Red Hat JBoss BRMS 6.4

Administration and Configuration Guide

The Administration and Configuration Guide for Red Hat JBoss BRMS

Last Updated: 2018-12-26

Red Hat JBoss BRMS 6.4 Administration and Configuration Guide

The Administration and Configuration Guide for Red Hat JBoss BRMS

Red Customer Content Services
brms-docs@redhat.com

Emily Murphy

Gemma Sheldon

Michele Haglund

Mikhail Ramendik

Stetson Robinson

Vidya Iyengar

Legal Notice

Abstract

A guide for administrators and advanced users dealing with Red Hat JBoss BRMS setup, configuration, and advanced usage.

Table of Contents

PART I. INTRODUCTION

CHAPTER 1. ASSET REPOSITORY

Business Rules and other assets and resources created in Business Central are stored in asset repository, which is otherwise known as the Knowledge Store.

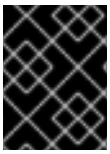
The Knowledge Store is a centralized repository for your business knowledge. The Knowledge Store connects to the Git repository to store various knowledge assets and artifacts at a single location. Business Central provides a web front-end that allows you to view and update the stored content. You can access the content by using the Project Explorer from the unified environment of Red Hat JBoss BRMS.

All business assets are stored in repositories. These repositories are then saved in directories called organizational units. By default, the Knowledge Store in Business Central does not contain any organizational unit. Therefore, to be able to create your own business assets, you need to create an organizational unit and a repository first.

1.1. CREATING AN ORGANIZATIONAL UNIT

It is possible to create an organizational unit either in the **Administration** perspective of Business Central, using the **`kie-config-cli`** tool, or the REST API calls.

Creating an Organizational Unit in Business Central



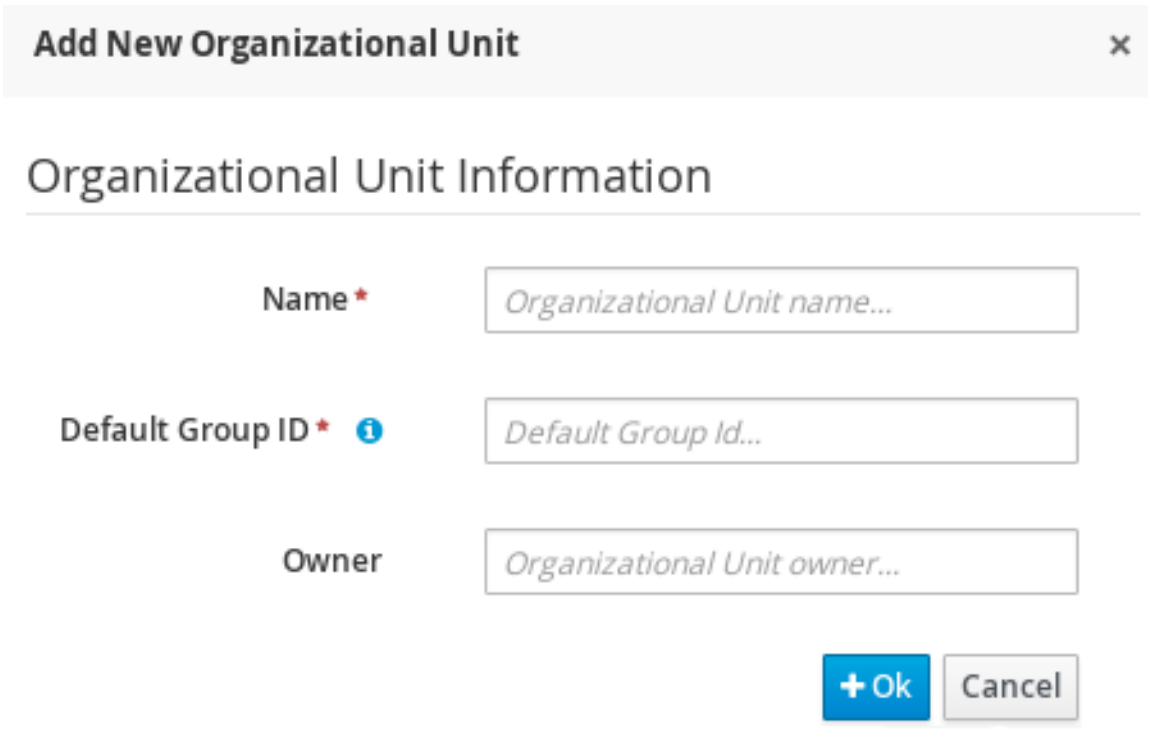
IMPORTANT

Note that only users with the **`admin`** role in Business Central can create organizational units.

Procedure: Using Business Central to Create an Organizational Unit

1. In Business Central, go to **Authoring** → **Administration**.
2. On the perspective menu, click **Organizational Units** → **Manage Organizational Units**.
3. In the **Organization Unit Manager** view, click **Add**.
   The **Add New Organizational Unit** dialog window opens.

Figure 1.1. Add New Organizational Unit_Dialog Window



4. Enter the two mandatory parameters (name and default group ID) and click **Ok**.

Creating an Organizational Unit Using the kie-config-cli Tool

Organizational units can be created using the **`kie-config-cli`** tool as well. To do so, run the **`create-org-unit`** command. The tool then guides you through the entire process of creating an organizational unit by asking for other required parameters. Type **`help`** for a list of all commands.

For more information about the **`kie-config-cli`** tool, see Chapter 6, *Command Line Configuration*.

Creating an Organizational Unit Using the REST API

IMPORTANT

Note that only users with the **`rest-all`** role can create organizational units.

To create an organizational unit in Knowledge Store, issue the **`POST`** REST API call. Details of the organizational unit are defined by the JSON entity.

Input parameter of the call is a **`OrganizationalUnit`** instance. Call returns a **`CreateOrganizationalUnitRequest`** instance.

Example 1.1. Creating an Organizational Unit Using the Curl Utility

Example JSON entity containing details of an organizational unit to be created:

```
{
  "name"         : "helloWorldUnit",
  "owner"        : "tester",
  "description"  :  null,
  "repositories" :  []
}
```

Execute the following command:

```
curl -X POST 'localhost:8080/business-central/rest/organizationalunits/'
-u USERNAME:PASSWORD -H 'Accept: application/json' -H 'Content-Type:
application/json' -d
'{"name":"helloWorldUnit","owner":"tester","description":null,"repositor
ies":[]}'
```

For further information, refer to chapter *Knowledge Store REST API*, section *Organizational Unit Calls* of *Red Hat JBoss BPM Suite Development Guide*.

1.2. CREATING A REPOSITORY

There are three ways to create a repository: through the **Administration** perspective of Business Central, the **`kie-config-cli`** tool, or using the REST API calls.

Creating a Repository in Business Central

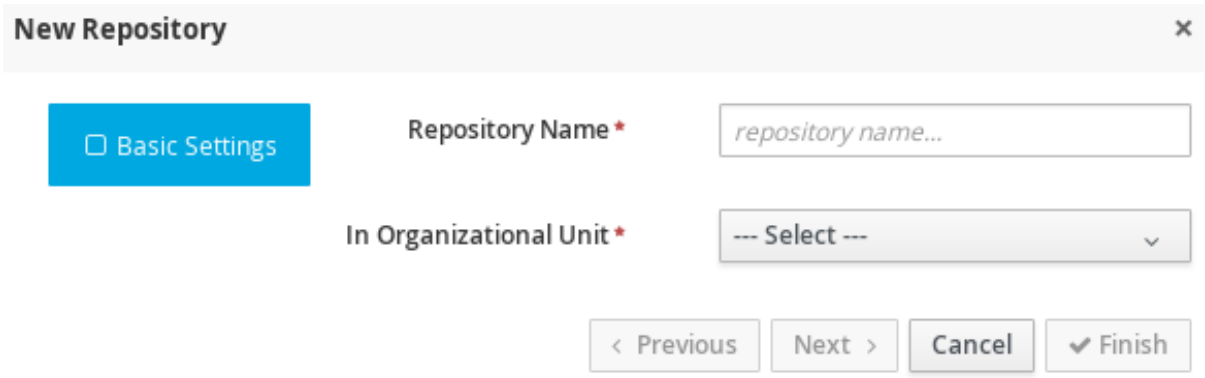
IMPORTANT

Note that only users with the **`admin`** role in Business Central can create repositories.

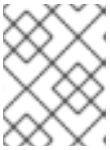
Procedure: Using Business Central to Create a Repository

1. In Business Central, go to **Authoring** → **Administration**.
2. On the perspective menu, click **Repositories** → **New repository**.
   The **New Repository** pop-up window is displayed.

   Figure 1.2. New Repository Dialog Window



3. Specify the two mandatory parameters:
   - **Repository name**



NOTE

Make sure that the repository name is a valid file name. Avoid using a space or any special character that might lead to an invalid name.

   - **Organizational unit**: Specifies the location of the newly created repository.

4. Click **Finish**.

You can view the newly created repository either in the **File Explorer** or the **Project Explorer**.

Creating Repository Using kie-config-cli Tool
To create a new Git repository using the **`kie-config-cli`** tool, run the **`create-repo`** command. The tool then guides you through the entire process of creating a repository by asking for other required parameters. Type **`help`** for a list of all commands.

For more information about the **`kie-config-cli`** tool, see Chapter 6, *Command Line Configuration*.

Creating Repository Using REST API

IMPORTANT

Note that only users with the **`rest-all`** role can create repositories.

To create a repository in the Knowledge Store, issue the **`POST`** REST API call. Details of the repository are defined by the JSON entity. Make sure you established an authenticated HTTP session before executing this call.

Input parameter of the call is a **`RepositoryRequest`** instance. Call returns a **`CreateOrCloneRepositoryRequest`** instance.

Example 1.2. Creating Repository Using Curl Utility

Example JSON entity containing details of a repository to be created:

```
{
  "name"                  : "newRepository",
  "description"           :  null,
  "gitURL"                :  null,
  "requestType"           : "new",
  "organizationalUnitName" : "helloWorldUnit"
}
```

Execute the following command:

```
curl -X POST 'localhost:8080/business-central/rest/repositories/' -u
USERNAME:PASSWORD -H 'Accept: application/json' -H 'Content-Type:
application/json' -d
'{"name":"newRepository","description":null,"requestType":"new","gitURL"
:null,"organizationalUnitName":"helloWorldUnit"}'
```

For further information, refer to chapter *Knowledge Store REST API*, section *Repository Calls* of *Red Hat JBoss BPM Suite Development Guide*.

1.3. CLONING A REPOSITORY

It is possible to clone a repository either in Business Central or using the REST API calls. The **`kie-config-cli`** tool cannot be used to clone arbitrary repositories. Run **`git clone`**, or use one of the following options instead:

Cloning a Repository in Business Central

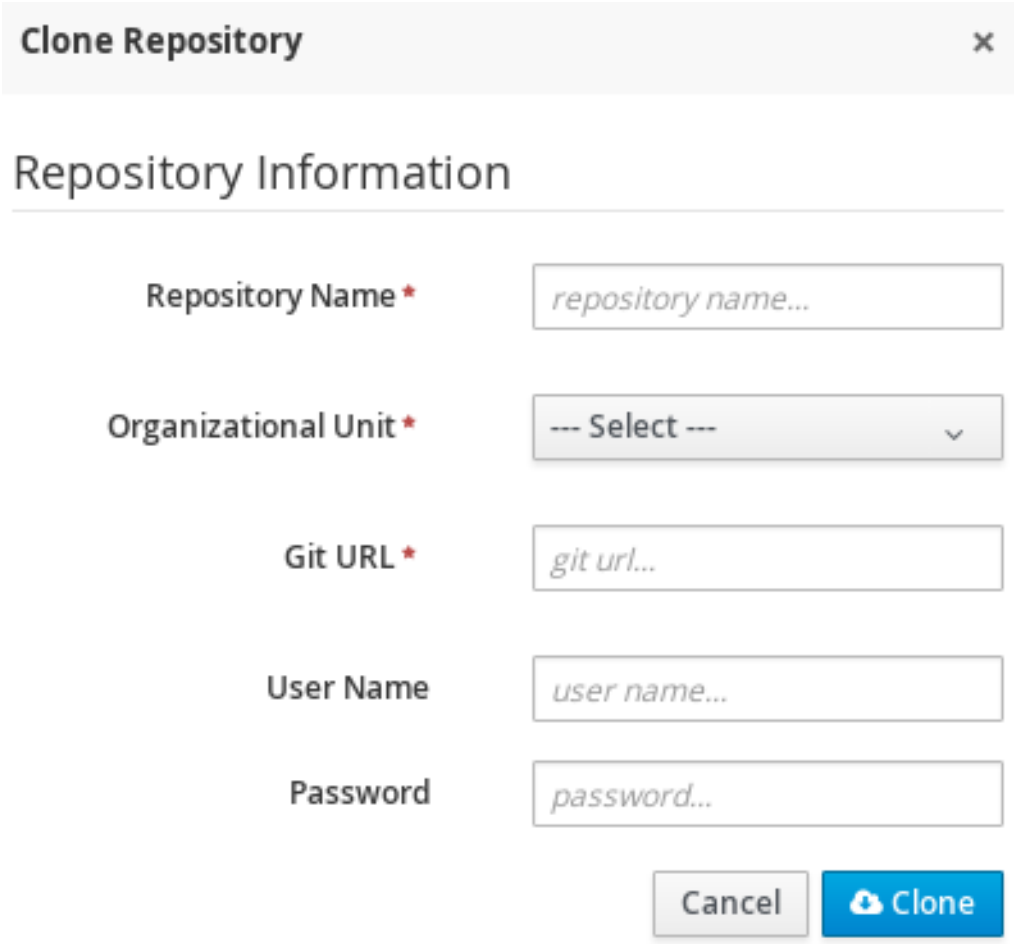
IMPORTANT

Note that only users with the **admin** role in Business Central can clone repositories.

Procedure: Using Business Central to Clone a Repository

1. In Business Central, go to **Authoring** → **Administration**.
2. On the perspective menu, choose **Repositories** → **Clone repository**.
   The **Clone Repository** pop-up window is displayed.

   Figure 1.3. Clone Repository Dialog Window



3. In the **Clone Repository** dialog window, enter the repository details:
   a. Enter the **Repository Name** to be used as the repository identifier in the Asset repository and select the **Organizational Unit** it should be added to.
   b. Enter the URL of the Git repository:
      - For a local repository, use `file:///PATH_TO_REPOSITORY/REPOSITORY_NAME`.
      - For a remote or preexisting repository, use `https://github.com/USERNAME/REPOSITORY_NAME.git` or `git://HOST_NAME/REPOSITORY_NAME`.

IMPORTANT

It is important to use the HTTPS or Git protocol instead of a SCP-style SSH URL. Business Central does not support the basic SSH URL and fails with **`Invalid URL format`**.

NOTE

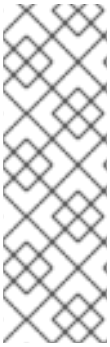
The file protocol is only supported for READ operations. WRITE operations are *not* supported.

c. If applicable, enter the **User Name** and **Password** of your Git account to be used for authentication.

4. Click **Clone**.
   A confirmation prompt with the notification that the repository was created successfully is displayed.

5. Click **Ok**.
   The repository is now being indexed. Some workbench features may be unavailable until the indexing has completed.

You can view the cloned repository either in the **File Explorer** or the **Project Explorer**.



NOTE

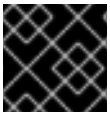
If you are deploying Business Central on WebLogic server, set the following Java system property in the **`setDomainEnv.sh`** file (for Linux) or **`setDomainEnv.cmd`** file (for Windows):

```
JAVA_OPTIONS="%JAVA_OPTIONS% -DUseSunHttpHandler=true"
```

This enables the WebLogic server to use the HTTP handlers.

Cloning a Repository Using the REST API
To clone a repository, issue the **`POST`** REST API call. This call creates or clones (according to the value of the **`requestType`** parameter) the repository defined by the JSON entity.

The input parameter of the call is a **`RepositoryRequest`** instance. The Call returns a **`CreateOrCloneRepositoryRequest`** instance.



IMPORTANT

Note that, only users with the **`rest-all`** role can clone repositories.

Example 1.3. Cloning a Repository Using the Curl Utility

Example JSON entity containing details of a repository to be cloned:

```
{
  "name"                  : "clonedRepository",
  "description"           :  null,
  "requestType"           : "clone",
```

```
  "gitURL"                : "git://localhost:9418/newRepository",
  "organizationalUnitName" : "helloWorldUnit"
}
```

Execute the following command:

```
curl -X POST 'localhost:8080/business-central/rest/repositories/' -u
USERNAME:PASSWORD -H 'Accept: application/json' -H 'Content-Type:
application/json' -d
'{"name":"clonedRepository","description":null,"requestType":"clone","gi
tURL":"git://localhost:9418/newRepository","organizationalUnitName":"hel
loWorldUnit"}'
```

For further information, refer to chapter *Knowledge Store REST API*, section *Repository Calls* of *Red Hat JBoss BPM Suite Development Guide*.

1.4. REMOVING A REPOSITORY

Repositories can be removed using any of the following procedures.

Removing a Repository in Business Central

The simplest way to remove a repository is using the **RepositoryEditor** in Business Central.

Procedure: Using Business Central to Remove a Repository

1. In Business Central, go to **Authoring** → **Administration**.
2. Select **Repositories** from the tree menu on the left.
3. In the **RepositoryEditor** on the right side of the page, locate the repository you want to delete from the list of available repositories.
4. From the drop-down menu, select **master** → **Delete**.
   The following message will appear:

   ```
   Are you sure you want to remove Repository "REPOSITORY_NAME"? Some
   editors may become inoperable if their content is inaccessible.
   ```

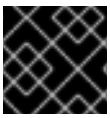
5. Press **OK** to delete the repository.

Removing a Repository Using the kie-config-cli Tool

Repositories can be removed using the **`kie-config-cli`** tool as well. To do so, run the **`remove-repo`** command.

For further information about the **`kie-config-cli`** tool, see Chapter 6, *Command Line Configuration*.

Removing a Repository Using the REST API



IMPORTANT

Note that only users with the **`rest-all`** role can remove repositories.

To remove a repository from the Knowledge Store, issue the **DELETE** REST API call. Make sure you established an authenticated HTTP session before executing this call.

The call returns a **RemoveRepositoryRequest** instance.

Example 1.4. Removing a Repository Using the Curl Utility

Execute the following command:

```
curl -X DELETE 'localhost:8080/business-
central/rest/repositories/REPOSITORY_NAME' -u USERNAME:PASSWORD -H
'Accept: application/json' -H 'Content-Type: application/json'
```

For further information, refer to chapter *Knowledge Store REST API*, section *Repository Calls* of *Red Hat JBoss BPM Suite Development Guide*.

1.5. MANAGING ASSETS

NOTE

To activate and use the feature described below, login to Business Central with a user that has the **kiemgmt** role assigned.

To make management of projects easier, Red Hat JBoss BRMS now provides a way to manage multiple projects based on standards. This allows you to create repository structures using industry standard best practices for maintenance, versioning and distribution of your projects.

To start with, repositories can now be managed or unmanaged.

Managed and Unmanaged Repositories

Unmanaged Repositories are the repository structures that you are used to. They can contain multiple unrelated projects.

Managed Repositories, on the other hand, provide version control at the project level and project branches for managing the release cycle. Further, Managed Repositories can be restricted to just a single project or encompass multiple projects. When you create a Managed Repository, the asset management configuration process is automatically launched in order to create the repository branches. Corresponding project structure is created as well.

NOTE

Multi-project repositories must be managed.

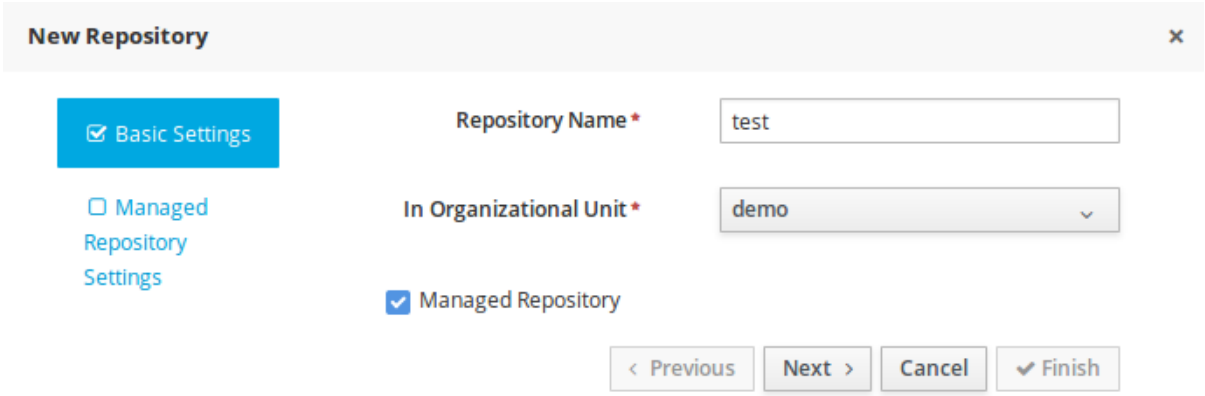
Procedure: Creating an Unmanaged Repository

1. In Business Central, go to **Authoring** → **Administration**.
2. Click **Repositories** → **New Repository**.
   The **New Repository** window is displayed.
3. Enter the repository name and select an organizational unit the repository belongs to.

4. Click **Finish**.

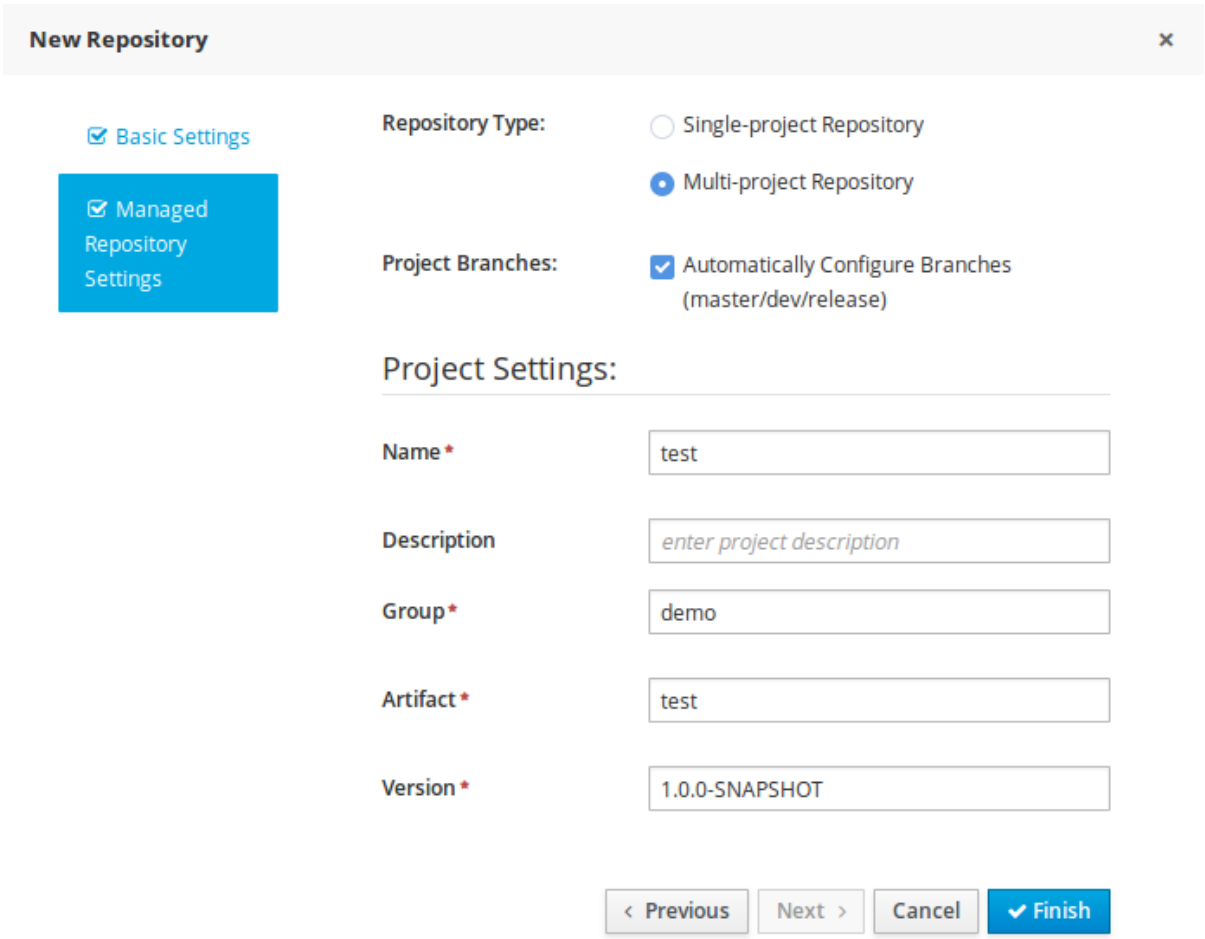
Procedure: Creating a Managed Repository

1. In Business Central, go to **Authoring** → **Administration**.
2. Click **Repositories** → **New Repository**.
   The **New Repository** window is displayed.
3. Enter the repository name and select an organizational unit the repository belongs to.
4. Select the **Managed Repository** check box and click **Next** to enter additional details of the Managed Repository.



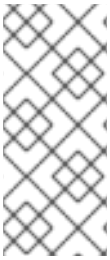
5. Choose either the **Single-project Repository** or the **Multi-project Repository** radio button.
   If the project you are creating is simple and self-contained, select the **Single-project Repository** radio button. Note that you will not be able to add more projects to this repository later.

   For more complex projects, where there is likely to be a parent project that encompasses smaller ones, select the **Multi-project Repository** radio button.



6. Enter the details of the managed project along with the GAV (Group, Artifact, Version) details. Note that all projects created in a **Multi-project Repository** will be managed together, with their version numbers being incremented together as well. Details of the parent project will be inherited by all future projects that you create in this Managed Repository.

7. Click **Finish** .



NOTE

If you create an unmanaged repository and add a **`pom.xml`** file (a parent Project Object Model (POM) file) directly under the repository directory, Red Hat JBoss BRMS converts the repository to a managed repository. For instructions about changing the repository status back to unmanaged, see the Red Hat Knowledgebase article An Unmanaged Repository Unexpectedly Turned Into "Managed" in BRMS / BPM Suite.

Managed Branches

With Managed Repositories comes the added advantage of Managed Branches. As in Git, you can choose to work on different branches of your project (for example: master, dev and release). This process of branching can also be automated for you, by selecting the checkbox while creating a new Managed Repository (for both single and multi-projects).

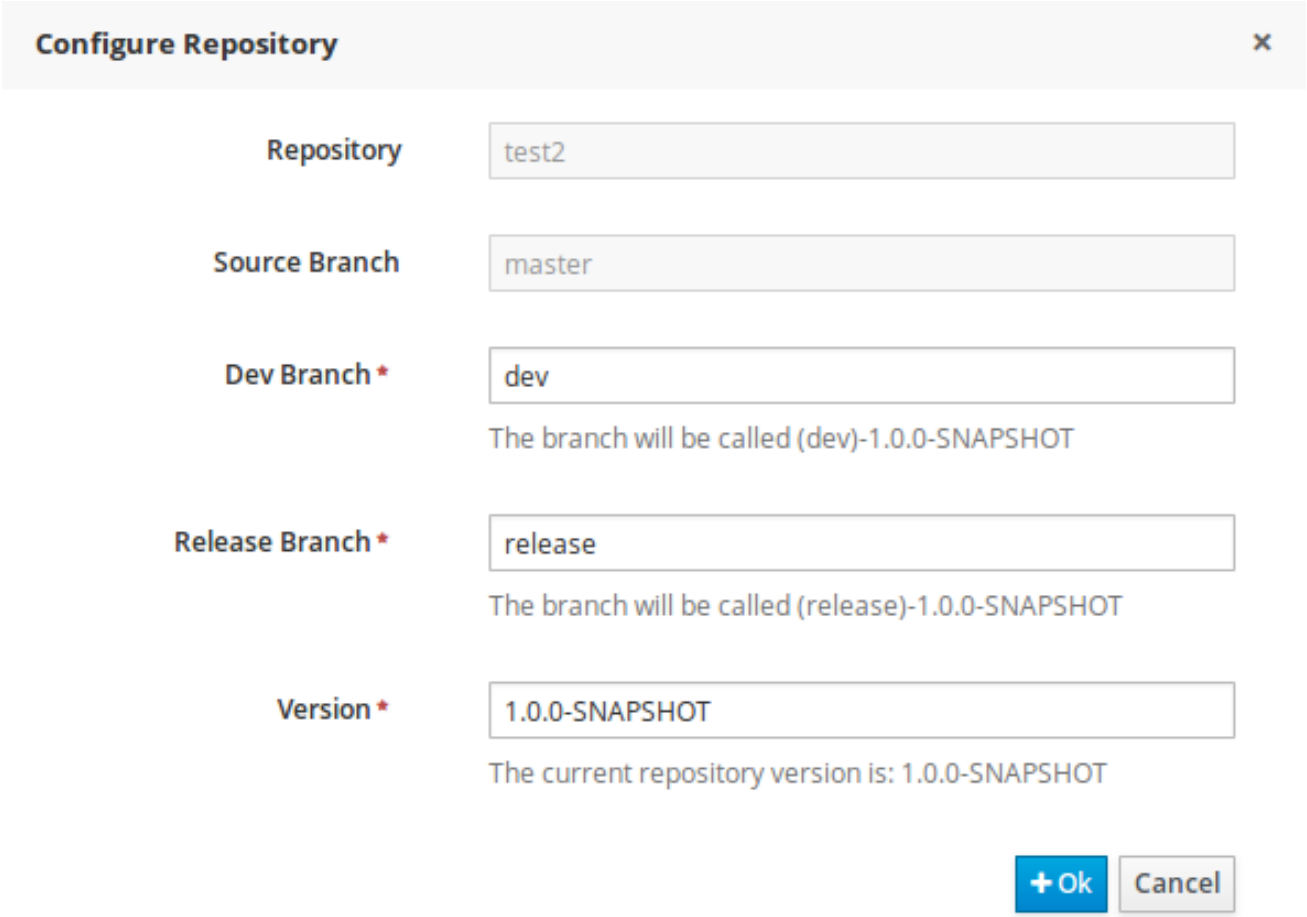
You can switch between branches by selecting the desired branch while working in the Project Explorer.

Repository Structure

If you do not select automatic branch management while creating a repository, you can create branches manually afterwards. For Managed Repositories, you can do so by using the **Configure** button. This button, along with **Promote** and **Release** buttons, is provided in the **Repository Structure** view. You

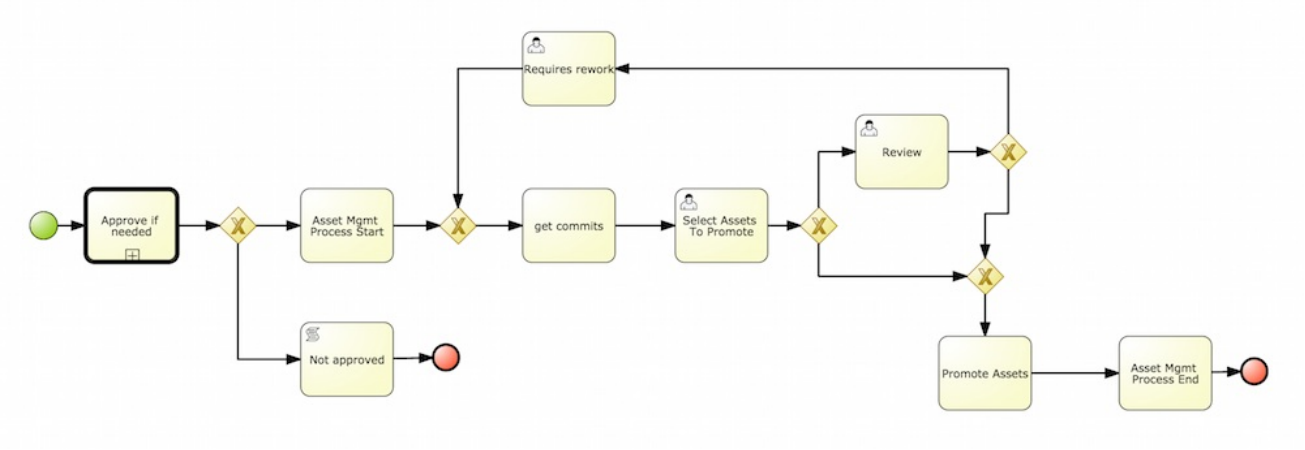
can access this view, by clicking on **Repository** → **Repository Structure** in the Project Explorer perspective menu.

Clicking on the **Configure** button allows you to create branches or edit automatically created ones.



You can promote assets from the master branch to other branches using the **Promote** button. Similarly, you can Release branches and deploy them on the server using the **Release** button.

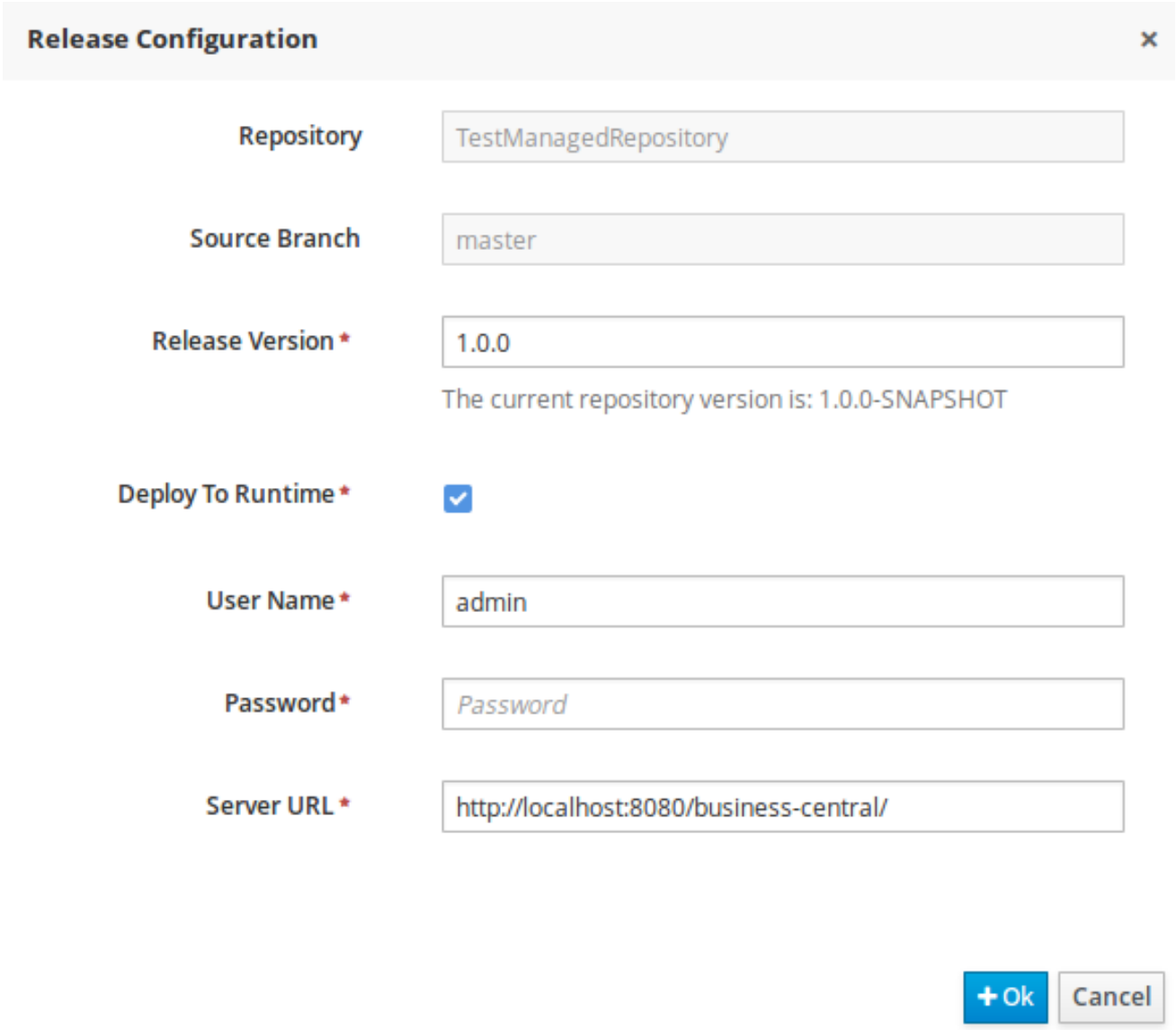
Both these functions are controlled internally by the use of pre-defined processes that are deployed on your instance. For example, when you click on **Promote** button after having done work on your development branch, a Promote Changes process is started in the background. A user, with the role of **`kiemgmt`** will have a user task appear in this task list to review the assets being promoted. This user can claim this task, and decide to promote all, some or none of the assets. The underlying process will cherry-pick the commits selected by the user to a release branch. This user can also request another review of these assets and this process can be repeated multiple times till all the assets are ready for release. The flow for this process is shown below:



Similarly, when you click on the **Release** button, a release process flow is initiated. This process flow builds the project and updates all the Maven artifacts to the next version, and deploys the project to the runtime, if runtime deployment details are supplied.

WARNING

Project branches to be released must start with the keyword `release`.



WARNING

Do not use **Deploy To Runtime** with Red Hat JBoss BRMS as it causes deploy failure. This function can only be used with Red Hat JBoss BPM Suite.

1.6. MAVEN REPOSITORY

Maven is a software project management tool which uses a project object model (POM) file to manage:

- Builds
- Documentation
- Reporting
- Dependencies
- Releases

- SCMs
- Distribution

A Maven repository is used to hold or store the build artifacts and project dependencies and is generally of two types:

Local

Refers to a local repository where all the project dependencies are stored and is located with the current installation in the default folder as "m2". It is a cache of the remote downloads, and also contains the temporary build artifacts which have not yet been released.

Remote

Refers to any other type of repository that can be accessed by a variety of protocols such as **`file://`** or **`http://`**. These repositories can be at a remote location set up by a third-party for downloading of artifacts or an internal repository set up on a file or HTTP server, used to share private artifacts between the development teams for managing internal releases.

1.7. CONFIGURING DEPLOYMENT TO A REMOTE NEXUS REPOSITORY

Nexus is a repository manager frequently used in organizations to centralize storage and management of software development artifacts. It is possible to configure your project so that artifacts produced by every build are automatically deployed to a repository on a remote Nexus server.

To configure your project to deploy artifacts to a remote Nexus repository, add a **`distributionManagement`** element to your project's **`pom.xml`** file as demonstrated in the code example below.

```
<distributionManagement>
  <repository>
    <id>deployment</id>
    <name>Internal Releases</name>

<url>http://your_nexus_host:8081/nexus/content/repositories/releases</url>
  </repository>
  <snapshotRepository>
    <id>deployment</id>
    <name>Internal Releases</name>

<url>http://your_nexus_host:8081/nexus/content/repositories/snapshots/</ur
l>
  </snapshotRepository>
</distributionManagement>
```

Replace the URLs in the example with real URLs of your Nexus repositories. The repository specified in the **`snapshotRepository`** element is used when the **`-SNAPSHOT`** qualifier is appended to the project's current version number. In other cases the repository specified in the **`repository`** element is used.

If your Nexus server requires authentication, you will also need to modify your projects Maven settings to add your credentials in the **`settings-security.xml`** file, using a master password. By default, this file is in the **`$M2_HOME/conf`** folder, unless you have changed its location by modifying the **`kie.maven.settings.custom`** system property. It is also possible to create new user-specific **`settings.xml`** and **`settings-security.xml`** files in the **`~/.m2`** folder. In that case, these files will override the original ones.

See the following example of the **settings-security.xml**:

```
<servers>
  <server>
    <id>deployment</id>
    <username>admin</username>
    <password>{COQLCE6DU6GtcS5P=}</password>
  </server>
</servers>
```

IMPORTANT

Note that keeping your server authentication credentials (for example the passwords) as a plain text in the **settings.xml** file is *not* recommended. All the information should be hashed with a master password in the **settings-security.xml** file.

For further information about password encryption and creating a master password, see the Apache Maven documentation, article Password Encryption.

With this configuration in place, clicking the **Build & Deploy** button in Business Central executes a Maven build and deploys the built artifacts both to the local repository and to one of the Nexus repositories specified in the **pom.xml** file.

CHAPTER 2. BUSINESS CENTRAL CONFIGURATION

All Business Central configuration settings are loaded from the ***EAP_HOME*/standalone/deployments/business-central.war/WEB-INF/web.xml** file. If you deploy Business Central on Red Hat JBoss EAP server, files **jboss-web.xml** and **jboss-deployment-structure.xml** contain configuration settings as well.

NOTE

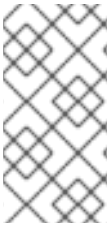
Business Central can be run on different platforms. For more information, see Red Hat JBoss BRMS Installation and Configuration Guide.

2.1. ACCESS CONTROL

Workbench Configuration

Within Red Hat JBoss BRMS, users may set up roles using LDAP to modify existing roles. Users may modify the roles in the workbench configuration to ensure the unique LDAP based roles conform to enterprise standards by editing the deployments directory located at ***JBOSS_HOME*/standalone/deployments/business-central.war/WEB-INF/classes/workbench-policy.propeties**.

If authenticating user via LDAP over Git, administrators must set system property **org.uberfire.domain** to the name of login module it should use to authenticate users via the Git service. This must be set in the **standalone.xml**file in EAP.



NOTE

You can further customize Business Central with parameters *no_build* or *no_search*. The parameters disable the build and search functionality. Include one or both parameters in the Business Central URL, for example **http://*SERVER*:PORT/business-central/kie-wb.jsp?no_build&no_search**.

2.2. BRANDING BUSINESS CENTRAL APPLICATION

The Business Central web application can be customized by overriding some of its default styles. The personalized Business Central branding allows you to get a consistent appearance across all your applications, while it is also possible to create a different user interfaces for each team within your company. The customizable elements are built by using HTML files and images, which enables an easy and flexible customization of the application without having to recompile the code.

The following Business Central application elements can be customized:

- In the login screen, the foreground corner images, company logo, and project logo can be changed.
- The upper application banner displayed after logging in can be personalized.
- In help pop-up windows, the label text and splash help images can be customized.

2.2.1. Customizing Business Central Login Page

Procedure: Changing Foreground Corner Images

1. Start the EAP server and open Business Central in a web browser.
2. To change the upper right corner foreground image, copy the substitute PNG file named **bg-login.png** to the ***EAP_HOME*/standalone/deployments/business-central.war/img/** directory in your Red Hat JBoss BRMS installation.
3. To change the lower right corner foreground image, copy the substitute PNG file named **bg-login-2.png** to the ***EAP_HOME*/standalone/deployments/business-central.war/img/** directory in your Red Hat JBoss BRMS installation.
4. Force a full reload of the login page, bypassing the cache, to view the changes. For example, in most Linux and Windows web browsers, press **Ctrl**+**F5**.

Procedure: Changing Company Logo and Project Logo

1. Start the EAP server and open Business Central in a web browser.
2. Navigate to the ***EAP_HOME*/standalone/deployments/business-central.war/img/** directory in your Red Hat JBoss BRMS installation.
3. To change the company logo that appears at the upper right hand corner of the login page, replace the default image **login-screen-logo.png** with a new image in the PNG format.
4. To change the project logo that appears above the login text fields, replace the default image **RH_JBoss_BRMS_Logo.svg** with a new SVG file.
5. Force a full reload of the login page, bypassing the cache, to view the changes.

2.2.2. Customizing Business Central Application Header

1. Start the EAP server, open Business Central in a web browser, and log in with your user credentials.
2. Copy your new application header image in the SVG format to the ***EAP_HOME*/standalone/deployments/business-central.war/banner/** directory in your Red Hat JBoss BRMS installation.
3. Open ***EAP_HOME*/standalone/deployments/business-central.war/banner/banner.html** file in a text editor.
4. In the **banner.html** file, edit the following **<img>** tag to provide the name of your new header image:

```
<img src="banner/logo.svg"/>
```

5. Force a full reload of the page, bypassing the cache, to view the changes.

2.2.3. Customizing Business Central Splash Help Windows

Each splash page and its corresponding HTML file are located in the ***EAP_HOME*/standalone/deployments/business-central.war/plugins/** directory. The files contain information about the images and the text to be displayed. For example, the **authoring_perspective.splash.js** splash page points to the

authoring_perspective.splash.html file, which contains the names, captions, and location of all the image files that appear in the splash help pop-up windows of the Business Central Authoring perspective.

Procedure: Changing Splash Help Images and Captions

1. Start the EAP server, open Business Central in a web browser, and log in with your user credentials.
2. Copy the new splash help images to the ***EAP_HOME*/standalone/deployments/business-central.war/images/** directory in your Red Hat JBoss BRMS installation.
3. Open the corresponding HTML file from the **plugins** directory in a text editor.
4. Edit the HTML file to point to your new splash help image. For example, to change the first image that appears in the Authoring perspective splash help, edit the following **<img>** tag in the **authoring_perspective.splash.html** file to add your new image:

```
<img src="images/authoring_perspective1.png" alt="">
```

5. To change the image caption that appears on the splash help, edit the **<h4>** and **<p>** tag contents in the ** ** tag below the correspoding **<img>** tag:

```
<div class="carousel-caption">
  <h4>Authoring</h4>

  <p>Modularized and customizable workbench</p>
</div>
```

6. Force a full reload, bypassing the cache, and access the splash help pop-up windows to view the changes.

2.3. EXTENDING BUSINESS CENTRAL

Starting with version 6.1 of Red Hat JBoss BRMS, Business Central can be configured to add new screens, menus, editors, splashscreens and perspectives by the Administrator. These elements can extend functionality of Business Central and can be accessed through the **Extensions → Plugin Management**.

You can now define your own Javascript and HTML based plugins to extend Business Central and add them without having to worry about copying files in the underlying filesystem. Let us add a new screen in the system to show you the basics of this functionality.

2.3.1. Plugin Management

You access the **Plugin Management** screen by clicking on **Extensions → Plugin Management**. This brings up the *Plugin Explorer* screen that lists all the existing plugins under their respective categories:

- *Perspective Plugin*
- *Screen Plugin*
- *Editor Plugin*

- *Splashscreen Plugin*
- and *Dynamic Menu*

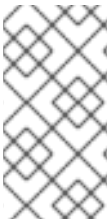
Open any of these, and you will see the existing plugins in each category, including the uneditable system generated ones.

Let us create a new plugin that echoes "Hello World" when users visit the screen for that plugin. In general, the steps to creating a new plugin are:

1. Create a new screen
2. Create a new perspective (and add the new screen to it)
3. Create a new menu (and add the new perspective to it)
4. Apps (optional)

Adding a New Screen

Click the  button and select **New Screen**. You will be prompted to enter the name of this new screen. Enter "HelloWorldJS" and press the **OK** button. The Screen plugin editor will open, divided into 4 sections: **Template**, **CSS**, **JavaScript** and **Media**.



NOTE

All manually created elements go into their respective categories in case you want to edit them later. In this case, to open the Screen plugin editor again if you close it, open the **Screen Plugin** category and scroll past the system generated screens to your manually created plugin and click on it to open the Screen plugin editor again.

Template is where your HTML goes, CSS is for styling, JavaScript is for your functions and Media is for uploading and managing images.

Since we are making a simple Hello World plugin, enter the following code in the Template section: **`<div>My Hello World Screen</div>`**. This can be any HTML code, and you can use the supplied **`Angular`** and **`Knockout`** frameworks. For the purposes of this example, we are not using any of those frameworks, but you can choose to by selecting them from the drop down in the Template section.

Enter your JavaScript code in the JavaScript section. Some common methods and properties are defined for you, including **`main`**, **`on_close`** and **`on_open`**. For this demo, select the **`on_open`** and enter the following: **`function () { alert('Hello World'); }`**

Click the **Save** button to finish creating the screen. After you save the screen, refresh business central so that the Screen Plugin is listed in the Screen Component of Perspective plugin.

Adding New Perspective

Once a screen has been created, you need to create a perspective on which this screen will reside. Perspectives can also be created similar to the way a screen is created by clicking on the New button and then selecting **New Perspective**. You can now provide a name for this perspective, say **`HelloWorldPerspective`**. This will open the Perspective plugin editor, similar to the Screen plugin editor.

The Perspective Editor is like a drag and drop grid builder for screens and HTML components. Remove any existing grids and then drag a 6×6 grid on the right hand side to the left hand side.

Next, open the **Components** category and drag a Screen Component on the right hand side to the left hand side (in any grid). This will open the **Edit Component** dialog box that allows you to select the screen created in the previous step (**`HelloWorldJS`**). Click the **OK** button and then click **Save** to save this perspective. To tag your perspective, enter **`Home`** in the tag name field and click **Tags**. Click **OK** and save the changes.

You can open this perspective again from the Perspective plugins listed on the left hand side.

Adding New Menu
The final step in creating our plugin is to add a dynamic menu from where the new screen/perspective can be called up. To do so, go to **Extensions → Plugin Management** and then click on the *New* button to select *New Dynamic Menu*. Give this dynamic menu a name (**`HelloWorldMenu`**) and then click the **OK** button. The dynamic menu editor opens up.

Enter the perspective name (**`HelloWorldPerspective`**) as the **Activity Id** and the name for the drop down menu (**`HelloWorldMenuDropDown`**). Click **OK** and then **Save**.

This new menu will be added to your workbench the next time you refresh Business Central. Refresh it now to see **HelloWorldMenu** added to your top level menu. Click on it to reveal **HelloWorldMenuDropDown**, which when clicked will open your perspective/screen with the message **`Hello World`**.

You have created your first Plugin!

Working with Apps (Optional)
If you create multiple plugins, you can use the Apps directory feature to organize your own components and plugins, instead of having to rely on just the top menu entries.

When you save a new perspective, you can add labels (tags) for them and these labels (tags) are used to associate a perspective with an App directory. You can open the App directories by clicking on **Extensions → Apps**.

The Apps directory provides an alternate way to open your perspective. When you created your **`HelloWorldPerspective`**, you entered the tag **`Home`**. The Apps directory by default contains a single directory called **`Home`** with which you associated your perspective. This is where you will find it when you open the Apps directory. You can click on it to run the perspective now.

You can create multiple directories and associate perspectives with those directories depending on functional and vertical business requirements. For example, you could create an HR directory and then associate all HR related perspectives with that directory to better manage Apps.

You can create a new directory by clicking the  button.

2.3.2. The JavaScript (JS) API for Extensions

The extensibility of Business Central is achieved by an underlying JavaScript (JS) API which is automatically loaded if it is placed in the **`plugins`** folder of the Business Central webapp (typically: ***`INSTALL_DIR`*`/business-central.war/plugins/`**), or it can be loaded via regular JavaScript calls.

This API is divided into multiple sets depending on the functionality it performs.

Register Perspective API

Allows for the dynamic creation of perspectives. The example below creates a panel using the **`registerPerspective`** method:

```
$registerPerspective({
    id: "Home",
    is_default: true,
    panel_type:
"org.uberfire.client.workbench.panels.impl.MultiListWorkbenchPanelPresen
ter",
    view: {
        parts: [
            {
                place: "welcome",
                min_height: 100,
                parameters: {}
            }
        ],
        panels: [
            {
                width: 250,
                min_width: 200,
                position: "west",
                panel_type:
"org.uberfire.client.workbench.panels.impl.MultiListWorkbenchPanelPresen
ter",
                parts: [
                    {
                        place: "YouTubeVideos",
                        parameters: {}
                    }
                ]
            },
            {
                position: "east",
                panel_type:
"org.uberfire.client.workbench.panels.impl.MultiListWorkbenchPanelPresen
ter",
                parts: [
                    {
                        place: "TodoListScreen",
                        parameters: {}
                    }
                ]
            },
            {
                height: 400,
                position: "south",
                panel_type:
"org.uberfire.client.workbench.panels.impl.MultiTabWorkbenchPanelPresent
er",
                parts: [
                    {
                        place: "YouTubeScreen",
                        parameters: {}
                    }
                ]
            }
```

```
        ]
    }
});
```

Editor API

Allows you to dynamically create editors and associate them with a file type. The example below creates a sample editor and associates it with **`filename`** file type.

```
$registerEditor({
    "id": "sample editor",
    "type": "editor",
    "templateUrl": "editor.html",
    "resourceType":
"org.uberfire.client.workbench.type.AnyResourceType",
    "on_concurrent_update":function(){
        alert('on_concurrent_update callback')

$vfs_readAllString(document.getElementById('filename').innerHTML,
function(a) {
            document.getElementById('editor').value= a;
        });
    },
    "on_startup": function (uri) {
        $vfs_readAllString(uri, function(a) {
            alert('sample on_startup callback')
        });
    },
    "on_open":function(uri){
        $vfs_readAllString(uri, function(a) {
            document.getElementById('editor').value=a;
        });
        document.getElementById('filename').innerHTML = uri;
    }
});
```

In addition to **`on_startup`** and **`on_open`** methods seen in the previous example, the API exposes the following callback events for managing the editor's lifecycle:

- **`on_concurrent_update;`**
- **`on_concurrent_delete;`**
- **`on_concurrent_rename;`**
- **`on_concurrent_copy;`**
- **`on_rename;`**
- **`on_delete;`**
- **`on_copy;`**
- **`on_update;`**
- **`on_open;`**

- **on_close;**
- **on_focus;**
- **on_lost_focus;**
- **on_may_close;**
- **on_startup;**
- **on_shutdown;**

You can display this editor via an HTML template:

```
<div id="sampleEditor">
    <p>Sample JS editor (generated by editor-sample.js)</p>
    <textarea id="editor"></textarea>

    <p>Current file:</p><span id="filename"></span>
    <button id="save" type="button"
onclick="$vfs_write(document.getElementById('filename').innerHTML,
document.getElementById('editor').value,  function(a)
{});">Save</button>
    <br>

    <p>This button change the file content, and uberfire send a callback
to the editor:</p>
    <button id="reset" type="button"
onclick="$vfs_write(document.getElementById('filename').innerHTML,
'Something else',  function(a) {});">Reset File</button>
</div>
```

PlaceManager API

The methods of this API allow you to request that the Business Central display a particular component associated with a target: **`$goToPlace("componentIdentifier");`**

Register plugin API

The methods of this API allow you to create dynamic plugins (that will be transformed in Business Central screens) via the JS API.

```
$registerPlugin( {
    id: "my_angular_js",
    type: "angularjs",
    templateUrl: "angular.sample.html",
    title: function () {
        return "angular " + Math.floor(Math.random() * 10);
    },
    on_close: function () {
        alert("this is a pure JS alert!");
    }
});
```

The plugin references the **`angular.sample.html`** template:

```
<div ng-controller="TodoCtrl">
```

```
    <span>{{remaining()}} of {{todos.length}} remaining</span>
    [ <a href="" ng-click="archive()">archive</a> ]
    <ul class="unstyled">
        <li ng-repeat="todo in todos">
            <input type="checkbox" ng-model="todo.done">
            <span class="done-{{todo.done}}">{{todo.text}}</span>
        </li>
    </ul>
    <form ng-submit="addTodo()">
        <input type="text" ng-model="todoText" size="30"
placeholder="add new todo here">
        <input class="btn-primary" type="submit" value="add">
    </form>
    <form ng-submit="goto()">
        <input type="text" ng-model="placeText" size="30"
placeholder="place to go">
        <input class="btn-primary" type="submit" value="goTo">
    </form>
</div>
```

A plugin can be hooked to Business Central events via a series of JavaScript callbacks:

- **on_concurrent_update;**
- **on_concurrent_delete;**
- **on_concurrent_rename;**
- **on_concurrent_copy;**
- **on_rename;**
- **on_delete;**
- **on_copy;**
- **on_update;**
- **on_open;**
- **on_close;**
- **on_focus;**
- **on_lost_focus;**
- **on_may_close;**
- **on_startup;**
- **on_shutdown;**

Register splash screens API

use the methods in this API to create splash screens.

```
$registerSplashScreen({
    id: "home.splash",
    templateUrl: "home.splash.html",
    body_height: 325,
    title: function () {
        return "Cool Home Splash " + Math.floor(Math.random() * 10);
    },
    display_next_time: true,
    interception_points: ["Home"]
});
```

Virtual File System (VFS) API

with this API, you can read and write a file saved in the file system using an asynchronous call.

```
$vfs_readAllString(uri,  function(a) {
  //callback logic
});

$vfs_write(uri,content,  function(a) {
  //callback logic
})
```

2.4. CONFIGURING TABLE COLUMNS

Business Central allows you to configure views that contain lists of items in the form of tables. You can resize columns, move columns, add or remove the default list of columns and sort the columns. This functionality is provided for all views that contain tables.

Once you make changes to the columns of a table view, these changes are persisted for the current logged in user.

Adding and Removing Columns

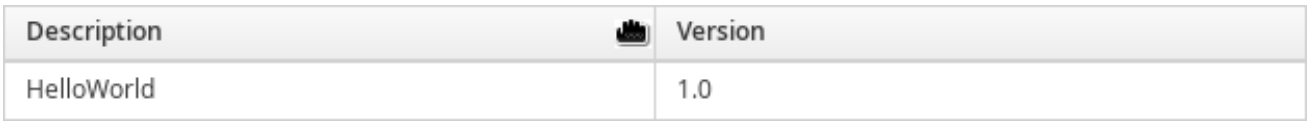
Tables that allow columns to be configured have the  button in the top right corner. Clicking on this button opens up the list of columns that can added or removed to the current table with a check box next to each column:

Resizing Columns

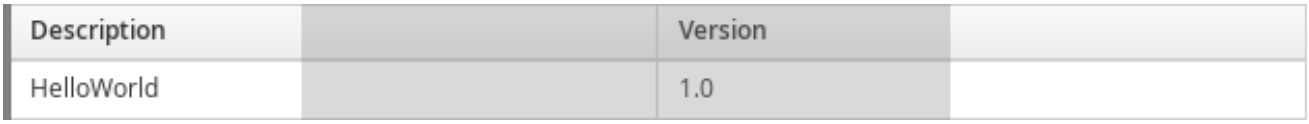
To resize columns, place your cursor between the edges of the column header and move in the direction that you want:

Moving Columns

To re-order and drag and drop a column in a different position, hover your mouse over the rightmost area of the column header:



You can now grab the column and move it:



Drop it over the column header that you want to move it to.

Sorting Columns

To sort columns, click on the desired column’s header. To reverse-sort, click on the header again.

CHAPTER 3. REALTIME DECISION SERVER

The Realtime Decision Server is a standalone, built-in component that can be used to instantiate and execute rules through interfaces available for REST, JMS, or a Java client side application. Created as a web deployable WAR file, this server can be deployed on any web container. The current version of the Realtime Decision Server is included with default extensions for both Red Hat JBoss BRMS and Red Hat JBoss BPM Suite.

This server has a low footprint with minimal memory consumption and therefore can be deployed easily on a cloud instance. Each instance of this server can open and instantiate multiple KIE containers which allows you to execute multiple rule services in parallel.

You can provision the Realtime Decision Server instances through Business Central. In this chapter, the steps required to set up the Realtime Decision Server, provision and connect to the server through Business Central, control what artifacts go in each instance, and go through its lifecycle are provided.

NOTE

For more information, see the Intelligent Process Server and Realtime Decision Server chapter of the *Red Hat JBoss BPM Suite Development Guide*.

3.1. DEPLOYING REALTIME DECISION SERVER

The Realtime Decision Server is distributed as a web application archive (WAR) file **`kie-server.war`**. When you install Red Hat JBoss BRMS, the WAR file is installed and deployed in your web container by default.

You can copy the WAR file and deploy it in any other web container, such as Red Hat JBoss Web Server or another Red Hat JBoss EAP install. Note that the WAR file must be compatible with the container.

1. Once you have deployed the WAR file, create a user with the role **`kie-server`** in the web container.

NOTE

You can change the **`kie-server`** user role in the **`web.xml`** file or in Keycloak.

1. Verify that you can access the decision engine: open **`http://SERVER:PORT/kie-server/services/rest/server/`** in a web browser and enter the user name and the password specified in the previous step.

2. Once authenticated, an XML response in the form of engine status opens:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response type="SUCCESS" msg="Kie Server info">
  <kie-server-info>
    <capabilities>BPM</capabilities>
    <capabilities>KieServer</capabilities>
    <capabilities>BRM</capabilities>
    <location>http://localhost:8230/kie-
server/services/rest/server</location>
    <name>KieServer@/kie-server</name>
    <id>15ad5bfa-7532-3eea-940a-abbbbc89f1e8</id>
```

```
    <version>6.4.0.Final-redhat-5</version>
  </kie-server-info>
</response>
```

3.2. INSTALLING REALTIME DECISION SERVER IN OTHER CONTAINERS

3.2.1. Red Hat JBoss Web Server 2.*X*/3.*X*, Tomcat 8.*X*/9.*X*

Use the following procedure to install the Realtime Decision Server in a Tomcat container.

1. Follow the steps described in section Installing Red Hat JBoss BRMS on Red Hat JBoss Web Server of *Red Hat JBoss BRMS Installation Guide* to download and extract the generic deployable archive containing **kie-server.war**.

2. Configure users and roles. Make sure that the ***TOMCAT_HOME*/conf/tomcat-users.xml** file contains the following role and user definition. The user name and password should be unique, however, the role must be defined as **kie-server**:

```
<role rolename="kie-server"/>
<user username="serveruser" password="my.s3cr3t.pass" roles="kie-
server"/>
```

3. Start the server: run ***TOMCAT_HOME*/bin/startup.sh** or in a Windows system ***TOMCAT_HOME*/bin/startup.bat**. To make sure that the application was deployed successfully, see Tomcat logs in the ***TOMCAT_HOME*/logs** directory. See Section 3.3.1, “Bootstrap Switches” for the list of bootstrap switches that can be used to properly configure the instance. For example:

```
$ ./startup.sh -Dorg.kie.server.id=first-kie-server -
Dorg.kie.server.location=http://localhost:8080/kie-
server/services/rest/server
```

4. To verify that the server is running, open **http://*SERVER*:*PORT*/kie-server/services/rest/server/** in a web browser and enter the user name and the password specified before. You should see a simple XML message with basic information about the server.

IMPORTANT

It is not possible to use the JMS interface if the Realtime Decision Server was installed on Tomcat or any other web container. The web container version of the WAR file contains only the REST interface.

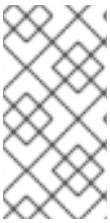
3.3. REALTIME DECISION SERVER SETUP

3.3.1. Bootstrap Switches

The Realtime Decision Server accepts a number of bootstrap switches (system properties) to configure the behavior of the server.

Table 3.1. Bootstrap Switches for Disabling Realtime Decision Server Extensions

Property	Values	Default	Description
`org.drools.server.ext.disabled`	`true`, `false`	`false`	If set to `true`, disables the BRM support (for example rules support).
`org.jbpm.server.ext.disabled`	`true`, `false`	`false`	If set to `true`, disables the BPM support (for example processes support).
`org.optaplanner.server.ext.disabled`	`true`, `false`	`false`	If set to `true`, disables the BRP support (for example planner support).
`org.jbpm.ui.server.ext.disabled`	`true`, `false`	`false`	If set to `true`, disables the Realtime Decision Server UI extension.
`org.kie.executor.disabled`	`true`, `false`	`false`	Disables the Red Hat JBoss BRMS executor.



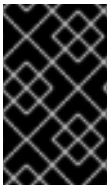
NOTE

Some controller properties listed below are marked as required. Set these properties when you handle Realtime Decision Server container creation and removal in Business Central. If you use the Realtime Decision Server separately without any interaction with Business Central, the properties do not have to be set.

Table 3.2. Bootstrap Switches Required for Using a Controller

Property	Values	Default	Description
`org.kie.server.id`	String	N/A	An arbitrary ID to be assigned to the server. If a remote controller is configured, this is the ID under which the server will connect to the controller to fetch the KIE container configurations. If not provided, the ID is automatically generated.
`org.kie.server.user`	String	`kieserver`	The user name used to connect with the Realtime Decision Server from the controller, required when running in managed mode. Set this property in Business Central system properties. Setting this property is required when using a controller.
`org.kie.server.pwd`	String	`kieserver1!`	The password used to connect with the Realtime Decision Server from the controller, required when running in managed mode. Set this property in Business Central system properties. Setting this property is required when using a controller.

Property	Values	Default	Description
org.kie.server.token	String	N/A	This property enables you to use a token-based authentication between the controller and the Realtime Decision Server instead of the basic user name/password authentication. The controller sends the token as a parameter in the request header. Note that long-lived access tokens are required as the tokens are not refreshed.
org.kie.server.location	URL	N/A	The URL of the Realtime Decision Server instance used by the controller to call back on this server, for example: **http://localhost:8230/kie-server/services/rest/server**. Setting this property is required when using a controller.
org.kie.server.controller	Comma-separated list	N/A	A comma-separated list of URLs to the controller REST endpoints, for example **http://localhost:8080/business-central/rest/controller**. Setting this property is required when using a controller.
org.kie.server.controller.user	String	**kieserver**	The user name to connect to the controller REST API. Setting this property is required when using a controller.
org.kie.server.controller.pwd	String	**kieserver1!**	The password to connect to the controller REST API. Setting this property is required when using a controller.
org.kie.server.controller.token	String	N/A	This property enables you to use a token-based authentication between the Realtime Decision Server and the controller instead of the basic user name/password authentication. The server sends the token as a parameter in the request header. Note that long-lived access tokens are required as the tokens are not refreshed.
org.kie.server.controller.connect	Long	**10000**	The waiting time in milliseconds between repeated attempts to connect the Realtime Decision Server to the controller when the server starts.



IMPORTANT

Make sure the new data source for the Realtime Decision Server points to a different database schema than the data source used by Business Central by modifying the **`org.kie.server.persistence.ds`** property in order to avoid conflicts.

Table 3.3. Bootstrap Switches for Executor Properties

Property	Values	Default	Description
`org.kie.executor.interval`	Integer	**`3`**	The time between the moment the Red Hat JBoss BRMS executor finishes a job and the moment it starts a new one, in a time unit specified in the **`org.kie.executor.timeunit`** property.
`org.kie.executor.timeunit`	**`java.util.concurrent.TimeUnit`** constant	**`SECONDS`**	The time unit in which the **`org.kie.executor.interval`** property is specified.
`org.kie.executor.pool.size`	Integer	**`1`**	The number of threads used by the Red Hat JBoss BRMS executor.
`org.kie.executor.retry.count`	Integer	**`3`**	The number of retries the Red Hat JBoss BRMS executor attempts on a failed job.

Table 3.4. Other Bootstrap Switches

Property	Values	Default	Description
`kie.maven.settings.custom`	Path	N/A	The location of a custom **`settings.xml`** file for Maven configuration.
`kie.server.jms.queues.response`	String	**`queue/KIE.SERVER.RESPONSE`**	The response queue JNDI name for JMS.
`org.drools.server.filter.classes`	**`true`**, **`false`**	**`false`**	When set to **`true`**, the Drools Realtime Decision Server extension accepts custom classes annotated by the **`XmlRootElement`** or **`Remotable`** annotations only.
`org.kie.server.domain`	String	N/A	The JAAS **`LoginContext`** domain used to authenticate users when using JMS.

Property	Values	Default	Description
`org.kie.server.repo`	Path	`.`	The location where Realtime Decision Server state files will be stored.
`org.kie.server.sync.deploy`	`true`, `false`	`false`	Instructs the Realtime Decision Server to hold the deployment until the controller provides the containers deployment configuration. This property affects only the servers running in managed mode. The options are as follows: • `false`; the connection to the controller is asynchronous. The application starts, connects to the controller, and once successful, deploys the containers. The application accepts requests even before the containers are available. • `true`; the deployment of the server application joins the controller connection thread with the main deployment and awaits its completion. This option can lead to a potential deadlock in case more applications are on the same server instance. It is strongly recommended to use only one application (the server) on one server instance.

3.3.2. Managed Realtime Decision Server

A managed instance requires an available controller to start the Realtime Decision Server.

A controller manages the Realtime Decision Server configuration in a centralized way. Each controller can manage multiple configurations at once, and there can be multiple controllers in the environment. Managed Realtime Decision Server can be configured with a list of controllers, but will only connect to one at a time.

IMPORTANT

Controllers should be synchronized to ensure that the same set of configuration is provided to the server, regardless of the controller to which it connects.

When the Realtime Decision Server is configured with a list of controllers, it will attempt to connect to each of them at startup until a connection is successfully established with one of them. If a connection cannot be established, the server will not start, even if there is a local storage available with configuration. This ensures consistence and prevents the server from running with redundant configuration.

NOTE

To run the Realtime Decision Server in standalone mode without connecting to controllers, see Section 3.3.3, “Unmanaged Realtime Decision Server”.

Configuring Realtime Decision Server Managed by Business Central

WARNING

This section provides a sample setup that you can use for testing purposes. Some of the values are unsuitable for a production environment, and are marked as such.

Configure the Business Central to manage a Realtime Decision Server instance by performing the following steps:

Configuring Realtime Decision Server Managed by Business Central

1. Make sure users with the following roles exist:
   - In Business Central, a user with the role **`rest-all`**.
   - On the Realtime Decision Server, a user with the role **`kie-server`**.

   NOTE

   In production environments, use two distinct users, each with one role. In this sample situation, we use only one user named **`controllerUser`** that has both the **`rest-all`** and the **`kie-server`** roles.

   If such users do not exist, create them.

   - On Red Hat JBoss EAP, go to ***`EAP_HOME`*`/bin/`** and execute:

     ```
     $ ./add-user.sh -a --user controllerUser --password
     controllerUser1234; --role kie-server,rest-all
     ```

     WARNING

     Plain-text passwords are not secure. For production environments, use a password vault. For more information, see the Password Vault chapter of the *Red Hat JBoss Enterprise Application Platform 6.4 How To Configure Server Security* guide.

- On Red Hat JBoss Web Server, see Section 3.2.1, "Red Hat JBoss Web Server 2.*X*/3.*X*, Tomcat 8.*X*/9.*X*".
- On IBM WebSphere Application Server, see the Creating Users and Groups section of the *Red Hat JBoss BRMS IBM WebSphere Installation and Configuration Guide*.
- On Oracle WebLogic Server, see the Configuring Security Settings section of the *Red Hat JBoss BRMS Oracle Weblogic Installation and Configuration Guide*.

2. Set the following JVM properties. The location of Business Central and the Realtime Decision Server may be different. In such case, ensure you set the properties on the correct server instances.
   - On Red Hat JBoss EAP, modify the **`<system-properties>`** section in:
     - ***`EAP_HOME`*`/standalone/configuration/standalone*.xml`** for standalone mode.
     - ***`EAP_HOME`*`/domain/configuration/domain.xml`** for domain mode.
   - On Red Hat JBoss Web Server, see Section 3.2.1, "Red Hat JBoss Web Server 2.*X*/3.*X*, Tomcat 8.*X*/9.*X*".
   - On IBM WebSphere Application Server, see section Adding Custom JVM Properties of the *Red Hat JBoss BRMS IBM WebSphere Installation and Configuration Guide*.
   - On Oracle WebLogic Server, see section Setting Environment Variables of the *Red Hat JBoss BRMS Oracle WebLogic Installation and Configuration Guide*.

Table 3.5. JVM Properties for Managed Realtime Decision Server Instance

Property	Value	Note
`org.kie.server.id`	`default-kie-server`	The Realtime Decision Server ID.
`org.kie.server.controller`	`http://localhost:8080/business-central/rest/controller`	The location of Business Central.
`org.kie.server.controller.user`	`controllerUser`	The user name with the role `rest-all` as mentioned in the previous step.
`org.kie.server.controller.pwd`	`controllerUser1234;`	The password of the user mentioned in the previous step.
`org.kie.server.location`	`http://localhost:8080/kie-server/services/rest/server`	The location of the Realtime Decision Server.

Table 3.6. JVM Properties for Business Central Instance

Property	Value	Note
org.kie.server.user	**controllerUser**	The user name with the role **kie-server** as mentioned in the previous step.
org.kie.server.pwd	**controllerUser1234;**	The password of the user mentioned in the previous step.

3. Verify the successful start of the Realtime Decision Server by sending a GET request to **http://*SERVER*:*PORT*/kie-server/services/rest/server/**. Once authenticated, you get an XML response similar to this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response type="SUCCESS" msg="Kie Server info">
    <kie-server-info>
        <capabilities>BRM</capabilities>
        <capabilities>BPM-UI</capabilities>
        <capabilities>BPM</capabilities>
        <capabilities>BRP</capabilities>
        <capabilities>KieServer</capabilities>
        <location>http://localhost:8080/kie-
server/services/rest/server</location>
        <messages>
            <content>Server KieServerInfo{serverId='local-server-
123', version='6.4.0.Final-redhat-3',
location='http://localhost:8080/kie-
server/services/rest/server'}started successfully at Fri Jun 03
13:48:44 CEST 2016</content>
            <severity>INFO</severity>
            <timestamp>2016-06-03T13:48:44.606+02:00</timestamp>
        </messages>
        <name>local-server-123</name>
        <id>local-server-123</id>
        <version>6.4.0.Final-redhat-3</version>
    </kie-server-info>
</response>
```

4. Verify successful registration by logging into Business Central and selecting **Deploy** → **Execution Servers**. If successful, you can see the registered server ID.

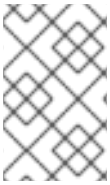
3.3.3. Unmanaged Realtime Decision Server

An unmanaged Realtime Decision Server is a standalone instance, and therefore must be configured individually using REST/JMS API from the Realtime Decision Server itself. There is no controller involved. The configuration is automatically persisted by the server into a file and that is used as the internal server state, in case of restarts.

The configuration is updated during the following operations:

- Deploy KIE Container

- Undeploy KIE Container
- Start KIE Container
- Stop KIE Container



NOTE

If the Realtime Decision Server is restarted, it will attempt to re-establish the same state that was persisted before shutdown. Therefore, KIE Containers that were running will be started, but the ones that were stopped will not.

3.4. CREATING CONTAINERS

Once the Realtime Decision Server is registered, you can start adding containers. Containers are self-contained environments that have been provisioned to hold instances of your packaged and deployed rule instances.

To create a container:

1. Log in to Business Central.
2. In the main menu on the top, click **Deploy** → **Execution Servers**.
3. Select your server from the **SERVER TEMPLATES** section on the left side of the page.
4. Click **Add Container** in the **KIE CONTAINERS** section.
   The **New Container** dialog window opens.
5. Enter a name of your container and search for the project you want to deploy in the container.
   Click **Select** next to the project to automatically enter the project's details.
   Alternatively, you can enter **Group Name**, **Artifact Id**, and **Version** manually.

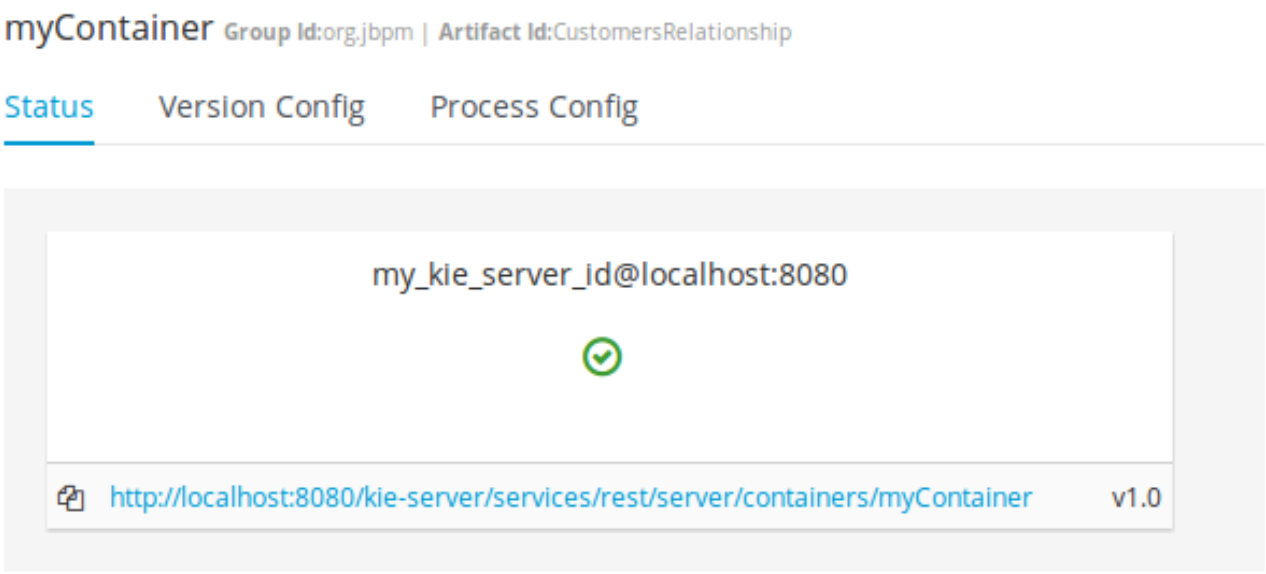
WARNING

When entering the container's version number, do *not* use the **LATEST** or **RELEASE** keywords. This feature has been deprecated and can cause deployment issues.

6. Click **Next** to configure the runtime strategy, **`KieBase`**, **`KieSession`**, and merge mode for your container. You can skip this step.
7. Click **Finish**.

After the container is successfully created, click **Start** at the upper right hand corner to start it.

Figure 3.1. Container in Started Mode



To verify that the container is running, send a [GET] request to the endpoint.

Example 3.1. Server Response

```
<response type="SUCCESS" msg="Info for container myContainer">
  <kie-container container-id="myContainer" status="STARTED">
    <messages>
      <content>Container myContainer successfully created with module
org.jbpm:CustomersRelationship:1.0.</content>
      <severity>INFO</severity>
      <timestamp>2016-03-02T11:43:40.806+01:00</timestamp>
    </messages>
    <release-id>
      <artifact-id>CustomersRelationship</artifact-id>
      <group-id>org.jbpm</group-id>
      <version>1.0</version>
    </release-id>
    <resolved-release-id>
      <artifact-id>CustomersRelationship</artifact-id>
      <group-id>org.jbpm</group-id>
      <version>1.0</version>
    </resolved-release-id>
    <scanner status="DISPOSED"/>
  </kie-container>
</response>
```

3.5. MANAGING CONTAINERS

Containers within the Realtime Decision Server can be started, stopped, and updated from Business Central.

3.5.1. Starting, Stopping, and Deleting Containers

A container is stopped by default. To start the container:

1. Log in to Business Central.
2. In the main menu on the top, click **Deploy** → **Execution Servers**.
3. Select your server from the **SERVER TEMPLATES** section on the left side of the page.
4. Find the container you want to start under the **KIE CONTAINERS** section on the left.
5. Click **Start** at the upper right hand corner.
   Alternatively, click **Stop** to stop a running container. Once a container is stopped, you can click **Remove** to remove it.

3.5.2. Upgrading Containers

It is possible to update deployed containers without restarting the Realtime Decision Server, which can be used in cases where business rule changes cause new versions of packages to be provisioned. You can have multiple versions of the same package provisioned and deployed.

To upgrade a container:

1. Log in to Business Central.
2. In the main menu on the top, click **Deploy** → **Execution Servers**.
3. Select your server from the **SERVER TEMPLATES** section on the left side of the page.
4. Find the container you want to upgrade under the **KIE CONTAINERS** section on the left.
5. Click on the **Version Configuration** tab at the top.
6. Enter a new version and click **Upgrade**.
   Optionally, if you want a deployed container to always have the latest version of your deployment without manually editing it, set the **Version** value to **LATEST** and click **Scan Now**. If a newer version of a container deployment is found in the repository during the scanning, the container is automatically upgraded to this newer version. To start the scanner in the background, click **Start Scanner** and specify a scan interval in milliseconds.

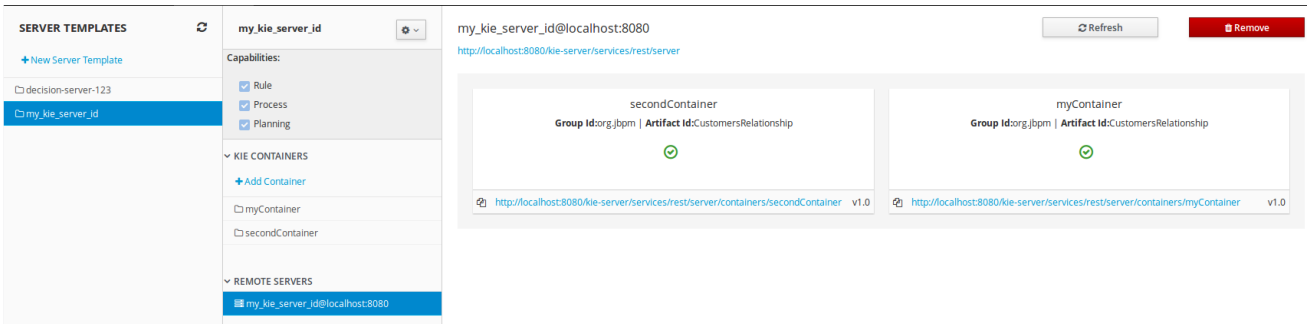
   The **Version** value can be set to **LATEST** if you are creating a deployment for the first time.

3.5.3. Managing Multiple Containers

The Realtime Decision Server allows you to create and provision multiple containers.

Select your server under the **REMOTE SERVERS** section to view all containers and their statuses.

Figure 3.2. Managing Multiple Containers



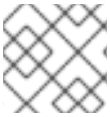
CHAPTER 4. LOGGING

4.1. LOGBACK FUNCTIONALITY

Red Hat JBoss BRMS provides **logback** functionality for logging configuration.

Accordingly, everything configured is logged to the *Simple Logging Facade for Java* SLF4J, which delegates any log to Logback, Apache Commons Logging, Log4j or java.util.logging. Add a dependency to the logging adaptor for your logging framework of choice. If you are not using any logging framework yet, you can use Logback by adding this Maven dependency:

```
<dependency>
   <groupId>ch.qos.logback</groupId>
   <artifactId>logback-classic</artifactId>
   <version>1.x</version>
</dependency>
```



NOTE

slf4j-nop and **slf4j-simple** are ideal for a light environment.

4.2. CONFIGURING LOGGING

To configure the logging level of the packages, create a **logback.xml** file in **business-central.war/WEB-INF/classes/logback.xml**. To set the logging level of the **org.drools** package to "debug" for verbose logging, you would need to add the following line to the file:

```
<configuration>
  <logger name="org.drools" level="debug"/>

  ...
<configuration>
```

Similarly, you can configure logging for packages such as the following:

- **org.guvnor**
- **org.jbpm**
- **org.kie**
- **org.slf4j**
- **org.dashbuilder**
- **org.uberfire**
- **org.errai**
- etc…

If configuring with **log4j**, the **log4j.xml** can be located at **business-central.war/WEB-INF/classes/log4j.xml** and can be configured in the following way:

```
<log4j:configuration xmlns:log4j="http://jakarta.apache.org/log4j/">

  <category name="org.drools">
    <priority value="debug" />
  </category>

  ...

</log4j:configuration>
```

NOTE

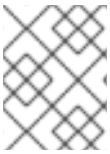
Additional logging can be configured in the individual container. To configure logging for JBoss Enterprise Application Platform, please refer to the Red Hat JBoss Enterprise Application Platform Administration and Configuration Guide.

CHAPTER 5. REPOSITORY HOOKS

In Business Central, it is possible to trigger a chosen action every time a particular event happens. For this purpose, you can configure the repository to use scripts called hooks.

5.1. CONFIGURING GIT HOOKS

Business Central can automatically push changes to a remote repository using the Git hooks. Git hooks support has been introduced with the release of Red Hat JBoss BRMS 6.2.0.

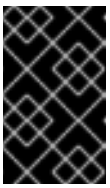


NOTE

Please note that currently only the **`post-commit`** hook is supported. **`Post-commit`** hooks are triggered after finishing the entire commit process.

The following procedure shows how to configure the **`post-commit`** hook to automatically push your changes to the remote repository.

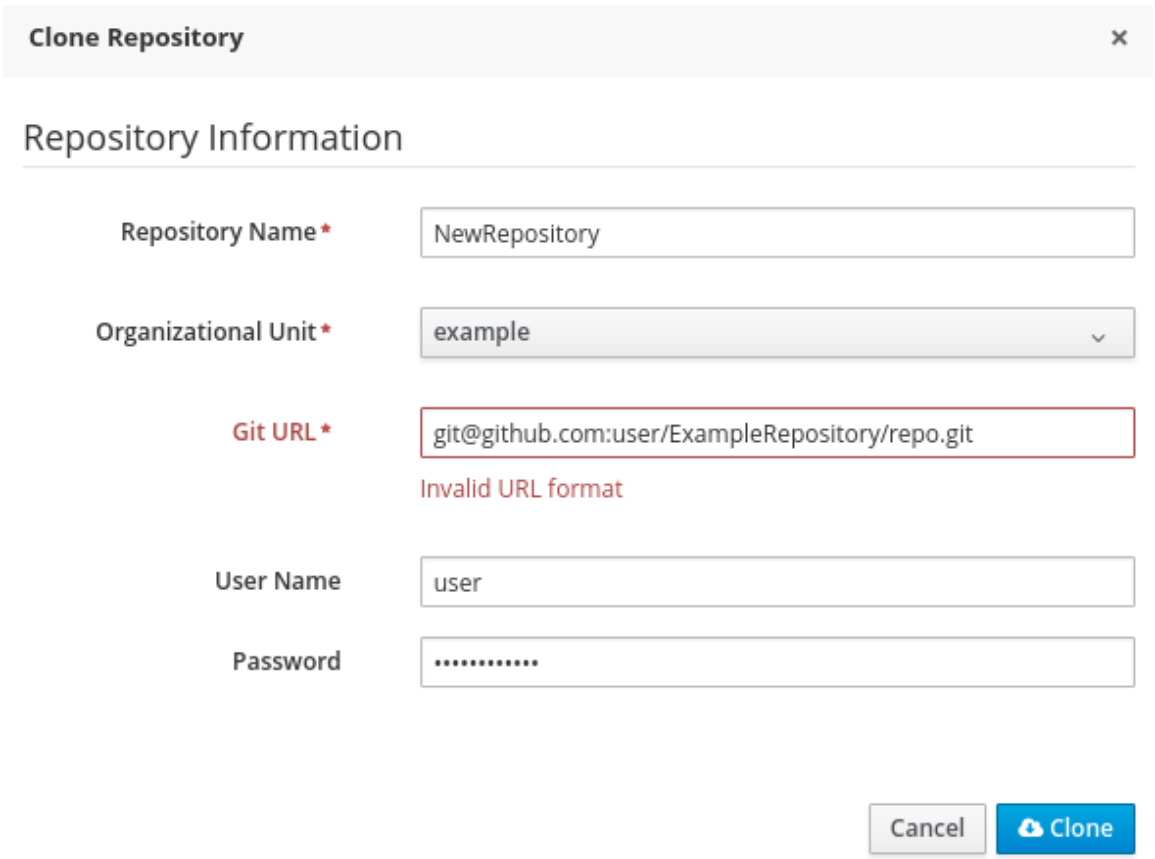
1. In Business Central, go to **Authoring** → **Administration**.
2. Below the main menu, click **Repositories** → **Clone repository**.
3. In the displayed **Clone repository** dialog box, fill in the repository information:
   - Repository Name
   - Organizational Unit
   - Git URL: For example **`https://github.com/USERNAME/REPOSITORY_NAME.git`**



IMPORTANT

It is important to use the HTTPS or Git protocol instead of a SCP-style SSH URL. Business Central does not support the basic SSH URL and fails with *Invalid URL format*.

Figure 5.1. An invalid SCP-style SSH URL.



- User Name: your Git user name
- Password: your Git password

4. Change to the created repository:

```
cd JBOSS_HOME/bin/.niogit/REPOSITORY_NAME.git
```

5. Change the remote URL:

```
git remote set-url origin
git@github.com:USERNAME/REPOSITORY_NAME.git
```

Make sure that you can access the remote repository through command line using SSH. For example, the private SSH key for the repository should exist under the **~/.ssh/** directory.

If you created a new repository, you may encounter the following error:

```
fatal: No such remote 'origin'
```

To resolve it, add the remote origin URL:

```
git remote add origin git@github.com:USERNAME/REPOSITORY_NAME.git
```

6. Verify that the remote repository was successfully added:

```
git remote -v
```

The command should list the following:

```
origin  git@github.com:USERNAME/REPOSITORY_NAME.git (fetch)
origin  git@github.com:USERNAME/REPOSITORY_NAME.git (push)
```

7. Create a file named **`post-commit`** with the permissions set to **`rwxr--r--`** under **`JBOSS_HOME/bin/.niogit/REPOSITORY_NAME.git/hooks`** with the following content:

```
#!/bin/sh
git push origin master
```

8. Make sure that the configuration was successful by creating a new guided rule in Business Central:
   a. Go to **Authoring** → **Project Authoring** → **New Item** → **Guided Rule**.
   b. Fill in the required information in the displayed **Create new Guided Rule** window.
   c. Click **Ok**.

All of the changes should be pushed automatically.

For further information about remote Git repositories, see How to configure the BxMS 6 server to use a remote Git repository for storing assets?.

It is also possible to specify the system property **`org.uberfire.nio.git.hooks`**. Its value determines a directory with default hook files, which will be copied to the newly created Git repositories. See the example of a **`standalone.xml`** file with this setting below:

```
<system-properties>
  <property name="org.uberfire.nio.git.hooks" value="/opt/jboss-as/git-
hooks">
  </property>
  ...
</system-properties>
```

CHAPTER 6. COMMAND LINE CONFIGURATION

The **kie-config-cli** tool is a command line configuration tool that provides capabilities to manage the system repository from the command line and can be used in an online or offline mode.

Online mode (default and recommended)

On startup, the tool connects to a Git repository using a Git server provided by **kie-wb**. All changes are made locally and published to upstream only after explicitly executing the push-changes command. Use the exit command to publish local changes. To discard local changes on exit, use the discard command.

Offline mode (installer style)

Creates and manipulates the system repository directly on the server (there is no discard option).

The tool is available on the Red Hat Customer Portal. To download the **kie-config-cli** tool, do the following:

1. Go to the Red Hat Customer Portal and log in.
2. Click **DOWNLOADS** at the top of the page.
3. In the **Product Downloads** page that opens, click **Red Hat JBoss BRMS**.
   a. From the **Version** drop-down menu, select **6.4.0**.
   b. In the displayed table, navigate to the **Supplementary Tools** row and then click **Download**.

Extract the zip package for supplementary tools you downloaded from the Red Hat Customer Portal. It contains the directory **kie-config-cli-6.MINOR_VERSION-redhat-x-dist** with file **kie-config-cli.sh**.

6.1. STARTING THE KIE-CONFIG-CLI TOOL IN ONLINE MODE

1. To start the **kie-config-cli** tool in online mode, navigate to the **kie-config-cli-6.MINOR_VERSION-redhat-x-dist** directory where you installed the tool and then execute the following command.
2. In a Unix environment run:

```
./kie-config-cli.sh
```

In a Windows environment run:

```
./kie-config-cli.bat
```

By default, the tool starts in online mode and asks for user credentials and a Git URL to connect to (the default value is **git://localhost/system**). To connect to a remote server, replace the host and port with appropriate values.

Example: **git://kie-wb-host:9148/system**

6.2. STARTING THE KIE-CONFIG-CLI TOOL IN OFFLINE MODE

To operate in offline mode, append the offline parameter to the command as below.

1. Navigate to the **kie-config-cli-6.MINOR_VERSION-redhat-x-dist** directory where you installed the tool.
2. In a Unix environment, run:

```
./kie-config-cli.sh offline
```

   In a Windows environment, run:

```
./kie-config-cli.bat offline
```

Executing this command changes the tool's behaviour and displays a request to specify the folder where the system repository (**.niogit**) is located. If .niogit does not yet exist, the folder value can be left empty and a brand new setup is created.

6.3. COMMANDS AVAILABLE FOR THE KIE-CONFIG-CLI TOOL

The following commands are available for managing the Git repository using the **kie-config-cli** tool:

- **add-deployment**: Adds a new deployment unit
- **add-repo-org-unit**: Adds a repository to the organizational unit
- **add-role-org-unit**: Adds role(s) to an organizational unit
- **add-role-project**: Adds role(s) to a project
- **add-role-repo**: Adds role(s) to a repository
- **create-org-unit**: Creates new organizational unit
- **create-repo**: Creates a new git repository
- **discard**: Does not publish local changes, cleans up temporary directories and closes the tool
- **exit**: Publishes work, cleans up temporary directories and closes the tool
- **fetch-changes**: Fetches changes from upstream repository
- **help**: Prints available commands with descriptions
- **list-deployment**: Lists available deployments
- **list-org-units**: Lists available organizational units
- **list-repo**: Lists available repositories
- **push-changes**: Pushes changes to upstream repository (in online mode only)
- **remove-deployment**: Removes existing deployment
- **remove-org-unit**: Removes existing organizational unit
- **remove-repo**: Removes an existing repository from config only

- **`remove-repo-org-unit`**: Removes a repository from the organizational unit
- **`remove-role-org-unit`**: Removes role(s) from an organizational unit
- **`remove-role-project`**: Removes role(s) from a project
- **`remove-role-repo`**: Removes role(s) from a repository

CHAPTER 7. MIGRATION

Migrating your projects from Red Hat JBoss BRMS 5 to Red Hat JBoss BRMS 6 requires careful planning and step by step evaluation of the various issues. You can plan for migration either manually, or by using automatic processes. Most real world migration will require a combination of these two processes.

Because Red Hat JBoss BRMS 6 uses Git for storing assets, artifacts and code repositories including processes and rules, you should start by creating an empty project in Red Hat JBoss BRMS 6 as the basis for your migration with dummy files as placeholders for the various assets and artifacts. Running a Git clone of this empty project into your favorite IDE will initiate the migration process.

Based on the placeholder files in your cloned project, you can start adding assets at the correct locations. The Red Hat JBoss BRMS 6 system is smart enough to pick these changes and apply them correctly. Ensure that when you are importing old rule files that they are imported with the right package name structure.

Since Maven is used for building projects, the projects assets like the rules, processes and models are accessible as a simple JAR file.

This section lists the generally accepted step by step ways to migrate your project. These are just guidelines though, and actual migration may vary a lot from this.

In general, you should:

1. Migrate the data first: These are your business assets.
2. Next, migrate your runtime processes.
3. Finally, convert old API calls to new ones one by one.

Let us look at these steps in more detail in the next few sections:

7.1. DATA MIGRATION

To migrate data from Red Hat JBoss BRMS 5, do the following:

1. Download the migration tool by logging in at the Red Hat Customer Portal and then navigating to Red Hat JBoss BRMS Software Downloads section. Click on **Red Hat JBoss BRMS Migration Tool** to download the zip archive.
2. Unzip the downloaded zip archive in a directory of your choice and navigate to this directory in a command prompt. This directory contains four folders:
   - **`bin`** - contains the launch scripts.
   - **`jcr-exporter-libs`** - contains the libs specific to the **`export-from-JCR`** part of the migration.
   - **`vfs-importer-libs`** - contains the libs specific to the **`import-into-Git`** part of the migration.
   - **`conf`** - contains global migration tool configuration.
3. For production databases, copy the JDBC driver for the database that is used by the JCR repository into the **`jcr-exporter-libs`** directory of the migration tool.

4. Execute the following command:

```
./bin/runMigration.sh -i <source-path> -o <destination-path> -r
<repository-name>
```

Where:

- **<source-path>** is a path to a source JCR repository.
- **<desintation-path>** is a path to a destination Git VFS. This folder must not exist already.
- **<repository-name>** an arbitrary name for the new repository.

The repository is migrated at the specified destination.

Besides the **-i** command, you can also use **-h** to print out a help message and **-f** which forces an overwrite of the output directory, thus eliminating the need for manual deletion of this directory.

Importing the Repository in Business Central

The repository can be imported in business central by cloning it. In the Administration perspective, click on the **Repositories** → **Clone Repository** menu to start the process.

NOTE

Assets can also be migrated manually as they are all just text files. The BPMN2 specification and the DRL syntax did not change between the different versions.

Importing the Repository in JBDS

To import the repository in JBoss Developer Studio, do the following

1. Start JBoss Developer Studio.
2. Start the Red Hat JBoss BRMS server (if not already running) by selecting the server from the server tab and click the start icon.
3. Select **File** → **Import…** and navigate to the Git folder. Open the Git folder to select **Projects from Git** and click next.
4. Select the repository source as **Existing local repository** and click next.
5. Select the repository that is to be configured from the list of available repositories.
6. Import the project as a general project in the next window and click next. Name this project and click Finish.

7.2. API AND BACKWARDS COMPATIBILITY

Migrating to Version 6.1

In version 6.1, 5.*X* APIs are no longer officially supported.

Red Hat JBoss BRMS no longer provides backward compatibility with the rule, event, and process application programming interface (API) from Red Hat JBoss BRMS 5. The content of the **knowledge-api** JAR file is no longer supported in version 6.1 and is replaced by APIs contained in the **kie-api** JAR file that were introduced in Red Hat JBoss BRMS 6.0.

If you used the legacy 5.x API (located in **knowledge-api.jar**), please migrate (rewrite) the API calls to the new KIE API. Please be aware that several other APIs have changed between Red Hat JBoss BRMS 5.x and Red Hat JBoss BRMS 6.x, namely the task service API and the REST API.

Migrating to Version 6.0

The Red Hat JBoss BRMS 6 system provides backward compatibility with the rule, event and process interactions from JBoss BRMS 5. You should eventually migrate (rewrite) these interactions to the all new revamped core API because this backward compatibility is likely to be deprecated.

If you cannot migrate your code to use the new API, then you can use the API provided by the purpose built **knowledge-api** JAR for backwards compatible code. This API is the public interface for working with JBoss BPM Suite and JBoss BRMS and is backwards compatible.

If you are instead using the REST API in Red Hat JBoss BRMS 5, note that this has changed as well and there is no mechanism in it for backwards compatibility.

PART II. INTEGRATION

CHAPTER 8. INTEGRATING RED HAT JBOSS BRMS WITH RED HAT JBOSS FUSE

Red Hat JBoss Fuse integration allows users of Red Hat JBoss Fuse to complement their integration solution with additional features provided by Red Hat JBoss BPM Suite and Red Hat JBoss BRMS.

Red Hat JBoss BRMS integration is provided by two **`features.xml`** files:

- **`drools-karaf-features-VERSION-features.xml`**
  This file provides core Red Hat JBoss BPM Suite and Red Hat JBoss BRMS features, which defines the OSGi features that can be deployed into Red Hat JBoss Fuse. This file is a part of the Red Hat JBoss BPM Suite and Red Hat JBoss BRMS product. OSGi users can install features from this file in order to install Red Hat JBoss BRMS engine or Red Hat JBoss BPM Suite engine into Red Hat JBoss Fuse and use it in their applications.

- **`karaf-features-VERSION-features.xml`**
  This file provides additional features used for integrating Red Hat JBoss BPM Suite and Red Hat JBoss BRMS with Apache Camel, primarily in Red Hat JBoss Fuse. This file is part of the Integration Pack and it defines OSGi features that enable integration with Apache Camel and SwitchYard. In addition to the **`karaf-features`** XML file, the Integration Pack also contains a **`features.xml`** file for quick starts.

- **`drools-karaf-features-VERSION-features-fuse-6_3.xml`**
  This file is used for integration of Red Hat JBoss BRMS with Red Hat JBoss Fuse 6.3.0.

For further information about integration of Red Hat JBoss BRMS with Red Hat JBoss Fuse, see the Install Integration Pack chapter of the *Red Hat JBoss Fuse Integration Guide*.

8.1. CORE RED HAT JBOSS BPM SUITE AND RED HAT JBOSS BRMS FEATURES

Core Red Hat JBoss BPM Suite and Red Hat JBoss BRMS features are provided by one of the following files:

- For integration with Red Hat JBoss Fuse 6.2.1, use the **`drools-karaf-features-VERSION-features.xml`** file.

- For integration with Red Hat JBoss Fuse 6.3.0, use the **`drools-karaf-features-VERSION-features-fuse-6_3.xml`** file
  `VERSION` is the version of Red Hat JBoss BRMS you are using, for example **`6.5.0.Final-redhat-9`**. See the Supported Component Versions chapter of the *Red Hat JBoss BRMS Installation Guide*.

The files are present in the product Maven repository (**`org/drools/drools-karaf-features`**) or the **`jboss-brms-bpmsuite-VERSION-redhat-VERSION-fuse-features.zip`** file. These files provide the following features:

- **`drools-common`**
- **`drools-module`**
- **`drools-templates`**
- **`drools-decisiontable`**

- **drools-jpa**
- **kie**
- **kie-ci**
- **kie-spring**
- **kie-aries-blueprint**
- **jbpm-commons**
- **jbpm-human-task**
- **jbpm**
- **droolsjbpm-hibernate**
- **h2**

The following table provides example of use cases for some of the features listed above.

Table 8.1. Features and Use Case Examples

Feature	Use Case
drools-module	Use the Red Hat JBoss BRMS engine for rules evaluation, without requiring persistence, processes, or decision tables.
drools-jpa	Use the Red Hat JBoss BRMS engine for rules evaluation with persistence and transactions, but without requiring processes or decision tables. The **drools-jpa** feature already includes **drools-module**, however you may also need to install the **droolsjbpm-hibernate** feature, or ensure there is a compatible hibernate bundle installed.
drools-decisiontable	Use the Red Hat JBoss BRMS engine with decision tables.
jbpm	Use the Red Hat JBoss BPM Suite (or Red Hat JBoss BRMS engine with processes). The **jbpm** feature already includes **drools-module** and **drools-jpa**. You may also need to install the **droolsjbpm-hibernate** feature, or ensure that there is a compatible hibernate bundle installed.
jbpm and **jbpm-human-task**	Use the Red Hat JBoss BPM Suite (or Red Hat JBoss BRMS engine with processes) with Human Task.

Feature	Use Case
core engine JARs and **kie-ci**	Use Red Hat JBoss BRMS or Red Hat JBoss BPM Suite with **KieScanner** (KIE-CI) to download kJARs from a Maven repository.
kie-spring	Use KIE-Spring integration. See the section called "kie-spring Feature Further Information" for more information.
kie-spring and **kie-aries-blueprint**	Use KIE-Aries-Blueprint integration.

kie-spring Feature Further Information

- Use **org.drools.osgi.spring.OsgiKModuleBeanFactoryPostProcessor** instead of **org.kie.spring.KModuleBeanFactoryPostProcessor** to postprocess KIE elements in an OSGi environment.
- Do not install the **drools-module** feature before the **kie-spring** feature. Otherwise, the **drools-compiler** bundle does not detect packages exported by **kie-spring**. Run ***osgi:refresh drools-compiler_bundle_ID*** if you have installed the features in the incorrect order to force **drools-compiler** to rebuild its Import-Package metadata.

8.2. ADDITIONAL FEATURES FOR SWITCHYARD AND APACHE CAMEL INTEGRATION

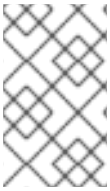
The following additional features for integration with SwitchYard and Apache Camel on Red Hat JBoss Fuse are provided by the integration pack:

- **fuse-bxms-switchyard-common-knowledge**
- **fuse-bxms-switchyard-rules**
- **fuse-bxms-switchyard-bpm**
- **kie-camel**
- **jbpm-workitems-camel**

The integration pack features are defined in the **karaf-features-*VERSION*-features.xml** file. This file (and supporting repositories) is located in http://repository.jboss.org/nexus/content/repositories/public, which is already configured for use on Red Hat JBoss Fuse out of the box in ***INSTALLATION_DIRECTORY*/etc/org.ops4j.pax.url.mvn.cfg**.

The file can also be downloaded from either the Red Hat JBoss Fuse product page or Red Hat JBoss BRMS product page on the Red Hat Customer Portal.

8.3. INSTALLING AND UPDATING CORE INTEGRATION FEATURES



NOTE

This section refers to features in the **drools-karaf-features-*VERSION*-features.xml** file. For additional integration features, see Section 8.4, "Installing Additional Integration Features".

If you have already installed an older version of the core Red Hat JBoss BPM Suite and Red Hat JBoss BRMS features (for example, **drools-karaf-features-6.2.0.Final-redhat-6-features.xml**), you need to remove them and all associated files before installing the most recent **features.xml** file.

Procedure: Removing Existing drools-karaf-features Installation

1. Start the Red Hat JBoss Fuse console using:

```
$ ./INSTALLATION_DIRECTORY/bin/fuse
```

2. Unistall old features or applications that used the previous **features.xml** file. For example:

```
JBossFuse:karaf@root> features:uninstall drools-module
JBossFuse:karaf@root> features:uninstall jbpm
JBossFuse:karaf@root> features:uninstall kie-ci
```

3. Search for references of bundles using **drools**, **kie**, or **jbpm**, and remove them:

```
karaf@root> list -t 0 -s | grep drools
karaf@root> list -t 0 -s | grep kie
karaf@root> list -t 0 -s | grep jbpm
```

   To remove the bundles:

```
karaf@root> osgi:uninstall BUNDLE_ID
```

4. Remove the old **drools-karaf-features** URL:

```
karaf@root> features:removeurl mvn:org.drools/drools-karaf-
features/6.2.0.Final-redhat-VERSION/xml/features
```

5. Restart Red Hat JBoss Fuse.

To install the **drools-karaf-features**:

Procedure: Installing Core Red Hat JBoss BPM Suite and Red Hat JBoss BRMS Features

1. Configure required repositories:

   - Edit the ***INSTALLATION_DIRECTORY*/etc/org.ops4j.pax.url.mvn.cfg** file in your Red Hat JBoss Fuse installation and add the following entry to the **org.ops4j.pax.url.mvn.repositories** variable (note that entries are separated by **,** \):

     - **https://maven.repository.redhat.com/ga/**

2. Start Red Hat JBoss Fuse:

```
$ ./INSTALLATION_DIRECTORY/bin/fuse
```

3. Add a reference to the core features file by running the following console command:
For Red Hat JBoss Fuse 6.2.1, use:

```
JBossFuse:karaf@root> features:addurl mvn:org.drools/drools-karaf-
features/VERSION/xml/features
```

For Red Hat JBoss Fuse 6.3.0, use:

```
JBossFuse:karaf@root> features:addurl mvn:org.drools/drools-karaf-
features/VERSION/xml/features-fuse-6_3
```

For example:

```
features:addurl mvn:org.drools/drools-karaf-features/6.4.0.Final-
redhat-10/xml/features-fuse-6_3
```

To see the current **drools-karaf-features** version, see the Supported Component Versions chapter of the *Red Hat JBoss BRMS Installation Guide*.

4. You can now install the features provided by this file by running, for example, the following console command:

```
JBossFuse:karaf@root> features:install drools-module
```

8.4. INSTALLING ADDITIONAL INTEGRATION FEATURES

Use the following procedure for additional integration with SwitchYard and Apache Camel.

Procedure: SwitchYard and Apache Camel Integration

1. Download the **fuse-integration** package that is aligned with your version of Red Hat JBoss Fuse.

NOTE

For instance, if you want to use the **6.2.0.redhat-117** version of Red Hat JBoss Fuse, you need to install the **fuse-6.2.0.redhat-117** Red Hat JBoss Fuse integration features.

2. Add the remote Maven repository that contains the fuse dependencies to your **karaf** instance:
   - Edit ***FUSE_HOME*/etc/org.ops4j.pax.url.mvn.cfg**
3. Update the Drools features URL:

```
JBossFuse:karaf@root> features:addurl
mvn:org.switchyard.karaf/mvn:org.switchyard.karaf/switchyard/SWITCHY
ARD_VERSION/xml/core-features
```

```
JBossFuse:karaf@root> features:addurl
mvn:org.jboss.integration.fuse/karaf-features/1.0.0.redhat-
VERSION/xml/features
```

Additionally, update the **drools-karaf-features** URL. For Red Hat JBoss Fuse 6.2.1, use:

```
JBossFuse:karaf@root> features:addurl mvn:org.drools/drools-karaf-
features/VERSION/xml/features
```

For Red Hat JBoss Fuse 6.3.0, use:

```
JBossFuse:karaf@root> features:addurl mvn:org.drools/drools-karaf-
features/VERSION/xml/features-fuse-6_3
```

To see the current **drools-karaf-features** version, see the Supported Component Versions chapter of the *Red Hat JBoss BRMS Installation Guide*.

4. You can now install the features provided for SwitchYard and Apache Camel integration by running, for example, the following console command:

```
JBossFuse:karaf@root> features:install fuse-bxms-switchyard-rules
JBossFuse:karaf@root> features:install kie-camel
JBossFuse:karaf@root> features:install jbpm-workitems-camel
```

8.5. CONFIGURING DEPENDENCIES

When you configure KIE, Red Hat JBoss BRMS, or Red Hat Jboss BPM Suite in your application, you can follow one of the following approaches to build your OSGi application bundles:

- Bundle required dependencies into your application bundle. In this approach, you declare all required artifacts as runtime dependencies in your **pom.xml**. Hence, you need not import the packages that provide these artifacts that you have already added as dependencies.
- Import the required dependencies into the application bundle. This is a preferred approach for building OSGi bundles as it adheres to the principles of OSGi framework. In this approach, you declare only the API JARs (such as **org.kie:kie-api**) as dependencies in your application bundle. You will need to install the required Red Hat JBoss BRMS and Red Hat JBoss BPM Suite bundles and then import them in your application.

WARNING

The **MVELUserGroupCallback** class fails to initialize in an OSGi environment. Do *not* use or include **MVELUserGroupCallback** as it is not designed for production purposes.

8.6. INSTALLING RED HAT JBOSS FUSE INTEGRATION QUICK START APPLICATIONS

The following features for Red Hat JBoss Fuse integration quick start applications are provided by **org/jboss/integration/fuse/quickstarts/karaf-features/VERSION/karaf-features-VERSION-features.xml**:

- **fuse-bxms-quickstart-switchyard-bpm-service**
- **fuse-bxms-quickstart-switchyard-rules-camel-cbr**
- **fuse-bxms-quickstart-switchyard-rules-interview**
- **fuse-bxms-quickstart-switchyard-rules-interview-container**
- **fuse-bxms-quickstart-switchyard-rules-interview-dtable**
- **fuse-bxms-quickstart-switchyard-library**
- **fuse-bxms-quickstart-switchyard-helpdesk**
- **fuse-bxms-quickstart-camel-blueprint-drools-decision-table**
- **fuse-bxms-quickstart-camel-spring-drools-decision-table**
- **fuse-bxms-quickstart-camel-jbpm-workitems**
- **fuse-bxms-spring-jbpm-osgi-example**

This file (and supporting repositories) is located in http://repository.jboss.org/nexus/content/repositories/public, which is already configured for use on Red Hat JBoss Fuse out of the box in ***INSTALLATION_DIRECTORY*/etc/org.ops4j.pax.url.mvn.cfg**.

Procedure: Installing Quick Start Applications

1. Add a reference to the features file by running the following console command:

```
JBossFuse:karaf@root> features:addurl
mvn:org.jboss.integration.fuse.quickstarts/karaf-
features/1.0.0.redhat-VERSION/xml/features
```

2. You can now install the quick start applications provided by this features file by running, for example, the following console command:

```
JBossFuse:karaf@root> features:install fuse-bxms-quickstart-
switchyard-bpm-service
```

Procedure: Downloading and Installing Quick Start ZIP Files

1. Download the quick start application ZIP file.
2. Unpack the contents of the quick starts directory into your existing ***INSTALLATION_DIRECTORY*/quickstarts** directory.
3. Unpack the contents of the system directory into your existing ***INSTALLATION_DIRECTORY*/system** directory.

8.6.1. Testing Your First Quick Start Application

Procedure: Testing Quick Start Application

1. Start Red Hat JBoss Fuse:

```
$ ./INSTALLATION_DIRECTORY/bin/fuse
```

2. Install and start the **switchyard-bpm-service** by running the following console command:

```
JBossFuse:karaf@root> features:install fuse-bxms-quickstart-
switchyard-bpm-service
```

NOTE

Any dependent features specified by the application's features file will be installed automatically.

3. Submit a web service request to invoke the SOAP gateway.

   a. Open a terminal window and navigate to the associated quick start directory that was unpacked from the quick start application ZIP file (in this case, **switchyard-bpm-service**).

   b. Run the following command:

```
$ mvn clean install
```

NOTE

You will need the following repositories configured in your **settings.xml** file:

- **http://maven.repository.redhat.com/ga/**
- **http://repository.jboss.org/nexus/content/repositories/public/**

   c. Run the following command:

```
$ mvn exec:java -Pkaraf
```

4. You will receive the following response:

```
SOAP Reply:
<soap:Envelope
xmlns:soap="http://schemas.xmlsoap.org/soap/envelope/">
  <SOAP-ENV:Header xmlns:SOAP-
ENV="http://schemas.xmlsoap.org/soap/envelope/"/>
  <soap:Body>
    <ns2:submitOrderResponse xmlns:ns2="urn:switchyard-
quickstart:bpm-service:1.0">
```

```
      <orderId>test1</orderId>
      <accepted>true</accepted>
      <status>Thanks for your order, it has been shipped!</status>
    </ns2:submitOrderResponse>
  </soap:Body>
</soap:Envelope>
```

CHAPTER 9. INTEGRATING RED HAT JBOSS BRMS WITH RED HAT SINGLE SIGN-ON

Red Hat Single Sign-On (RH-SSO) is a Single Sign-On solution that you can use for securing your browser applications and your REST web services. This chapter describes how you can integrate RH-SSO with Red Hat JBoss BRMS and leverage its features.

Integrating with RH-SSO brings an integrated SSO and IDM (Identity Management) environment for Red Hat JBoss BRMS. The session management feature of RH-SSO allows you to use different Red Hat JBoss BRMS environments on the web by authenticating only once.

For more information on RH-SSO, see the RH-SSO documentation.

Red Hat Single Sign On Integration Points

You can integrate RH-SSO with Realtime Decision Servers using the following integration points:

- **Business Central authentication through an RH-SSO server**
  Authenticating Red Hat JBoss BRMS Business Central through RH-SSO involves securing both the Business Central web client and remote services through RH-SSO. This integration enables you to connect to Business Central using either web interface or a remote service consumer through RH-SSO.

- **Realtime Decision Server authentication through an RH-SSO server**
  Authenticating Red Hat JBoss BRMS Realtime Decision Server through RH-SSO involves securing the remote services provided by the Realtime Decision Server as it does not provide a web interface for server authentication. This enables any remote Red Hat JBoss BRMS service consumer (user or a service) to authenticate through RH-SSO.

- **Third-party client authentication through an RH-SSO server**
  Authenticating a third-party client through an RH-SSO server involves third-party clients to authenticate themselves using RH-SSO to consume the remote service endpoints provided by Business Central and Realtime Decision Server, such as the REST API or remote file system services.

The following sections describe how to achieve RH-SSO integration through these integration points:

9.1. BUSINESS CENTRAL AUTHENTICATION THROUGH RH-SSO

To authenticate Business Central through RH-SSO:

1. Set up and run an RH-SSO server with a realm client for Business Central.
2. Install and set up the RH-SSO client adapter for EAP.
3. Secure Business Central Remote Service using RH-SSO.

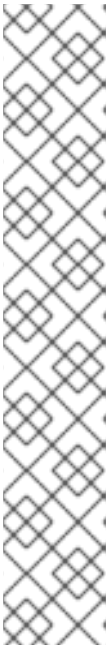
9.1.1. Setting Up RH-SSO with Realm Client for Business Central

Security realms are used to restrict access for the different applications resources. It is advised to create a new realm whether your RH-SSO instance is private or shared amongst other products. You can keep the master realm as a place for super administrators to create and manage the realms in your system. If you are integrating with an RH-SSO instance that is shared with other product installations to achieve Single Sign-On with those applications, all those applications must use the same realm.

Here is how you can install an RH-SSO server and create a security realm for Business Central:

Procedure: Setting Up RH-SSO with Realm Client

1. Download RH-SSO from the DOWNLOADS section of the Red Hat Customer Portal.
2. Install and configure a basic RH-SSO standalone server. To do this, follow the instructions in chapter Install and Boot of the *Red Hat Single Sign On Getting Started Guide*. For production environment settings, consult the *Red Hat Single Sign On* Server Administration Guide.



NOTE

If you want to run both RH-SSO and Red Hat JBoss BRMS servers on the same machine, ensure that you avoid port conflicts. To do so, do one of the following:

- Update the ***RHSSO_HOME*/standalone/configuration/standalone.xml** file and set a port offset to 100. For example:

```
<socket-binding-group name="standard-sockets" default-
interface="public" port-
offset="${jboss.socket.binding.port-offset:100}">
```

- Use an environment variable to run the server:

```
bin/standalone.sh -Djboss.socket.binding.port-
offset=100
```

3. Start the RH-SSO server using the following command from ***RHSSO_HOME*/bin**:

```
./standalone.sh
```

Once the RH-SSO server starts, open http://localhost:8180/auth/admin in a web browser and log in using your admin credentials that you created while installing RH-SSO. When you login for the first time, you can set up the initial user using the new user registration form.

4. In the RH-SSO Admin Console, click the **Realm Settings** left menu item.
5. On the **Realm Settings** page, click **Add Realm**.
The **Add realm** page opens.
6. On the **Add realm** page, provide a name for the realm and click**Create**.
7. Click the **Clients** left menu item and click **Create**.
The **Add Client** page opens.
8. On the **Add Client** page, provide the required information to create a new client for your realm. For example:
   - **Client ID**: kie
   - **Client protocol**: openid-connect
   - **Root URL**: http://*localhost*:8080/business-central
9. Click **Save** to save your changes.

Once you create a new client, its **Access Type** is set to **`public`** by default. Change it to **`confidential`**.

At this point, the RH-SSO server is configured with a realm with a client for Red Hat JBoss BRMS applications (Business Central, in this example) and running and listening for HTTP connections at **`localhost:8180`**. This realm provides different users, roles, and sessions for the Red Hat JBoss BRMS applications.

9.1.2. Setting Up RH-SSO Client Adapter for EAP

To set up the RH-SSO client adapter for EAP:

1. Install the RH-SSO adapter for EAP.
2. Configure the Red Hat JBoss BRMS application and the RH-SSO client adapter.

Procedure: Installing the RH-SSO Adapter for Red Hat JBoss EAP 6 and 7

1. Install Red Hat JBoss EAP 6.4.*X* or 7.0.
   For version 6, see chapter Installation Instructions from the *Red Hat JBoss Enterprise Application Platform Installation Guide*.

   For version 7, see chapter Installing JBoss EAP from the *Red Hat JBoss Enterprise Application Platform Installation Guide*.
2. Install Red Hat JBoss BRMS in the freshly installed JBoss EAP.
   If you configure RH-SSO adapter by making changes in **`standalone.xml`**, and then unzip Red Hat JBoss BRMS, you may overwrite and lose the RH-SSO adapter configuration.
3. Download the EAP adapter from the Red Hat Customer Portal.
4. Unzip and install the adapter. For installation instructions, see section JBoss EAP Adapter from the *Red Hat Single Sign On Securing Applications and Services Guide*.
5. For version 7, go to ***`EAP_HOME`*`/standalone/configuration`** and open the **`standalone.xml`** and **`standalone-full.xml`** files. Delete the **`<single-sign-on/>`** element from both of them.
   You do not need to perform this step for Red Hat JBoss EAP 6.

Procedure: Configuring the RH-SSO Adapter

1. Navigate to ***`EAP_HOME`*`/standalone/configuration`** in your EAP installation and edit **`standalone.xml`** to add the RH-SSO subsystem configuration. For example:

```
<subsystem xmlns="urn:jboss:domain:keycloak:1.1">
 <secure-deployment name="business-central.war">
   <realm>demo</realm>
   <realm-public-
key>MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCrVrCuTtArbgaZzL1hvh0xtL5m
c7o0NqPVnYXkLvgcwiC3BjLGw1tGEGoJaXDuSaRllobm53JBhjx33UNv+5z/UMG4kytB
WxheNVKnL6GgqlNabMaFfPLPCF8kAgKnsi79NMo+n6KnSY8YeUmec/p2vjO2NjsSAVcW
EQMVhJ31LwIDAQAB</realm-public-key>
   <auth-server-url>http://localhost:8180/auth</auth-server-url>
   <ssl-required>external</ssl-required>
   <enable-basic-auth>true</enable-basic-auth>
```

```
    <resource>kie</resource>
    <credential name="secret">759514d0-dbb1-46ba-b7e7-
ff76e63c6891</credential>
    <principal-attribute>preferred_username</principal-attribute>
  </secure-deployment>
</subsystem>
```

Here,

- **`secure-deployment name`**: Name of your application's WAR file.
- **`realm`**: Name of the realm that you created for the applications to use.
- **`realm-public-key`**: The public key of the realm you created. You can find the key in the **Keys** tab in the **Realm settings** page of the realm you created in the RH-SSO Admin Console. If you do not provide a value for **`realm-public-key`**, the server retrieves it automatically.
- **`auth-server-url`**: The URL for the RH-SSO authentication server.
- **`enable-basic-auth`**: The setting to enable basic authentication mechanism, so that the clients can use both token-based and basic authentication approaches to perform the requests.
- **`resource`**: The name for the client that you created.
- **`credential name`**: The secret key for the client you created. You can find the key in the **Credentials** tab on the **Clients** page of the RH-SSO Admin Console.
- **`principal-attribute`**: The login name of the user. If you do not provide this value, your User Id is displayed in the application instead of your user name.

NOTE

The RH-SSO server converts the user names to lowercase. Therefore, after integration with RH-SSO, your user name will appear in lowercase in Business Central. If you have user names in upper-case letters hard coded in business processes, the application may not be able to identify the upper-case user.

2. Navigate to ***`EAP_HOME`*`/bin/`** and start the EAP server using the following command:

```
./standalone.sh
```

NOTE

You can also configure RH-SSO adapter for EAP by updating your applications WAR file to use the RH-SSO security subsystem. However, the recommended approach is configuring the adapter through the RH-SSO subsystem. This means that you are updating EAP configuration instead of applying the configuration on each WAR file.

9.1.3. Adding a New User

To add new users and assign them a role to access Business Central:

1. Log in to the RH-SSO Admin Console and open the realm to which you wish to add a user.
2. Click the **Users** left menu item under the **Manage** section.
   An empty user list page called **Users** opens.
3. Click the **Add User** button on the empty user list to start creating your new user.
   An **Add user** page opens.
4. Provide user information on the **Add user** page and click **Save**.
5. Set a new password under the **Credentials** tab.
6. Assign the new user one of the roles that allow access to Business Central. For example, the **`admin`** or **`analyst`** role.
   Define the roles as realm roles in the **Realm Roles** tab under the **Roles** section.
7. Click **Role Mappings** tab on the **Users** page to assign roles.

You can now log in to your Red Hat JBoss BRMS application (in this example, Business Central) once the server is running using the user credentials you just created.

9.1.4. Securing Business Central Remote Service Using RH-SSO

Business Central provides different remote service endpoints that can be consumed by third-party clients using remote API. To authenticate those services through RH-SSO, you must disable a security filter called **`BasicAuthSecurityFilter`**. To do this, follow these steps:

Procedure: Disabling BasicAuthSecurityFilter

1. Open your application deployment descriptor file (**`WEB-INF/web.xml`**) and apply the following changes to it:
   - Remove the following lines to remove the servlet filter and its mapping for class **`org.uberfire.ext.security.server.BasicAuthSecurityFilter`**:

```
<filter>
  <filter-name>HTTP Basic Auth Filter</filter-name>
  <filter-
class>org.uberfire.ext.security.server.BasicAuthSecurityFilter</f
ilter-class>
  <init-param>
    <param-name>realmName</param-name>
    <param-value>KIE Workbench Realm</param-value>
  </init-param>
</filter>

<filter-mapping>
  <filter-name>HTTP Basic Auth Filter</filter-name>
  <url-pattern>/rest/*</url-pattern>
  <url-pattern>/maven2/*</url-pattern>
  <url-pattern>/ws/*</url-pattern>
</filter-mapping>
```

   - Add the following lines to add the **`security-constraint`** for the url-patterns that you have removed from the filter mapping:

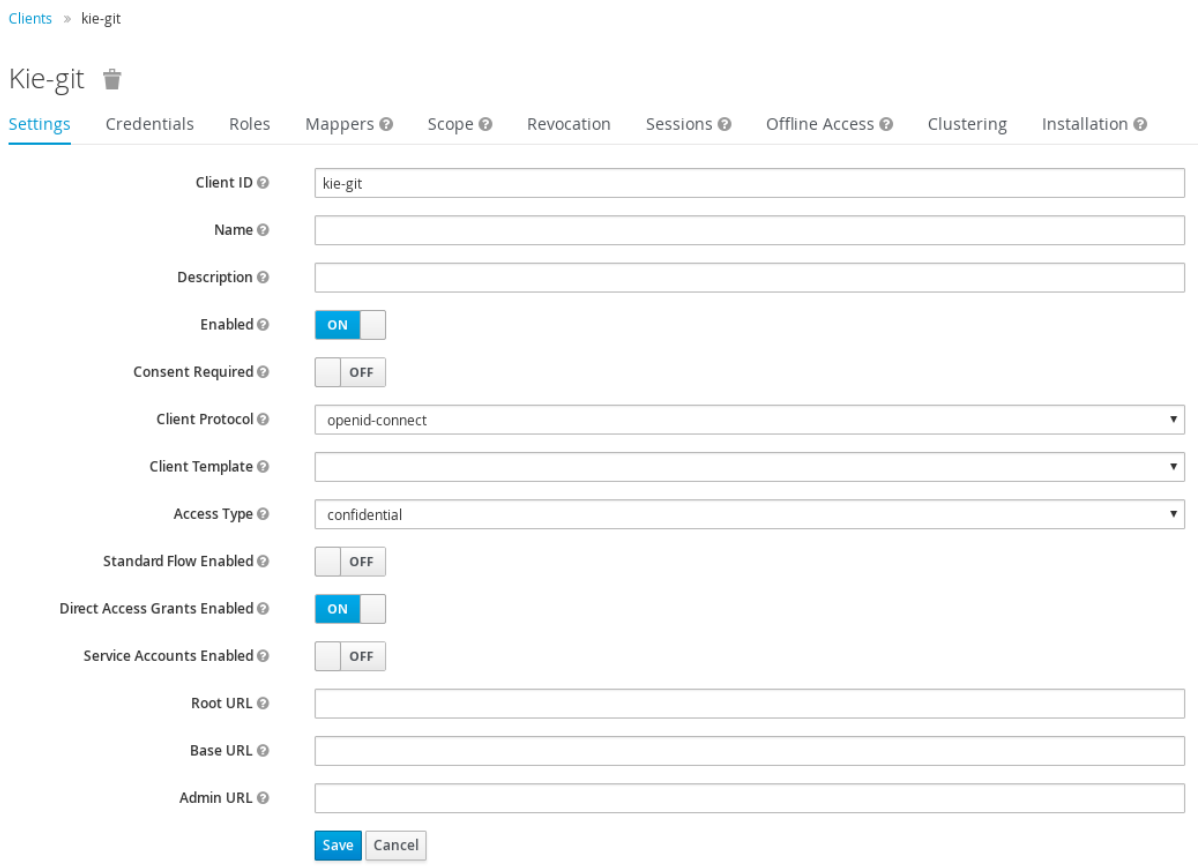
```
<security-constraint>
  <web-resource-collection>
    <web-resource-name>remote-services</web-resource-name>
    <url-pattern>/rest/*</url-pattern>
    <url-pattern>/maven2/*</url-pattern>
    <url-pattern>/ws/*</url-pattern>
  </web-resource-collection>
  <auth-constraint>
    <role-name>rest-all</role-name>
    <role-name>rest-project</role-name>
    <role-name>rest-deployment</role-name>
    <role-name>rest-process</role-name>
    <role-name>rest-process-read-only</role-name>
    <role-name>rest-task</role-name>
    <role-name>rest-task-read-only</role-name>
    <role-name>rest-query</role-name>
    <role-name>rest-client</role-name>
  </auth-constraint>
</security-constraint>
```

2. Save your changes.

9.1.5. Securing Business Central File System Services Using RH-SSO

To consume other remote services, such as file systems (for example, a remote GIT service), you need to specify a correct RH-SSO login module. First, generate a JSON configuration file:

1. Navigate to the **RH-SSO Admin Console** located at http://localhost:8080/auth/admin.
2. Click the **Clients** left menu item.
3. Create a new client with the following settings:
   - Set **Client ID** as **`kie-git`**.
   - Set **Access Type** as **`confidential`**.
   - Disable the **Standard Flow Enabled** option.
   - Enable the **Direct Access Grants Enabled** option.



4. Click **Save**.
5. Click the **Installation** tab at the top of the client configuration screen and choose **Keycloak OIDC JSON** as a **Format Option**.
6. Click **Download**.
7. Move the downloaded JSON file to an accessible directory in the server's file system or add it to the application class path.
   For more information, see chapter JAAS plugin of the Keycloak *Securing Applications and Services Guide*.

After you successfuly generate and download the JSON configuration file, specify the correct RH-SSO login module in the ***EAP_HOME*/standalone/configuration/standalone.xml** file. By default, the security domain in Business Central is set to **other**. Replace the default values of the **login-module** in this security domain with the ones presented in the example below:

```
<security-domain name="other" cache-type="default">
  <authentication>
    <login-module
code="org.keycloak.adapters.jaas.DirectAccessGrantsLoginModule"
flag="required">
      <module-option name="keycloak-config-file" value="$EAP_HOME/kie-
git.json"/>
    </login-module>
  </authentication>
</security-domain>
```

The JSON file specified in the **module-option** element contains a client used for securing the remote services. Replace the **$*EAP_HOME*/kie-git.json** value of the **module-option** element with the

absolute path or the class path (**classpath:/*EXAMPLE_PATH*/kie-git.json**) to this JSON configuration file.

At this point, all users authenticated through the RH-SSO server can clone internal GIT repositories. In the following command, change *USER_NAME* to a RH-SSO user, for example **admin**:

```
git clone ssh://USER_NAME@localhost:8001/system
```

9.2. REALTIME DECISION SERVER AUTHENTICATION THROUGH RH-SSO

The Red Hat JBoss BRMS Realtime Decision Server provides a REST API for third-party clients. You can integrate the Realtime Decision Server with RH-SSO to delegate the third-party clients identity management to the RH-SSO server.

Once you have created a realm client for Business Central and set up the RH-SSO client adapter for EAP, you can repeat the same steps to integrate the Realtime Decision Server with RH-SSO.

9.2.1. Creating Client for Realtime Decision Server on RH-SSO

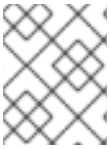
To create a new client on your already created realm on RH-SSO Admin Console:

Procedure: Creating a Client for the Realtime Decision Server

1. In the RH-SSO Admin Console, open the security realm that you created.
2. Click the **Clients** left menu item and click **Create**.
   The **Add Client** page opens.
3. On the **Add Client** page, provide the required information to create a new client for your realm. For example:
   - **Client ID**: kie-execution-server
   - **Root URL**: http://*localhost*:8080/kie-server
   - **Client protocol**: openid-connect
4. Click **Save** to save your changes.
   Once you create a new client, its **Access Type** is set to **public** by default. Change it to **confidential** and click **Save** again.
5. Navigate to the **Credentials** tab and copy the secret key. The secret key is necessary to configure the kie-execution-server client in the next section.

9.2.2. Installing and Setting Up Realtime Decision Server with Client Adapter

To consume the Realtime Decision Server remote service endpoints, you must first create and assign the **kie-server** role in the RH-SSO Admin Console.



NOTE

If you deployed the Realtime Decision Server to a different application server than Business Central, install and configure RH-SSO on your second server as well.

Procedure: Setting Up the Realtime Decision Server

1. Navigate to ***EAP_HOME*/standalone/configuration** in your EAP installation and edit **standalone.xml** to add the RH-SSO subsystem configuration. For example:

```
<subsystem xmlns="urn:jboss:domain:keycloak:1.1">
  <secure-deployment name="kie-server.war">
     <realm>demo</realm>
     <realm-public-
key>MIGfMA0GCSqGSIb3DQEBAQUAA4GNADCBiQKBgQCrVrCuTtArbgaZzL1hvh0xtL5m
c7o0NqPVnYXkLvgcwiC3BjLGw1tGEGoJaXDuSaRllobm53JBhjx33UNv+5z/UMG4kytB
WxheNVKnL6GgqlNabMaFfPLPCF8kAgKnsi79NMo+n6KnSY8YeUmec/p2vjO2NjsSAVcW
EQMVhJ31LwIDAQAB</realm-public-key>
     <auth-server-url>http://localhost:8180/auth</auth-server-url>
     <ssl-required>external</ssl-required>
     <resource>kie-execution-server</resource>
     <enable-basic-auth>true</enable-basic-auth>
     <credential name="secret">03c2b267-7f64-4647-8566-
572be673f5fa</credential>
     <principal-attribute>preferred_username</principal-attribute>
  </secure-deployment>
</subsystem>

<system-properties>
  <property name="org.kie.server.sync.deploy" value="false"/>
</system-properties>
```

   Here,

   - **secure-deployment name**: Name of your application WAR file.
   - **realm**: Name of the realm that you created for the applications to use.
   - **realm-public-key**: The public key of the realm you created. You can find the key in the **Keys** tab in the **Realm settings** page of the realm you created in the RH-SSO Admin Console. If you do not provide a value for this public key, the server retrieves it automatically.
   - **auth-server-url**: The URL for the RH-SSO authentication server.
   - **resource**: The name for the server client that you created.
   - **enable-basic-auth**: The setting to enable basic authentication mechanism, so that the clients can use both token-based and basic authentication approaches to perform the requests.
   - **credential name**: The secret key of the server client you created. You can find the key in the **Credentials** tab on the **Clients** page of the RH-SSO Admin Console.
   - **principal-attribute**: The login name of the user. If you do not provide this value, your User Id is displayed in the application instead of your user name.

2. Save your configuration changes in **standalone.xml**.
3. Use the following command to restart the EAP server and run the Realtime Decision Server.

```
EXEC_SERVER_HOME/bin/standalone.sh -Dorg.kie.server.id=<ID> -
Dorg.kie.server.user=<USER> -Dorg.kie.server.pwd=<PWD> -
Dorg.kie.server.location=<LOCATION_URL> -Dorg.kie.server.controller=
<CONTROLLER_URL> -Dorg.kie.server.controller.user=<CONTROLLER_USER>
-Dorg.kie.server.controller.pwd=<CONTOLLER_PASSWORD>
```

Here is an example:

```
EXEC_SERVER_HOME/bin/standalone.sh -Dorg.kie.server.id=kieserver1 -
Dorg.kie.server.user=kieserver -Dorg.kie.server.pwd=password -
Dorg.kie.server.location=http://localhost:8080/kie-
server/services/rest/server -
Dorg.kie.server.controller=http://localhost:8080/business-
central/rest/controller -
Dorg.kie.server.controller.user=kiecontroller -
Dorg.kie.server.controller.pwd=password
```

4. Once the Realtime Decision Server is running, you can check the server status. In the following command, **kieserver** is a user name with the **kie-server** role and password **password**:

```
curl http://kieserver:password@localhost:8080/kie-
server/services/rest/server/
```

Token-based authorization is also supported for communication between Business Central and the Realtime Decision Server. You can use the complete token as a system property of your application server, instead of the user name and password, for your applications. However, you must ensure that the token does not expire for the period of interaction between the applications, as it is not automatically refreshed. To get the token, see Section 9.3.2, “Token-Based Authentication”.

For the Business Central to manage the Realtime Decision Server using the tokens:

- Set the **org.kie.server.token** property.
  In such case, do not set the **org.kie.server.user** and **org.kie.server.pwd** properties. The Business Central will then use the **Authorization: Bearer $TOKEN** authentication method.

If you want to use the REST API using the token-based authentication:

- Set the **org.kie.server.controller.token** property.
  In such case, do not set the **org.kie.server.controller.user** and **org.kie.server.controller.pwd** properties.

NOTE

As the Realtime Decision Server is unable to refresh the token, use a high-lifespan token. A token’s lifespan must not exceed January 19 2038. Check with your security best practices to see whether this is a suitable solution for your environment.

9.3. THIRD-PARTY CLIENT AUTHENTICATION THROUGH RH-SSO

To use the different remote services provided by Business Central or by the Realtime Decision Server, your client, such as curl, wget, web browser, or a custom REST client, must authenticate through the RH-SSO server and have a valid token to perform the requests. To use the remote services, the

authenticated user must have assigned the following roles:

- **`rest-all`**: For using the Business Central remote services.
- **`kie-server`**: For using the Realtime Decision Server remote services.

Use the RH-SSO Admin Console to create these roles and assign them to the users that will consume the remote services.

Your client can authenticate through RH-SSO using one of these options:

- Basic authentication, if it is supported by the client.
- Token-based authentication.

9.3.1. Basic Authentication

If you have enabled the basic authentication in the RH-SSO client adapter configuration for both Business Central and Realtime Decision Server, you can avoid the token grant/refresh calls and call the services as shown in the examples below:

- For web based remote repositories endpoint:

```
 curl http://admin:password@localhost:8080/business-
central/rest/repositories
```

- For the Realtime Decision Server:

```
curl http://admin:password@localhost:8080/kie-
server/services/rest/server/
```

9.3.2. Token-Based Authentication

If you want to opt for a more secure option of authentication, you can consume the remote services from both Business Central and Realtime Decision Server using a granted token provided by RH-SSO.

Procedure: Obtaining and Using Token for Authorizing Remote Calls

1. In the RH-SSO Admin Console, click the **Clients** left menu item and click **Create** to create a new client.
   The **Add Client** page opens.
2. On the **Add Client** page, provide the required information to create a new client for your realm. For example:
   - **Client ID**: kie-remote
   - **Client protocol**: openid-connect
3. Click **Save** to save your changes.
4. Change the token settings in **Realm Settings**:
   a. In the RH-SSO Admin Console, click the **Realm Settings** left menu item.

b. Click the **Tokens** tab.

c. Change the value for **Access Token Lifespan** to **15** minutes.
This gives you enough time to get a token and invoke the service before it expires.

d. Click **Save** to save your changes.

5. Once a public client for your remote clients is created, you can now obtain the token by making an HTTP request to the RH-SSO server's token endpoint using:

```
RESULT=`curl --data "grant_type=password&client_id=kie-
remote&username=admin&password=password"
http://localhost:8180/auth/realms/demo/protocol/openid-
connect/token`
```

The user used in the command above is an RH-SSO user. For further information, see Section 9.1.3, "Adding a New User".

6. To view the token obtained from the RH-SSO server, use the following command:

```
TOKEN=`echo $RESULT | sed 's/.*access_token":"//g' | sed 's/".*//g'`
```

You can now use this token to authorize the remote calls. For example, if you want to check the internal Red Hat JBoss BRMS repositories, use the token as shown below:

```
curl -H "Authorization: bearer $TOKEN" http://localhost:8080/business-
central/rest/repositories
```

CHAPTER 10. INTEGRATION WITH SPRING

10.1. CONFIGURING RED HAT JBOSS BRMS WITH SPRING

The **jboss-brms-engine.zip** file contains the Spring module, which is called **kie-spring-*VERSION*-redhat-*MINOR_VERSION*.jar**.

You can configure the Spring modules:

As a Self Managed Process Engine

If you require a single runtime manager instance, use the **RuntimeManager** API. The **RuntimeManager** API synchronizes the process engine and task service internally.

As a Shared Task Service

If you require multiple runtime manager instances, use the jBPM services.

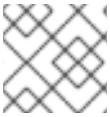
NOTE

Do not use the shared task service if you use Spring and your process is instantiated using the per process or per request runtime strategy.

10.1.1. Integrating Spring with Runtime Manager API

To integrate Spring with the Runtime Manager API, include the following factory beans:

- **org.kie.spring.factorybeans.RuntimeEnvironmentFactoryBean**
- **org.kie.spring.factorybeans.RuntimeManagerFactoryBean**
- **org.kie.spring.factorybeans.TaskServiceFactoryBean**



NOTE

TaskServiceFactoryBean is required only for shared task service.

RuntimeEnvironmentFactoryBean
RuntimeEnvironmentFactoryBean produces **RuntimeEnvironment** instances consumed by **RuntimeManager**.

You can create the following types of **RuntimeEnvironment** instances:

- **DEFAULT**: The default type.
- **EMPTY**: An empty environment.
- **DEFAULT_IN_MEMORY**: Same as **DEFAULT** with no persistence of the runtime engine.
- **DEFAULT_KJAR**: Same as **DEFAULT** with knowledge assets taken from kJAR and identified by **releaseID** or GAV (Group, Artifact, Version).
- **DEFAULT_KJAR_CL**: Built from class path that consists of a **kmodule.xml** descriptor.

Knowledge information is required for all the **RuntimeEnvironment** types. Provide one or more of the following:

- **knowledgeBase**
- **assets**
- **releaseId**
- **groupId**, **artifactId**, **version**

For the **DEFAULT**, **DEFAULT_KJAR**, **DEFAULT_KJAR_CL** types, configure persistence using **entity manager factory** or **transaction manager**.

RuntimeManagerFactoryBean
RuntimeManagerFactoryBean creates **RuntimeManager** instances based on **runtimeEnvironment**. You can create the following **runtimeEnvironment** instances:

- **SINGLETON** (default)
- **PER_REQUEST**
- **PER_PROCESS_INSTANCE**

Every **RuntimeManager** instance must have a unique ID. You can dispose of any **RuntimeManager** instance created by **RuntimeManagerFactoryBean** by calling the **close()** method.

TaskServiceFactoryBean
TaskServiceFactoryBean creates **TaskService** instance based on the given properties. Creates a single instance only.

Properties required:

- **entity manager factory**
- **transaction manager**

When using the **TaskServiceFactoryBean**, provide the Spring transaction manager. When using a shared entity manager from Spring, you can also provide **EntityManager** instance instead of entity manager factory.

Optional properties:

- **userGroupCallback**: **MVELUserGroupCallbackImpl** by default.
- **userInfo**: **DefaultUserInfo** by default.
- **listener**: List of **TaskLifeCycleEventListener** instances.

Sample RuntimeManager Configuration with Spring
To create a single runtime manager Spring configuration:

1. Configure the entity manager factory and the transaction manager, for example:

```
<bean id="jbpmEMF"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactor
```

```
yBean">
  <property name="persistenceUnitName"
value="org.jbpm.persistence.spring.jta"/>
</bean>

<bean id="btmConfig" factory-method="getConfiguration"
class="bitronix.tm.TransactionManagerServices"></bean>

<bean id="BitronixTransactionManager" factory-
method="getTransactionManager"
      class="bitronix.tm.TransactionManagerServices" depends-
on="btmConfig" destroy-method="shutdown" />

<bean id="jbpmTxManager"
class="org.springframework.transaction.jta.JtaTransactionManager">
  <property name="transactionManager"
ref="BitronixTransactionManager" />
  <property name="userTransaction" ref="BitronixTransactionManager"
/>
</bean>
```

This configuration provides:

- JTA transaction manager, backed by Bitronix for unit tests or servlet containers.
- The **`org.jbpm.persistence.spring.jta`** entity manager factory for persistence unit.

2. Configure resources you use, for example a business process:

```
<bean id="process" factory-method="newclass pathResource"
class="org.kie.internal.io.ResourceFactory">
  <constructor-arg>
    <value>jbpm/processes/sample.bpmn</value>
  </constructor-arg>
</bean>
```

The **`sample.bpmn`** process is included from the class path.

3. Configure **`RuntimeEnvironment`** using your entity manager, transaction manager, and resources:

```
<bean id="runtimeEnvironment"
class="org.kie.spring.factorybeans.RuntimeEnvironmentFactoryBean">
  <property name="type" value="DEFAULT"/>
  <property name="entityManagerFactory" ref="jbpmEMF"/>
  <property name="transactionManager" ref="jbpmTxManager"/>
  <property name="assets">
    <map>
      <entry key-ref="process"><util:constant static-
field="org.kie.api.io.ResourceType.BPMN2"/></entry>
    </map>
  </property>
</bean>
```

4. Create **`RuntimeManager`**:

```
<bean id="runtimeManager"
class="org.kie.spring.factorybeans.RuntimeManagerFactoryBean"
destroy-method="close">
  <property name="identifier" value="spring-rm"/>
  <property name="runtimeEnvironment" ref="runtimeEnvironment"/>
</bean>
```

An example of complete configuration:

```
<?xml version="1.0" encoding="UTF-8"?>
<beans xmlns="http://www.springframework.org/schema/beans"
       xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
       xmlns:util="http://www.springframework.org/schema/util"
       xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans-3.0.xsd
                           http://www.springframework.org/schema/util
http://www.springframework.org/schema/util/spring-util-3.0.xsd">

  <import resource="classpath:jbpm/configuration-template/assets.xml" />

  <bean id="jbpmEMF"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactoryBean"
>
    <property name="persistenceUnitName"
value="org.jbpm.persistence.spring.jta"/>
    <property name="persistenceXmlLocation"
value="classpath:jbpm/persistence-jta.xml"/>
  </bean>

  <bean id="btmConfig" factory-method="getConfiguration"
class="bitronix.tm.TransactionManagerServices"/>

  <bean id="BitronixTransactionManager" factory-
method="getTransactionManager"
        class="bitronix.tm.TransactionManagerServices" depends-
on="btmConfig" destroy-method="shutdown" />

  <bean id="jbpmTxManager"
class="org.springframework.transaction.jta.JtaTransactionManager">
    <property name="transactionManager" ref="BitronixTransactionManager"
/>
    <property name="userTransaction" ref="BitronixTransactionManager" />
  </bean>

  <bean id="runtimeEnvironment"
class="org.kie.spring.factorybeans.RuntimeEnvironmentFactoryBean">
    <property name="type" value="DEFAULT"/>
    <property name="entityManagerFactory" ref="jbpmEMF"/>
    <property name="transactionManager" ref="jbpmTxManager"/>
    <property name="assets" ref="assets"/>
  </bean>

  <bean id="logService" class="org.jbpm.process.audit.JPAAuditLogService"
depends-on="runtimeEnvironment">
    <constructor-arg value="#{runtimeEnvironment.environment}" />
    <constructor-arg value="STANDALONE_JTA" />
```

```
  </bean>
</beans>
```

10.1.2. Spring and jBPM Services

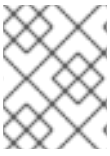
If you require multiple runtime managers, you can use jBPM services directly in your application. Due to the dynamic nature of jBPM services, processes and other assets can be added and removed without restarting your application.

To configure jBPM services:

1. Add the **kie-spring** Maven dependency into your **pom.xml**:

```
<dependencies>
  ...
  <dependency>
      <groupId>org.kie</groupId>
      <artifactId>kie-spring</artifactId>
      <version>6.5.0.Final-redhat-2</version>
    </dependency>
    ...
</dependencies>
```

   For the current Maven artifact version, see chapter Supported Component Versions of the *Red Hat JBoss BRMS Installation Guide*.



NOTE

Depending on your implementation, other dependencies may be necessary, for example **spring-security**.

1. Implement the **IdentityProvider** interface:

```
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

import org.kie.internal.identity.IdentityProvider;
import org.springframework.security.core.Authentication;
import org.springframework.security.core.GrantedAuthority;
import
org.springframework.security.core.context.SecurityContextHolder;

public class SpringSecurityIdentityProvider implements
IdentityProvider {

 public String getName() {

  Authentication auth =
SecurityContextHolder.getContext().getAuthentication();
  if (auth != null && auth.isAuthenticated()) {
   return auth.getName();
  }
  return "system";
```

```
 }

 public List<String> getRoles() {
  Authentication auth =
SecurityContextHolder.getContext().getAuthentication();
  if (auth != null && auth.isAuthenticated()) {
   List<String> roles = new ArrayList<String>();

   for (GrantedAuthority ga : auth.getAuthorities()) {
    roles.add(ga.getAuthority());
   }

   return roles;
  }

  return Collections.emptyList();
 }

 public boolean hasRole(String role) {
  return false;
 }

}
```

To configure jBPM services in a Spring application:

1. Configure the transaction manager:

```
<context:annotation-config />
<tx:annotation-driven />
<tx:jta-transaction-manager />

<bean id="transactionManager"
class="org.springframework.transaction.jta.JtaTransactionManager" />
```

2. Configure JPA and persistence:

```
<bean id="entityManagerFactory"
class="org.springframework.orm.jpa.LocalContainerEntityManagerFactor
yBean" depends-on="transactionManager">
   <property name="persistenceXmlLocation" value="classpath:/META-
INF/jbpm-persistence.xml" />
</bean>
```

3. Configure security providers:

```
<util:properties id="roleProperties"
location="classpath:/roles.properties" />

<bean id="userGroupCallback"
class="org.jbpm.services.task.identity.JBossUserGroupCallbackImpl">
  <constructor-arg name="userGroups"
ref="roleProperties"></constructor-arg>
</bean>
```

```
<bean id="identityProvider"
class="org.jbpm.spring.SpringSecurityIdentityProvider"/>
```

4. Configure the runtime manager factory:

```
<bean id="runtimeManagerFactory"
class="org.kie.spring.manager.SpringRuntimeManagerFactoryImpl">
  <property name="transactionManager" ref="transactionManager"/>
  <property name="userGroupCallback" ref="userGroupCallback"/>
</bean>

<bean id="transactionCmdService"
class="org.jbpm.shared.services.impl.TransactionalCommandService">
  <constructor-arg name="emf"
ref="entityManagerFactory"></constructor-arg>
</bean>

<bean id="taskService"
class="org.kie.spring.factorybeans.TaskServiceFactoryBean" destroy-
method="close">
  <property name="entityManagerFactory" ref="entityManagerFactory"/>
  <property name="transactionManager" ref="transactionManager"/>
  <property name="userGroupCallback" ref="userGroupCallback"/>
  <property name="listeners">
    <list>
      <bean
class="org.jbpm.services.task.audit.JPATaskLifeCycleEventListener">
        <constructor-arg value="true"/>
      </bean>
    </list>
  </property>
</bean>
```

The runtime manager factory is Spring context aware and can interact with Spring containers.

5. Configure jBPM services as Spring beans:

```
<!-- definition service -->
<bean id="definitionService"
class="org.jbpm.kie.services.impl.bpmn2.BPMN2DataServiceImpl"/>

<!-- runtime data service -->
<bean id="runtimeDataService"
class="org.jbpm.kie.services.impl.RuntimeDataServiceImpl">
  <property name="commandService" ref="transactionCmdService"/>
  <property name="identityProvider" ref="identityProvider"/>
  <property name="taskService" ref="taskService"/>
</bean>

<!-- -- deployment service -->
<bean id="deploymentService"
class="org.jbpm.kie.services.impl.KModuleDeploymentService" depends-
on="entityManagerFactory" init-method="onInit">
  <property name="bpmn2Service" ref="definitionService"/>
  <property name="emf" ref="entityManagerFactory"/>
```

```
  <property name="managerFactory" ref="runtimeManagerFactory"/>
  <property name="identityProvider" ref="identityProvider"/>
  <property name="runtimeDataService" ref="runtimeDataService"/>
</bean>

<!-- process service -->
<bean id="processService"
class="org.jbpm.kie.services.impl.ProcessServiceImpl" depends-
on="deploymentService">
  <property name="dataService" ref="runtimeDataService"/>
  <property name="deploymentService" ref="deploymentService"/>
</bean>

<!-- user task service -->
<bean id="userTaskService"
class="org.jbpm.kie.services.impl.UserTaskServiceImpl" depends-
on="deploymentService">
  <property name="dataService" ref="runtimeDataService"/>
  <property name="deploymentService" ref="deploymentService"/>
</bean>

<!-- register runtime data service as listener on deployment service
so it can receive notification about deployed and undeployed units -
->
<bean id="data"
class="org.springframework.beans.factory.config.MethodInvokingFactor
yBean" depends-on="deploymentService">
  <property name="targetObject" ref="deploymentService"></property>
  <property
name="targetMethod"><value>addListener</value></property>
  <property name="arguments">
  <list>
      <ref bean="runtimeDataService"/>
  </list>
  </property>
</bean>
```

CHAPTER 11. INTEGRATION WITH ARIES BLUEPRINT

This chapter explains the integration elements of Red Hat JBoss BRMS specific to Apache Aries Blueprint.

11.1. KIE NAMESPACE

KieModule

The **<kie:kmodule>** element defines a collection of a KieBase and its associated KieSessions.

Attribute	Description
id	The name to which other beans refer. Blueprint ID semantics applies. This attribute is required.

Possible children:

- **kie:kbase**

KieBase

The **<kie:kbase>** element has the following attributes:

Attribute	Description
name	The name of the KieBase. This attribute is required.
packages	A comma-separated list of the resource packages to be included in the KieBase.
includes	KieBase names to be included. All resources from the corresponding KieBases are included in the parent KieBase.
default	A Boolean. Sets the **kbase** as default. Set to **false** by default.
scope	Possible values: **prototype** or **singleton**. Set to **singleton** by default.
eventProcessingMode	Event Processing Mode. Possible values: **STREAM** or **CLOUD**.
equalsBehavior	Possible values: **IDENTITY** or **EQUALITY**.

Possible children:

- **kie:ksession**

The **kmodule** element can contain multiple **kbase** elements.

Example 11.1. kbase Definition Example

```
<kie:kmodule id="sample_module">
   <kie:kbase name="kbase1" packages="org.drools.blueprint.sample">
```

```
        ...
    </kie:kbase>
</kie:kmodule>
```

When you define a **kbase** or a **ksession**, you can set the bean scope:

- Set **scope** to **prototype** to instantiate a new bean instance every time the bean is called.
- Set **scope** to **singleton** to use the same bean instance every time the bean is called.

KieSession

The **<kie:ksession>** element defines both stateful and stateless KieSessions. It has the following parameters:

Attribute	Description
name	The name of the KieSession. This attribute is required.
type	Possible values: **stateful** or **stateless**. Set to **stateful** by default.
default	A Boolean. Sets the **ksession** as default. Set to **false** by default.
scope	Possible values: **prototype** or **singleton**. Set to **singleton** by default.
clockType	Possible values: **REALTIME** or **PSEUDO**.
listeners-ref	Specifies the reference to the event listeners group. For more information, see the section called “Defining a Group of Listeners”.

Example 11.2. ksession definition example

```
<kie:kmodule id="sample-kmodule">
  <kie:kbase name="drl_kiesample3" packages="drl_kiesample3">
    <kie:ksession name="ksession1" type="stateless"/>
    <kie:ksession name="ksession2"/>
  </kie:kbase>
</kie:kmodule>
```

Kie:ReleaseId

The **kie:releaseId** element represents Maven GAV (Group ID, Artifact ID, and Version). **kie:releaseId** requires the following properties:

Attribute	Description
id	The name to which other beans refer. Blueprint ID semantics applies.
groupId	Maven **groupId**.

Attribute	Description
artifactId	Maven **artifactId**.
version	Maven **version**.

Example 11.3. releaseId Definition Example

```
<kie:releaseId id="beanId" groupId="org.kie.blueprint"
               artifactId="named-artifactId" version="1.0.0-SNAPSHOT"/>
```

Kie:Import

Red Hat JBoss BRMS now supports **kie-aries-blueprint** importing KIE objects from KJARs. The **kie:import** element supports the following attributes:

Attribute	Description
releaseId	Reference to a bean ID. Standard Blueprint ID semantics applies.
enableScanner	Enable Scanner. This attribute is used only if **releaseId** is specified.
scannerInterval	Scanning Interval in milliseconds. This attribute is used only if **releaseId** is specified.

Red Hat JBoss BRMS supports two modes of importing KIE objects:

Global Import

The **kie:import** element uses the **KieServices.getKieClasspathContainer()** method to import KIE objects. For further information about KIE methods, see the KIE Api section of the *Red Hat JBoss BPM Suite Development Guide*.

Global Import

```
<kie:import />
```

Specific Import - ReleaseId

When using the **releaseId-ref** attribute on the import tag, only KIE objects identified by the referenced **releaseId** element are initialized and imported into the Blueprint context.

KIE Objects Import Using releaseId

```
<kie:import releaseId-ref="namedKieSession"/>
<kie:releaseId id="namedKieSession" groupId="org.drools"
               artifactId="named-kiesession" version="{revnumber}"/>
```

You can enable the KIE scanning feature, **enableScanner**, for KieBases imported with a specific

releaseId. This feature is currently not available for global imports.

Import KIE Objects using a releaseId - Enable Scanner

```
<kie:import releaseId-ref="namedKieSession"
            enableScanner="true" scannerInterval="1000"/>

<kie:releaseId id="namedKieSession" groupId="org.drools"
            artifactId="named-kiesession" version="{revnumber}"/>
```

If you define and enable a scanner, a **KieScanner** object is created with default values and inserted into the Blueprint container. You can get the **KieScanner** object from the Blueprint container using the **-scanner** suffix.

Retriving the KieScanner from a Blueprint Container

```
// the implicit name would be releaseId-scanner
KieScanner releaseIdScanner =
(KieScanner)container.getComponentInstance("namedKieSession-scanner");
releaseIdScanner.scanNow();
```

NOTE

kie-ci must be available on the classpath for the releaseId importing feature to work.

11.2. EVENT LISTENERS

Red Hat JBoss BRMS supports adding 3 types of listeners to KieSessions:

- **AgendaListener**
- **WorkingMemoryListener**
- **ProcessEventListener**

The **kie-aries-blueprint** module allows you to configure the listeners for KIE sessions using XML tags. The tags have identical names to the listener interfaces:

- **<kie:agendaEventListener>**
- **<kie:ruleRuntimeEventListener>**
- **<kie:processEventListener>**

The **kie-aries-blueprint** module allows you to define listeners as standalone listeners or as a group.

Defining Standalone Listeners

Standalone listeners support the following parameters:

Attribute	Description
ref	A reference to a bean.

Example 11.4. Listener Configuration Using bean:ref

```
<bean id="mock-agenda-listener" class="mocks.MockAgendaEventListener"/>
<bean id="mock-rr-listener" class="mocks.MockRuleRuntimeEventListener"/>
<bean id="mock-process-listener"
class="mocks.MockProcessEventListener"/>

<kie:kmodule id="listeners_kmodule">
  <kie:kbase name="drl_kiesample" packages="drl_kiesample">
    <kie:ksession name="ksession2">
      <kie:agendaEventListener ref="mock-agenda-listener"/>
      <kie:processEventListener ref="mock-process-listener"/>
      <kie:ruleRuntimeEventListener ref="mock-rr-listener"/>
    </kie:ksession>
  </kie:kbase>
</kie:kmodule>
```

Defining Multiple Listeners of One Type

You can also define multiple listeners of one type for a KIE session.

Example 11.5. Listener Configuration: Multiple Listeners of One Type.

```
<bean id="mock-agenda-listener1" class="mocks.MockAgendaEventListener"/>
<bean id="mock-agenda-listener2" class="mocks.MockAgendaEventListener"/>

<kie:kmodule id="listeners_module">
  <kie:kbase name="drl_kiesample" packages="drl_kiesample">
    <kie:ksession name="ksession1">
      <kie:agendaEventListener ref="mock-agenda-listener1"/>
      <kie:agendaEventListener ref="mock-agenda-listener2"/>
    </kie:ksession>
  </kie:kbase>
</kie:kmodule>
```

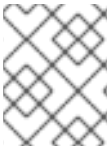
Defining a Group of Listeners

The **`kie-aries-blueprint`** module allows you to group listeners. This is useful when you define a set of listeners that you want to attach to multiple sessions, or when switching from testing to production use. The following attribute is required:

Attribute	Description
ID	Unique identifier

Possible children:

- **kie:agendaEventListener**
- **kie:ruleRuntimeEventListener**
- **kie:processEventListener**



NOTE

The declaration order does not matter. Only one declaration of each type is allowed in a group.

Example 11.6. Group of Listeners

```
<bean id="mock-agenda-listener" class="mocks.MockAgendaEventListener"/>
<bean id="mock-rr-listener" class="mocks.MockRuleRuntimeEventListener"/>
<bean id="mock-process-listener"
class="mocks.MockProcessEventListener"/>

<kie:kmodule id="listeners_module">
  <kie:kbase name="drl_kiesample" packages="drl_kiesample">
    <kie:ksession name="statelessWithGroupedListeners" type="stateless"
             listeners-ref="debugListeners"/>
  </kie:kbase>
</kie:kmodule>

  <kie:eventListeners id="debugListeners">
  <kie:agendaEventListener ref="mock-agenda-listener"/>
  <kie:processEventListener ref="mock-process-listener"/>
  <kie:ruleRuntimeEventListener ref="mock-rr-listener"/>
</kie:eventListeners>
```

11.3. LOGGERS

Red Hat JBoss BRMS supports the following loggers:

- **ConsoleLogger**
- **FileLogger**

The **kie-aries-blueprint** module allows you to configure the loggers using XML tags with identical names:

- **<kie:consoleLogger>**
- **<kie:fileLogger>**

Defining a Console Logger

The **<kie:consoleLogger/>** element has no attributes and must be present directly under a **<kie:ksession>** element.

Example 11.7. Defining a Console Logger

```
<kie:kmodule id="loggers_module">
  <kie:kbase name="drl_kiesample" packages="drl_kiesample">
    <kie:ksession name="ConsoleLogger-statefulSession" type="stateful">
      <kie:consoleLogger/>
    </kie:ksession>
  </kie:kbase>
</kie:kmodule>
```

Defining a File Logger

The **<kie:fileLogger/>** element supports the following attributes:

Attribute	Description
ID	Unique identifier. This attribute is required.
file	Path to the log file on the disk. This attribute is required.
threaded	Possible values: **true** or **false**. Set to **false** by default.
interval	An Integer. Specifies the interval for flushing the contents from memory to the disk.

Example 11.8. Defining a File Logger

```
<kie:kmodule id="loggers_module">
  <kie:kbase name="drl_kiesample" packages="drl_kiesample">
    <kie:ksession name="ConsoleLogger-statefulSession" type="stateful">
      <kie:fileLogger id="fl_logger" file="#{
systemProperties['java.io.tmpdir'] }/log1"/>
      <kie:fileLogger id="tfl_logger" file="#{
systemProperties['java.io.tmpdir'] }/log2"
                      threaded="true" interval="5"/>
    </kie:ksession>
  </kie:kbase>
</kie:kmodule>
```

Closing a FileLogger

It is recommended to close the **<kie:fileLogger>** logger to prevent memory leaks:

```
LoggerAdaptor adaptor = (LoggerAdaptor)
container.getComponentInstance("fl_logger");
adaptor.close();
```

Defining Batch Commands

The **<kie:batch>** element allows you to define a set of batch commands for a given KIE session. The **<kie:batch>** element has no attributes and must be placed under a **<kie:ksession>** element.

Supported Parameters for Initialization Batch Commands

- **`insert-object`**
  - **`ref`**: String. This parameter is optional.
  - **`Anonymous bean`**.
- **`set-global`**
  - **`identifier`**: String. This parameter is required.
  - **`reg`**: String. This parameter is optional.
  - **`Anonymous bean`**.
- **`fire-all-rules`**
  - **`max`**: Integer.
- **`fire-until-halt`**
- **`start-process`**
  - **`identifier`**: String. This parameter is required.
  - **`ref`**: String. This parameter is optional.
  - **`Anonymous bean`**.
- **`signal-event`**
  - **`ref`**: String. This parameter is optional.
  - **`event-type`**: String. This parameter is required.
  - **`process-instance-id`**: Integer. This parameter is optional.

Example 11.9. Batch Commands Example

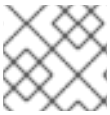
```
<kie:kmodule id="batch_commands_module">
  <kie:kbase name="drl_kiesample" packages="drl_kiesample">
    <kie:ksession name="ksessionForCommands" type="stateful">
      <kie:batch>
        <kie:insert-object ref="person2"/>
        <kie:set-global identifier="persons" ref="personsList"/>
        <kie:fire-all-rules max="10"/>
      </kie:batch>
    </kie:ksession>
  </kie:kbase>
</kie:kmodule>
```

CHAPTER 12. LOCALIZATION AND CUSTOMIZATION

12.1. AVAILABLE LANGUAGES

The Red Hat JBoss BRMS web user interface can be viewed in multiple languages:

- United States English (**en_US**)
- Spanish (**es_ES**)
- Japanese (**ja_JP**)
- Simplified Chinese (**zh_CN**)
- Traditional Chinese (**zh_TW**)
- Portuguese (**pt_BR**)
- French (**fr_CA**)
- German (**de_DE**)



NOTE

If a language is not specified, US English is used by default.

12.2. CHANGING LANGUAGE SETTINGS

Changing the User Interface Language in Business Central
By default, Business Central uses the system locale. If you need to change it, then append the required locale code at the end of the Business Central URL. For example, the following URL will set the locale to Portuguese (pt_BR).

http://*localhost*:8080/business-central/?locale=pt_BR

12.3. RUNNING THE JVM WITH UTF-8 ENCODING

Red Hat JBoss BRMS is designed to work with UTF-8 encoding. If a different encoding system is being used by the JVM, unexpected errors might occur.

To ensure UTF-8 is used by the JVM, use the JVM option **-Dfile.encoding=UTF-8**.

CHAPTER 13. MONITORING

13.1. JBOSS OPERATIONS NETWORK

A JBoss Operations Network plug-in can be used to monitor rules sessions for Red Hat JBoss BRMS.

Due to a limitation of passing the JVM monitoring arguments via the Maven command line, all **`com.sun.management.jmxremote.*`** parameters must be passed to the Red Hat JBoss BRMS application via the **`pom.xml`** configuration file.

See the Installation Guide of *Red Hat JBoss Operations Network* for installation instructions on the Red Hat JBoss ON server.

13.2. DOWNLOADING RED HAT JBOSS BRMS FOR JBOSS EAP

1. Go to the Red Hat Customer Portal and log in.
2. At the top of the page, click **DOWNLOADS**.
3. In the **Product Downloads** page that opens, click **Red Hat JBoss BRMS**.
4. From the **Version** drop-down menu, select version 6.4.0.
5. Select **Red Hat JBoss BRMS 6.4.0 Deployable for EAP 6** and then click **Download**.

13.3. INSTALLING THE JBOSS BRMS PLUG-IN INTO JBOSS ON

Red Hat JBoss BRMS plug-in for JBoss Operations Network can be installed by either copying the plug-in JAR files to the JBoss Operations Network plug-in directory or through the JBoss Operations Network GUI.

To copy the plug-in JAR files to the JBoss Operations Network plug-in directory, follow the following procedure:

Procedure: Copying the JBoss BRMS plug-in JAR files

1. Extract the JBoss BRMS plug-in pack archive to a temporary location. This creates a subdirectory with the name **`jon-plugin-pack-brms-bpms-3.3.0.GA`**. For example:

```
[root@server rhq-agent]# unzip jon-plugin-pack-brms-bpms-
3.3.0.GA.zip -d /tmp
```

2. Copy the extracted JBoss BRMS plug-in JAR files from the **`jon-plugin-pack-brms-bpms-3.3.0.GA/`** directory to the JBoss ON server plug-in directory. For example:

```
[root@server rhq-agent]# cp /tmp/jon-plugin-pack-brms-bpms-
3.3.0.GA/*.jar /opt/jon/jon-server-3.3.0.GA/plugins
```

3. Start the JBoss Operations Network server to update the JBoss BRMS plug-in.

To upload the JBoss BRMS plug-in through the JBoss Operations Network GUI, follow this procedure:

Procedure: Uploading the JBoss BRMS plug-in through GUI

1. Start the JBoss Operations Network Server and Log in to access the GUI.
2. In the top navigation of the GUI, open the **Administration** menu.
3. In the **Configuration** area on the left, select the **Agent Plugins** link.
4. At the bottom of the list of loaded agent plug-ins, click the **Upload a plugin** button and choose the BRMS plugin.
5. The JBoss BRMS plug-in for JBoss Operations Network is now uploaded.

13.4. MONITORING KIE BASES AND KIE SESSIONS

In order for JBoss Operations Network to monitor KieBases and KieSessions, MBeans must be enabled.

MBeans can be enabled either by passing the parameter **`-kie.mbeans = enabled`** or via the API:

```
KieBaseConfiguration kbconf =
KieServices.Factory.get().newKieBaseConfiguration();
    kbconf.setOption(MBeansOption.ENABLED);
```

NOTE

`Kie Services` have been implemented for JBoss BRMS 6; for JBoss BRMS 5, **`Drools Services`** was the naming convention used and it had different measurements on sessions. For example, **activation → match** renaming occurred in the updated version.

Please refer to the Resource Monitoring and Operations Reference guide of the *Red Hat JBoss Operations Network* for information on importing Kie Sessions into the Inventory View for monitoring purposes.

APPENDIX A. CONFIGURATION PROPERTIES

This chapter contains all public system properties and KIE environment entries that you can use to configure your deployment of Red Hat JBoss BRMS.

A.1. SYSTEM PROPERTIES

System properties configure the entire JVM Red Hat JBoss BRMS runs on. You can either provide them at runtime, or set them in the configuration XML file for your deployment.

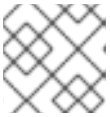
A.1.1. Runtime Configuration

To set a particular property at runtime, add a flag **`-D`** with the property's name and value when starting the server. You can list multiple such properties at the same time.

Configuring System Properties at Runtime (Standalone deployments)

1. Navigate to ***`SERVER_HOME`*`/bin`**.
2. Run the server with the desired flags:

```
./standalone.sh -Dorg.kie.custom.property=value -
Dorg.uberfire.switch=false
```



NOTE

On Windows, run **`standalone.bat`** instead of **`standalone.sh`**.

A.1.2. XML Configuration

To set a property in Red Hat JBoss BRMS configuration, add an entry under the **`<system-properties>`** section in the **`standalone.xml`** file:

```
<system-properties>
  <property name="org.kie.custom.property" value="random_value"/>
  <property name="org.uberfire.switch" value="false"/>
  ...
</system-properties>
```

When running Red Hat JBoss BRMS in domain mode, add the entries in the **`<system-properties>`** element of the appropriate node in the **`host.xml`** file.

A.1.3. List of System Properties

This section contains the alphabetically sorted list of all recognized system properties in Red Hat JBoss BRMS 6.4.

Red Hat JBoss BRMS System Properties

btm.root

Root directory for Bitronix Transaction Manager. The discovery of configuration and other files starts in this location.

Values	Default
String	N/A

drools.propertySpecific

Sets property reactivity behavior of the Red Hat JBoss BRMS engine. Options are following:

- **`DISABLED`**: Property reactivity turned off.
- **`ALLOWED`**: Property reactivity allowed.
- **`ALWAYS`**: Property reactivity always on.

Values	Default
`DISABLED`, **`ALLOWED`**, or **`ALWAYS`**	**`ALLOWED`**

drools.ruleEngine

Specifies which algorithm the Red Hat JBoss BRMS rule engine should use.

Values	Default
`phreak` or **`reteoo`**	**`phreak`**

drools.sequential

Enables sequential mode for stateless sessions.

Values	Default
`true` or **`false`**	**`false`**

drools.sequential.agenda

Selects static or dynamic agenda with sequential mode.

Values	Default
`static` or **`dynamic`**	**`static`** for standard mode, **`dynamic`** for sequential mode

jboss.node.name

A node name unique in a Red Hat JBoss BRMS cluster.

Values	Default
String	N/A

kie.maven.settings.custom

The location of a custom **`settings.xml`** file for Maven configuration.

Values	Default
Path	N/A

kie.server.jms.queues.response

The JNDI name of response queue for JMS.

Values	Default
String	**`queue/KIE.SERVER.RESPONSE`**

org.drools.server.ext.disabled

When set to **`true`**, disables the BRM support (for example rules support).

Values	Default
`true` or **`false`**	**`false`**

org.drools.server.filter.classes

When set to **`true`**, the Drools Realtime Decision Server extension accepts custom classes annotated by **`XmlRootElement`** or **`Remotable`** annotations only.

Values	Default
`true` or **`false`**	**`false`**

org.guvnor.inbox.disabled

When set to **`true`**, this system property disables the Inbox feature and the Inbox mechanism of tracking changes to files is turned off. Doing this enables faster response to file system operations.

Values	Default
`true` or **`false`**	**`false`**

org.guvnor.m2repo.dir

The location where Maven artifacts are stored. When you build and deploy a project, it is stored in this directory. Change the setting, for example, to allow easier backup of your maven repository.

Values	Default
Path	***`EAP_HOME`*`/repositories/kie`**

org.guvnor.project.gav.check.disabled

Disables a duplicate **`GroupId`**, **`ArtifactId`**, and **`Version`** (GAV) detection. When you build and deploy a project, Business Central scans the Maven repository for an artifact with the same GAV values. If set to **`true`**, Business Central silently overrides any previous project. If set to **`false`**, the user is required to confirm overriding the old project.

Values	Default
`true` or **`false`**	**`false`**

org.kie.auto.deploy.enabled

When enabled, issuing a Build & Deploy operation in Business Central always deploys to runtime.

Values	Default
`true` or **`false`**	**`true`**

org.kie.build.disable-project-explorer

Disables automatic build of the selected project in Project Explorer.

Values	Default
`true` or **`false`**	**`false`**

org.kie.demo

Enables external cloning of a demo application from GitHub. This System Property takes precedence over **`org.kie.example`**.

Values	Default
`true` or **`false`**	**`true`**

org.kie.example

When set to **`true`**, creates an example organization unit and repository. This system property allows you to create projects and assets without creating your custom organization unit and repository. It is useful, for example, to simplify the getting started experience.

Values	Default
`true` or **`false`**	**`false`**

org.kie.example.repositories

Sets the path to the directory containing example repositories. If you set this system property, repositories in the specified directory are automatically cloned into Business Central during startup. This property overrides **`org.kie.example`** and **`org.kie.demo`**.

WARNING

You must download the example repositories from the Customer Portal and extract them to this directory before setting this system property.

Values	Default
Path	N/A

org.kie.executor.disabled

Disables the Red Hat JBoss BRMS executor.

Values	Default
`true` or `false`	`false`

org.kie.executor.initial.delay

The initial delay before the Red Hat JBoss BRMS executor starts a job, in milliseconds.

Values	Default
Number (`Integer`)	100

org.kie.executor.interval

The time between the moment the Red Hat JBoss BRMS executor finishes a job and the moment it starts a new one, in a time unit specified in `org.kie.executor.timeunit`.

Values	Default
Number (`Integer`)	3

org.kie.executor.pool.size

The number of threads used by the Red Hat JBoss BRMS executor.

Values	Default
Number (`Integer`)	1

org.kie.executor.retry.count

The number of retries the Red Hat JBoss BRMS executor attempts on a failed job.

Values	Default
Number (**`Integer`**)	3

org.kie.executor.timeunit

The time unit in which the **`org.kie.executor.interval`** is specified.

Values	Default
A **`java.util.concurrent.TimeUnit`** constant	**`SECONDS`**

org.kie.git.deployments.enabled

When enabled, Red Hat JBoss BRMS uses a Git repository for storing deployments instead of a database.

Values	Default
`true` or **`false`**	**`false`**

org.kie.mail.session

The JNDI name of the mail session as registered in the application server, for use by **`EmailWorkItemHandler`**.

Values	Default
String	**`mail/jbpmMailSession`**

org.kie.server.controller

A comma-separated list of URLs to controller REST endpoints, for example **`http://localhost:8080/business-central/rest/controller`**. This property is required when using a controller.

Values	Default
Comma-separated list	N/A

org.kie.server.controller.connect

The waiting time between repeated attempts to connect Realtime Decision Server to the controller when Realtime Decision Server starts up, in milliseconds.

Values	Default
Number (**`Long`**)	**`10000`**

org.kie.server.controller.pwd

The password to connect to the controller REST API. This property is required when using a controller.

Values	Default
String	`kieserver1!`

org.kie.server.controller.token

This property allows you to use a token-based authentication between the KIE server and the controller instead of the basic user name/password authentication. The KIE server sends the token as a parameter in the request header. Note that long-lived access tokens are required as the tokens are not refreshed.

Values	Default
String	N/A

org.kie.server.controller.user

The user name to connect to the controller REST API. This property is required when using a controller.

Values	Default
String	`kieserver`

org.kie.server.domain

The JAAS `LoginContext` domain used to authenticate users when using JMS.

Values	Default
String	N/A

org.kie.server.id

An arbitrary ID to be assigned to this server. If a remote controller is configured, this is the ID under which the server will connect to the controller to fetch the KIE container configurations. If not provided, the ID is automatically generated.

Values	Default
String	N/A

org.kie.server.location

The URL of the Realtime Decision Server instance used by the controller to call back on this server, for example: `http://localhost:8230/kie-server/services/rest/server`. This property is required when using a controller.

Values	Default
URL	N/A

org.kie.server.pwd

The password used to connect with the KIE server from the controller, required when running in managed mode. You must set this property in Business Central system properties, and it is required when using a controller.

Values	Default
String	`kieserver1!`

org.kie.server.repo

The location where Realtime Decision Server state files will be stored.

Values	Default
Path	`.`

org.kie.server.sync.deploy

Instructs the KIE server to hold the deployment until the controller provides the containers deployment configuration. This property affects only the KIE servers running in managed mode. The options are as follows:

- **`false`**; the connection to the controller is asynchronous. The application starts, connects to the controller and once successful, deploys the containers. The application accepts requests even before the containers are available.
- **`true`**; the deployment of the KIE server application joins the controller connection thread with the main deployment and awaits its completion.
  This option can lead to a potential deadlock in case more applications are on the same server instance. It is strongly recommended to use only one application (the KIE server) on one server instance.

Values	Default
`true` or **`false`**	**`false`**

org.kie.server.token

This property allows you to use a token-based authentication between the controller and the KIE server instead of the basic user name/password authentication. The controller sends the token as a parameter in the request header. Note that long-lived access tokens are required as the tokens are not refreshed.

Values	Default
String	N/A

org.kie.server.user

The user name used to connect with the KIE server from the controller, required when running in managed mode. This property need to be set in Business Central system properties and is required when using a controller.

Values	Default
String	`kieserver`

org.kie.tx.lock.enabled

When enabled, Red Hat JBoss BRMS uses an interceptor that locks the `KieSession` to a single thread for the duration of a transaction, which prevents concurrency errors in Container Managed Transaction (CMT) environments.

Values	Default
`true` or `false`	`false`

org.kie.verification.disable-dtable-realtime-verification

Disables Business Central's decision table verification and validation feature.

Values	Default
`true` or `false`	`false`

org.optaplanner.server.ext.disabled

When set to `true`, disables the BRP support (for example planner support).

Values	Default
`true` or `false`	`false`

org.uberfire.cluster.autostart

Delays VFS clustering until the application is fully initialized to avoid conflicts when all cluster members create local clones.

Values	Default
`true` or `false`	`false`

org.uberfire.cluster.id

The name of the Helix cluster, for example: **`kie-cluster`**. You must set this property to the same value as defined in the Helix Controller.

Values	Default
String	N/A

org.uberfire.cluster.local.id

The unique ID of the Helix cluster node. Note that ':' is replaced with '_', for example **`node1_12345`**.

Values	Default
String	N/A

org.uberfire.cluster.vfs.lock

The name of the resource defined on the Helix cluster, for example: **`kie-vfs`**.

Values	Default
String	N/A

org.uberfire.cluster.zk

The location of the Zookeeper servers.

Values	Default
String of the form **`host1:port1`**,**`host2:port2`**,**`host3:port3`**, …	N/A

org.uberfire.domain

The security domain name for Business Central. For more information about security domains, see chapter Use a Security Domain in Your Application of the *Red Hat JBoss EAP Security Guide*.

Values	Default
String	**`ApplicationRealm`**

org.uberfire.metadata.index.dir

The location of the **`.index`** directory, which Apache Lucene uses when indexing and searching.

Values	Default
Path	Current working directory

org.uberfire.nio.git.daemon.enabled

Enables the Git daemon.

Values	Default
`true` or `false`	`true`

org.uberfire.nio.git.daemon.host

If the Git daemon is enabled, it uses this property as the localhost identifier.

Values	Default
URL	`localhost`

org.uberfire.nio.git.daemon.port

If the Git daemon is enabled, it uses this property as the port number.

Values	Default
Port number	`9418`

org.uberfire.nio.git.dir

The location of the directory `.niogit`. Change the value for example for backup purposes.

Values	Default
Path	Current working directory

org.uberfire.nio.git.hooks

The location where default Git hook files are stored. These files will be copied to newly created Git repositories.

Values	Default
Path	N/A

org.uberfire.nio.git.ssh.cert.dir

The location of the directory `.security`. Local certificates are stored here.

Values	Default
Path	Current working directory

org.uberfire.nio.git.ssh.enabled

Enables the SSH daemon.

Values	Default
`true` or **`false`**	**`true`**

org.uberfire.nio.git.ssh.host

If the SSH daemon is enabled, it uses this property as the localhost identifier.

Values	Default
URL	**`localhost`**

org.uberfire.nio.git.ssh.passphrase

The passphrase to access your operating system's public keystore when cloning Git repositories with scp-style URLs, for example **`git@github.com:user/repository.git`**.

Values	Default
String	N/A

org.uberfire.nio.git.ssh.port

If the SSH daemon is enabled, it uses this property as the port number.

Values	Default
Port number	**`8001`**

org.uberfire.secure.alg

The crypto algorithm used by password encryption.

Values	Default
String	**`PBEWithMD5AndDES`**

org.uberfire.secure.key

A secret password used by password encryption.

Values	Default
String	**`org.uberfire.admin`**

org.uberfire.sys.repo.monitor.disabled

Disables the configuration monitor.

WARNING

Do not use unless you are certain what you are doing.

Values	Default
`true` or **`false`**	**`false`**

A.2. ENVIRONMENT PROPERTIES

As opposed to system properties, environment properties are passed to an individual **`KieSession`**, allowing you to control its behaviour independently on the rest of the deployment.

The properties available to you are the constants of the class **`org.kie.api.runtime.EnvironmentName`**.

A.2.1. Configuration

To set the environment properties on a **`KieSession`**, you can create a new session with an instance of the **`Environment`** or **`RuntimeEnvironment`** interface:

Setting Environment Property using `Environment` interface

```
Environment env = EnvironmentFactory.newEnvironment();
env.set(EnvironmentName.SAMPLE_PROPERTY, true);
kbase.newKieSession(null, env);
```

Setting Environment Property using `RuntimeEnvironment` interface

```
RuntimeEnvironment environment = RuntimeEnvironmentBuilder.Factory.get()
.newDefaultBuilder()
....

environment.getEnvironment().set(EnvironmentName.SAMPLE_PROPERTY, true);
singletonManager =
RuntimeManagerFactory.Factory.get().newSingletonRuntimeManager(environment
);
```

A.2.2. List of Environment Properties

This section contains the alphabetically sorted list of all recognized environment properties in Red Hat JBoss BRMS 6.4.

Red Hat JBoss BRMS Environment Properties

CALENDARS

This property is not used.

DATE_FORMATS

This property is not used.

GLOBALS

Values	Default
Any object declared in DRL or BPMN	N/A

OBJECT_MARSHALLING_STRATEGIES

Enable use of pluggable variable persistence strategies. Allows storing variables in different data stores.

Values	Default
An instance of **`org.kie.api.marshalling.ObjectMarshallingStrategy`**	Dependent on the engine configuration. If no other strategy is available, **`org.drools.core.marshalling.impl.SerializablePlaceholderResolvingStrategy`** is used.

PERSISTENCE_CONTEXT_MANAGER

The **`ProcessPersistenceContextManager`** instance used for process persistence.

Values	Default
An instance of **`org.jbpm.persistence.ProcessPersistenceContextManager`**	An instance of **`org.jbpm.persistence.JpaProcessPersistenceContextManager`**

TRANSACTION

Optional property if **`UserTransaction`** can not be obtained using JNDI lookup.

Values	Default
An instance of **`UserTransaction`**	**`null`**

TRANSACTION_MANAGER

Get the **`TransactionManager`** instance from the environment or context. For example:

```
info = context.get(EnvironmentName.TRANSACTION_MANAGER)
```

Values	Default
An instance of `TransactionManager`	Depends on your configuration

TRANSACTION_SYNCHRONIZATION_REGISTRY

Allows access to and control of the active transaction. Used by Red Hat JBoss BRMS to efficiently manage persistence.

Values	Default
An instance of `TransactionSynchronizationRegistry`	Taken from the environment—usually JNDI lookup in JTA environments

APPENDIX B. VERSIONING INFORMATION

Documentation last updated on: Wednesday December 26, 2018.