModel** instance into XML and add the XML to **KieFileSystem**.

Creating kmodule.xml programmatically and adding it toKieFileSystem

```
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. Add any remaining Red Hat Process Automation Manager assets that you use in your project to your **KieFileSystem** instance. The artifacts must be in a Maven project file structure.

```
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))
;
```

In this example, the project assets are added both as a **String** variable and as a **Resource** instance. You can create the **Resource** instance using the **KieResources** factory, also provided by the **KieServices** instance. The **KieResources** class provides factory methods to convert **InputStream**, **URL**, and **File** objects, or a **String** representing a path of your file system to a **Resource** instance that the **KieFileSystem** can manage.

You can also explicitly assign a **ResourceType** property to a **Resource** object when you add project artifacts to **KieFileSystem**:

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

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

5. Use **KieBuilder** with **buildAll(ExecutableModelProject.class)** specified to build the content of **KieFileSystem** from an executable model, and create a KIE container to deploy it:

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

```
  KieContainer kieContainer = kieServices

.newKieContainer(kieServices.getRepository().getDefaultReleaseId());
```

After **KieFileSystem** is built from the executable model, the resulting **KieSession** uses constraints based on lambda expressions instead of less-efficient **mvel** expressions. To build the project in the standard method without an executable model, do not specify any argument in **buildAll()**.

A build **ERROR** indicates that the project compilation failed, no **KieModule** was produced, and nothing was added to the **KieRepository** singleton. A **WARNING** or an **INFO** result indicates that the compilation of the project was successful, with information about the build process.

3.4. STARTING A SERVICE IN PROCESS SERVER

If you have deployed Red Hat Process Automation Manager assets from a Maven or Java project outside of Business Central, you use a Process Server REST API call to start the KIE container (deployment unit) and the services in it. You can use the Process Server REST API to start services regardless of your deployment type, including deployment from Business Central, but projects deployed from Business Central either are started automatically or can be started within the Business Central interface.

Prerequisite

Process Server is installed and **kie-server** user access is configured. For installation options, see *Planning a Red Hat Process Automation Manager installation*.

Procedure

In your command terminal, run the following API request to load a service into a KIE container in the Process Server and to start it:

```
$ 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>
```

Replace the following values:

- **<username>**, **<password>:** The user name and password of a user with **kie-server** role.
- **<containerID>:** The identifier for the KIE container (deployment unit). You can use any random identifier but it must be the same in both places in the command (the URL and the data).
- **<groupID>**, **<artifactID>**, **<version>:** The project GAV values.
- **<serverhost>:** The host name for the Process Server, or **localhost** if you are running the command on the same host as the Process Server.
- **<serverport>:** The port number for the Process Server.

Example:

```
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
```

3.5. STOPPING AND REMOVING A SERVICE IN PROCESS SERVER

If you have started Red Hat Process Automation Manager services from a Maven or Java project outside of Business Central, you use a Process Server REST API call to stop and remove the KIE container (deployment unit) containing the services. You can use the Process Server REST API to stop services regardless of your deployment type, including deployment from Business Central, but services from Business Central can also be stopped within the Business Central interface.

Prerequisite

Process Server is installed and **`kie-server`** user access is configured. For installation options, see *Planning a Red Hat Process Automation Manager installation*.

Procedure

In your command terminal, run the following API request to stop and remove a KIE container with services on Process Server:

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

Replace the following values:

- **<username>**, **<password>:** The user name and password of a user with **`kie-server`** role.
- **<containerID>:** The identifier for the KIE container (deployment unit). You can use any random identifier but it must be the same in both places in the command (the URL and the data).
- **<serverhost>:** The host name for the Process Server, or **`localhost`** if you are running the command on the same host as the Process Server.
- **<serverport>:** The port number for the Process Server.

Example:

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

CHAPTER 4. ADDITIONAL RESOURCES

- "Executing rules" in *Designing a decision service using DRL rules*
- *Deploying a Red Hat Process Automation Manager authoring environment on Red Hat OpenShift Container Platform*
- *Deploying a Red Hat Process Automation Manager managed server environment on Red Hat OpenShift Container Platform*
- *Deploying a Red Hat Process Automation Manager immutable server environment on Red Hat OpenShift Container Platform*

APPENDIX A. VERSIONING INFORMATION

Documentation last updated on Friday, October 12, 2018.