Red Hat JBoss BPM Suite 6.1

Administration And Configuration Guide

The Administration and Configuration Guide for Red Hat JBoss BPM Suite

Last Updated: 2017-11-13

Red Hat JBoss BPM Suite 6.1 Administration And Configuration Guide

The Administration and Configuration Guide for Red Hat JBoss BPM Suite

Kanchan Desai
kadesai@redhat.com

Doug Hoffman

Eva Kopalova

Red Hat Content Services

Gemma Sheldon
Red Hat Engineering Content Services
gsheldon@redhat.com

Joshua Wulf
jwulf@redhat.com

Abstract

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

Table of Contents

PART I. INTRODUCTION

CHAPTER 1. BUSINESS PROCESS MODEL AND NOTATION

Business Process Model and Notation (BPMN) is a standard notation for business process modeling. It aspires to link the gap between business analysts and programmers by providing a workflow language that can be clearly understood by both.

1.1. COMPONENTS

Red Hat JBoss BPM Suite integrates multiple components to support business processes throughout their entire life cycle and to provide process management features and tools for business analysts, developers, and business users. The product can be deployed on various JEE-compliant servers; the recommended option is Red Hat JBoss Enterprise Application Platform 6.

Red Hat JBoss BPM Suite consists of the following main components:

- **`Execution Engine`** - provides the runtime environment for Processes and Business Rules. It encompasses a workflow library that can be embedded into a user web application. Runtime manager is the root object and contains the following components:
    - **`Runtime Engine`** - implements the core behavior of the computer language and it is provided by the runtime manager.
        - **`Process Engine`** - is the environment for business process model execution.
        - **`Task Service`** - handles human task lifecycles.
    - **`Rule Engine`** - can be used with the process engine or on its own.
        - **`Rules Evaluation`** - executes business rules on the provided set of facts.
        - **`Complex Event Processing`** - applies business rules on incoming stream of events.
- **`Business Central`** - a web-based application that accommodates tooling for asset creation, management, and monitoring by providing an integrated web environment.
    - **`Asset Repository`** - is the central sharing location (Knowledge Store) for business assets, processes, rules, forms, etc. Users access this repository through the Project Explorer view of Business Central via **Authoring** → **Project Authoring**. By default, the product initializes a local GIT repository as its Asset Repository. However, other repositories may be added or removed as necessary.
    - **`Artifact Repository`** - is a Maven based repository for storage of project jar artifacts.
    - **`Execution Server`** - provides an execution environment for business process instances and tasks.
    - **`Business Activity Monitor`** - provides customizable view on business performance.

NOTE

Red Hat JBoss BRMS comes with its own Business Central application that is a subset of the Business Central application in Red Hat JBoss BPM Suite.

1.2. PROJECT

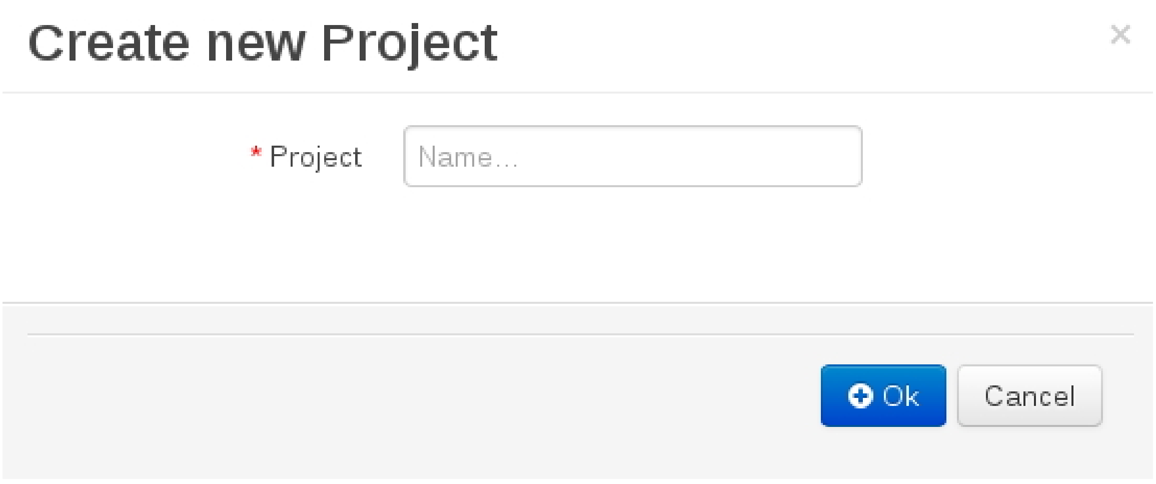
A project is a container for asset packages (business processes, rules, work definitions, decision tables, fact models, data models, and DSLs) that lives in the Knowledge Repository. It is this container that defines the properties of the KIE Base and KIE Session that are applied to its content. In the GUI, you can edit these entities in the Project Editor.

As a project is a Maven project, it contains the Project Object Model file (**`pom.xml`**) with information on how to build the output artifact. It also contains the Module Descriptor file, **`kmodule.xml`**, that contains the KIE Base and KIE Session configuration for the assets in the project.

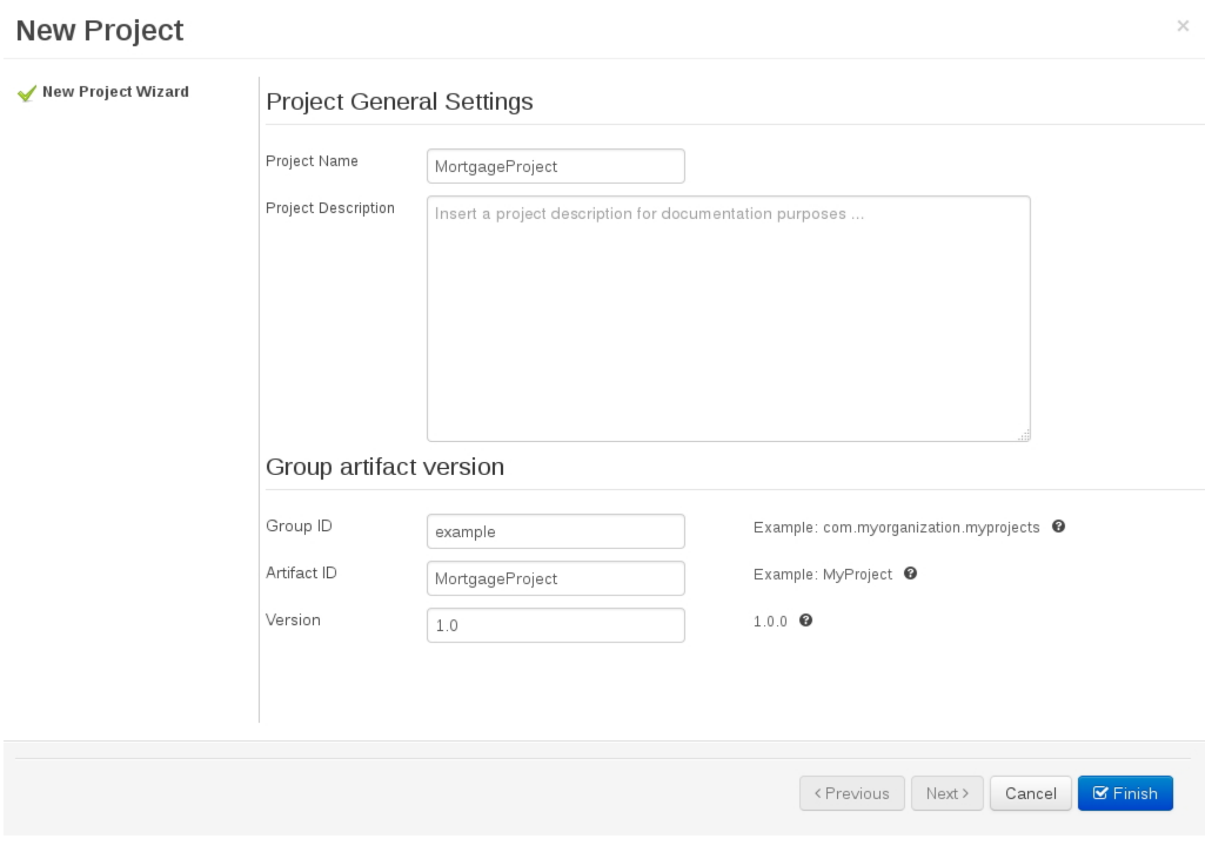
1.3. CREATING A PROJECT

To create a project, do the following:

1. Open the **`Project Authoring`** perspective: on the main menu, click **Authoring** → **Project Authoring**.
2. In the **`Project Explorer`**, select the organizational unit and the repository where you want to create the project.
3. In the perspective menu, go to **New Item** → **Project**.
4. In the **`Create new Project`** dialog window, define the project details:
    a. In the **`Project`** text box, enter the project name.



5. The explorer refreshes to show a **`New Project Wizard`** pop-up window.


New Project

New Project Wizard

Project General Settings

Project Name: MortgageProject

Project Description: Insert a project description for documentation purposes ...

Group artifact version

Group ID: example — Example: com.myorganization.myprojects

Artifact ID: MortgageProject — Example: MyProject

Version: 1.0 — 1.0.0

6. Define the **`Project General Settings`** and **`Group artifact version`** details for this new project. These parameters are stored inside the **`pom.xml`** Maven configuration file.

   - **`Project Name`**: The name for the project; for example **`MortgageProject`**
   - **`Project Description`**: The description of the project which may be useful for the project documentation purpose.
   - **`Group ID`**: group ID of the project; for example **`org.mycompany.commons`**
   - **`Artifact ID`**: artifact ID unique in the group; for example **`myframework`**. Avoid using a space or any special character that might lead to an invalid name.
   - **`Version ID`**: version of the project; for example **`2.1.1`**

   The **`Project Screen`** view is updated with the new project details as defined in the **`pom.xml`** file. Note, that you can switch between project descriptor files in the drop down-box with **`Project Settings`** and **`Knowledge Base Setting`**, and edit their contents.

1.4. ADDING DEPENDENCIES

To add dependencies to your project, do the following:

1. Open the Project Editor for the given project:

   a. In the **`Project Explorer`** view of the **`Project Authoring`** perspective, open the project directory.

   b. Click on the button to open the project view.

2. In the **`Project Screen`** view, select in the **`Project Settings`** drop-down box the **`Dependencies`** item.
3. On the updated **`Project Screen`**, click the **`Add`** button to add a maven dependency or click the **`Add from repository`** button to add a dependency from the Knowledge Store (Artifact repository):
   - When adding a maven dependency, a user has to define the **`Group ID`**, **`Artifact ID`** and the **`Version ID`** in the new row which is created in the dependency table.
   - When adding a dependency from the Knowledge Store, select the dependency in the displayed dialog box: the dependency will be added to the dependency table.
4. To apply the various changes, the dependencies must be saved.

WARNING

If working with modified artifacts, do not re-upload modified non-snapshot artifacts as Maven will not know these artifacts have been updated, and it will not work if it is deployed in this manner.

PART II. CONFIGURATION

CHAPTER 2. BUSINESS CENTRAL CONFIGURATION

As Business Central is a web application, any configuration settings are loaded from ***DEPLOY_DIRECTORY*/business-central.war/WEB-INF/web.xml** and the referenced files, and if deployed on Red Hat JBoss EAP 6, also in **jboss-web.xml** and **jboss-deployment-structure.xml**.

Note that the entire application can be run in different profiles (refer to the *Red Hat JBoss BPM Suite Installation Guide*).

2.1. ACCESS CONTROL

The access control mechanism includes authorization and authentication. In the unified environment of Red Hat JBoss BPM Suite, users are able to update the default user roles located within ***$JBOSS_HOME*/standalone/deployments/business-central.war/WEB-INF/classes/userinfo.properties**.

To grant a user access to JBoss BPM Suite, the user needs to have the respective role assigned:

- **admin**: administrates JBoss BPM Suite system and has full access rights to make any changes necessary including the ability to add and remove users from the system.
- **developer**: implements code required for processes to work and has access to everything except administration tasks.
- **analyst**: creates and designs processes and forms, instantiates the processes and deploys artifacts. This role is the similar to a developer, without access to asset repository and deployments.
- **user**: claims, performs, and invokes other actions (such as, escalation, rejection, etc.) on the assigned Tasks and has no access to authoring functions.
- **manager**: monitors the system and its statistics and only has access to the dashboard.
- **business user**: takes action on business tasks that are required for processes to continue forward. Works primarily with the task list.

If using Red Hat JBoss EAP, to create a user with particular roles, run the ***$JBOSS_HOME*/add-user.sh** script and create an Application User in the **ApplicationRealm** with the respectives roles.

Workbench Configuration

Within Red Hat JBoss BPM Suite, 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.

Authentication in Human Tasks

Every Task that needs to be executed is assigned to one or multiple roles or groups, so that any user with the given role or the given group assigned can claim the Task instance and execute it. Tasks can also be assigned to one or multiple users directly. JBoss BPM Suite uses the **UserGroupCallback** interface to assign tasks to user.

> **WARNING**
>
> A group for a Human Task must not be named after an existing user of the system. Doing so causes intermittent issues.

2.2. BUSINESS CENTRAL PROFILE CONFIGURATION

Red Hat JBoss BPM Suite 6 (or better) server is capable of starting the Business Central application in three different modes:

- Full profile - default profile that is active without additional configuration required (UI and remote services e.g. REST).
- Execution server profile - disables completely UI components of the application and allows only remote access e.g. via REST interface.
- UI server profile - disables remote services e.g REST and allows only UI access to the application.

To change the profile use the following configuration steps.

Procedure 2.1. Configuring Business Central Profiles

1. Select the desired **`web.xml`** inside **`$BPMS_HOME/standalone/deployments/business-central.war/WEB-INF/`**. The following files are provided.
   - **`web.xml`** (default) for full profile
   - **`web-exec-server.xml`** for execution server profile
   - **`web-ui-server.xml`** for UI server profile
2. To activate a profile other than the default full profile, the web-<PROFILE>.xml file must be renamed to **`web.xml`**. The following steps demonstrate one way to enable the execution server profile:
   a. Backup the **`web.xml`** file from the full profile

   ```
   $ mv web.xml web-full.xml
   ```

   b. Rename the **`web-exec-server.xml`** file:

   ```
   $ mv web-exec-server.xml web.xml
   ```

3. Start application server with additional system property to instruct the profile manager to activate given profile.

  - **`Dorg.kie.active.profile=full`** - to activate full profile or skip the property completely
  - **`Dorg.kie.active.profile=exec-server`** - to activate execution server profile
  - **`Dorg.kie.active.profile=ui-server`** - to activate UI server profile

2.3. BRANDING THE BUSINESS CENTRAL APPLICATION

The Business Central web application enables you to customize its look and feel by allowing you to override some of its default styles. The ability to customize the Business Central branding allows you to get a consistent appearance across all your applications thereby improving the user experience. It also helps in cases when multiple teams are using the application. Each team can develop their own customized user interface. The customizable elements are built using cascading style sheets (CSS), images, and HTML files, providing an easy and flexible approach to customize without having to recompile the code.

You can modify the following elements in the Business Central application to make it inline with your company's brand:

- Login screen

  You can customize the following attributes of the Business Central login screen:

  - The background image
  - The company logo
  - The application logo

- Application header

  You can customize the following attributes of the Business Central application header:

  - The Business Central header containing the title and banner logo

- Help pop-up windows

  You can customize the following attributes of the splash help pop-up windows:

  - The splash help images
  - The label text

2.3.1. Customizing Business Central Login Page

Procedure 2.2. Changing the Business Central Login Page Background Image

1. Start the EAP server and open http://localhost:8080/business-central in a web browser.
2. Copy the new background image to the **`$EAP_HOME/standalone/deployments/business-central.war/images`** directory in your JBoss BPM Suite installation.
3. Navigate to **`$EAP_HOME/standalone/deployments/business-central.war/styles`** directory and open the **`login-screen.css`** file in a text editor.

4. In the **`login-screen.css`** file, provide the location of your new background image in the following ***`background-image`*** attribute.

   ```
   background-image: url("../images/login-screen-background.jpg");
   ```

   The ***`background-image`*** attribute points to the default **`login-screen-background.jpg`** image.

   In addition to the background image, you can modify other attributes such as image size, position, and background color in the **`login-screen.css`** file.

Refresh the Business Central login page to view your changes.

Procedure 2.3. Changing the Business Central Login Page Company Logo and Project Logo

1. Start the EAP server and open http://localhost:8080/business-central in a web browser.
2. Navigate to the **`$EAP_HOME/standalone/deployments/business-central.war/images`** directory in your JBoss BPM Suite installation.
3. Replace the default image **`login-screen-logo.png`** with a new one. This is the company logo that appears on the top right hand corner of the login page.
4. Replace the default image **`RH_JBoss_BPMS_Logo.pngRH_JBoss_BRMS_Logo.png`** with a new one. This is the project logo that appears on the center left hand side of the login page.

Refresh the Business Central login page to view your changes.

2.3.2. Customizing Business Central Application Header

Procedure 2.4. Changing the Business Central Application Header (Banner)

1. Start the EAP server and open http://localhost:8080/business-central in a web browser.
2. Log in to the Business Central application with your user credentials.
3. Copy your new application header image to the **`$EAP_HOME/standalone/deployments/business-central.war/banner`** directory in your JBoss BPM Suite installation.
4. Open **`$EAP_HOME/standalone/deployments/business-central.war/banner/banner.html`** file in a text editor.
5. In the **`banner.html`** file, edit the following <img> tag to provide the name of your new header image:

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

   The default image is **`logo.png`**.

Refresh the Business Central Home page to view your changes.

2.3.3. Customizing Business Central Splash Help Windows

The **`$EAP_HOME/standalone/deployments/business-central.war/plugins`** directory contains the splash pages and the corresponding html files. Each splash page holds the name of the html file, which contains information about the image(s) and the text to be displayed. For example, the **`authoring_perspective.splash.js`** splash page points to the **`authoring_perspective.splash.html`** file. The **`authoring_perspective.splash.html`** contains the names and location of all the image files that appear on the Authoring Perspective splash help and also their captions. You can customize the images and the corresponding captions of the existing splash help pop-up windows.

Procedure 2.5. Changing the Business Central Splash Help Pop-Up Images and Captions

1. Start the EAP server and open http://localhost:8080/business-central in a web browser.

2. Log in to the Business Central application with your user credentials.

3. Copy your new splash help image(s) to the **`$EAP_HOME/standalone/deployments/business-central.war/images`** directory in your JBoss BPM Suite installation.

4. Open the corresponding html file from **`$EAP_HOME/standalone/deployments/business-central.war/plugins`** directory in a text editor.

5. 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="">
   ```

   The default image is **`authoring_perspective1.png`**, which appears on the first page of the Authoring Perspective splash help.

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

   ```
   <h4>Authoring</h4>
   <p>Modularized and customizable workbench</p>
   ```

Refresh the Business Central Home page and access the splash help pop-up windows to view your changes.

2.4. DEPLOYMENT DESCRIPTORS

Processes and rules within Red Hat JBoss BPM Suite 6 onwards are stored in Apache Maven based packaging, and are known as knowledge archives or kjar. The rules, processes, assets, etc. are part of a jar file built and managed by Maven. A file kept inside the **`META-INF`** directory of the kjar called **`kmodule.xml`** can be used to define the knowledge bases and sessions. This **`kmodule.xml`** file, by default, is empty.

Whenever a runtime component such as Business Central is about to process the kjar, it looks up **`kmodule.xml`** to build the runtime representation.

Deployment Descriptors, a new feature introduced in the 6.1 branch of Red Hat JBoss BPM Suite, allows you fine grained control over your deployment and supplements the **`kmodule.xml`** file. The presence of these descriptors is optional and your deployment will proceed successfully without them. The properties that you can set using these descriptors are purely technical in nature and include meta values like persistence, auditing and runtime strategy.

These descriptors allow you to configure the execution server on multiple levels (server level default, different deployment descriptor per kjar and so on). This allows you to make simple customizations to the execution server's out-of-the-box configuration (possibly per kjar).

You define these descriptors in a file called **`kie-deployment-descriptor.xml`** and place this file next to your **`kmodule.xml`** file in the **`META-INF`** folder. You can change this default location (and the filename) by specifying it as a system parameter:

```
-Dorg.kie.deployment.desc.location=file:/path/to/file/company-deployment-
descriptor.xml
```

2.4.1. Deployment Descriptor Configuration

Deployment descriptors allow the user to configure the execution server on multiple levels:

- server level: the main level and the one that applies to all kjars deployed on the server.
- kjar level: this allows you to configure descriptors on a per kjar basis.
- deploy time level: descriptors that apply while a kjar is being deployed.

The granular configuration items specified by the deployment descriptors take precedence over the server level ones, except in case of configuration items that are collection based, which are merged. The hierarchy works like this: *deploy time configuration > kjar configuration > server configuration*.

NOTE

The deploy time configuration applies to deployments done via the REST API.

For example, if the persistence mode (one of the items you can configure) defined at the server level is ***`NONE`*** but the same mode is specified as ***`JPA`*** at the kjar level, the actual mode will be ***`JPA`*** for that kjar. If nothing is specified for the persistence mode in the deployment descriptor for that kjar (or if there is no deployment descriptor), it will fall back to the server level configuration, which in this case is ***`NONE`*** (or to ***`JPA`*** if there is no server level deployment descriptor).

Can you override this hierarchal merge mode behavior?

Yes. In the default way, if there are deployment descriptors present at multiple levels, the configuration properties are merged with the granular ones overriding the coarse values, and with missing configuration items at the granular level being supplied with those values from the higher levels. The end result is a merged Deployment Descriptor configuration. This default merge mode is called the ***`MERGE_COLLECTIONS`*** mode. But you can change it (Section 2.4.2, "Managing Deployment Descriptors") if it doesn't suit your environment to one of the following modes:

- KEEP_ALL: in this mode, all higher level values override all lower level values (server level values replace kjar level values)
- OVERRIDE_ALL: in this mode, all lower level values override all higher level values (kjar values replace server level values)
- OVERRIDE_EMPTY: in this mode, all *non empty* configuration items from lower levels replace those at higher levels, including items that are represented as collections.

- MERGE_COLLECTIONS (DEFAULT): in this mode, all non empty configuration items from lower level replace those from higher levels (like in OVERRIDE_EMPTY), but collection properties are merged (combined).

Deployment Descriptors from dependent kjars are placed lower than the actual kjar being deployed, but they still have higher hierarchy than the server level.

Do I need to provide a full Deployment Descriptor for all kjars?

No. And this is where the beauty of the merge between different files can help you. Providing partial Deployment Descriptors is possible and recommended. For example, if you want to only override the audit mode in a kjar, then you just need to provide that and the rest of the values will be merged from server level or higher level kjars.

It is worth noting that when using OVERRIDE_ALL merge mode, all configuration items should be specified since the relevant kjar will always use them and will not merge with any other deployment descriptor in the hierarchy.

What can you configure?

High level technical configuration details can be configured via deployment descriptors. The following table lists these along with the permissible and default values for each.

Table 2.1. Deployment Descriptors

Configuration	XML Entry	Permissible Values	Default Value
Persistence unit name for runtime data	persistence-unit	Any valid persistence package name	org.jbpm.domain
Persistence unit name for audit data	audit-persistence-unit	Any valid persistence package name	org.jbpm.domain
Persistence mode	persistence-mode	JPA, NONE	JPA
Audit mode	audit-mode	JPA, JMS or NONE	JPA
Runtime Strategy	runtime-strategy	SINGLETON, PER_REQUEST or PER_PROCESS_INSTANCE	SINGLETON
List of Event Listeners to be registered	event-listeners	Valid listener class names as **`ObjectModel`**	No default value
List of Task Event Listeners to be registered	task-event-listeners	Valid listener class names as **`ObjectModel`**	No default value
List of Work Item Handlers to be registered	work-item-handlers	Valid Work Item Handler classes given as **`NamedObjectHandler`**	No default value

Configuration	XML Entry	Permissible Values	Default Value
List of Globals to be registered	globals	Valid Global variables given as **`NamedObjectModel`**	No default value
Marshalling strategies to be registered (for pluggable variable persistence)	marshalling-strategies	Valid **`ObjectModel`** classes	No default value
Required Roles to be granted access to the resources of the kjar	required-roles	String role names	No default value
Additional Environment Entries for Knowledge Session	environment-entries	Valid **`NamedObjectModel`**	No default value
Additional configuration options of Knowledge Session	configurations	Valid **`NamedObjectModel`**	No default value

How do you provide values for collections based configuration items?
In the table of valid configuration items earlier, you would have noticed that the valid values for the collection based items are either **`ObjectModel`** or **`NamedObjectModel`**. Both are similar and provide a definition of the object to be built or created at runtime, with the exception that the **`NamedObjectModel`** object details name the object to be looked. Both these types are defined using an identifier, optional parameters and resolver (to resolve the object).

- identifier - defines all the information about the object, such as fully qualified class name, Spring bean id or an MVEL expression.
- parameters - optional parameters that should be used while creating instances of objects from this model.
- resolver - identifier of the resolver that will be used to create object instances from the model - (reflection, mvel or Spring).

As an example, if you have built a custom marshaling strategy and want your deployments to use that strategy instead of the default, you will need to provide that strategy as an **`ObjectModel`**, with the identifier being **`com.mycompany.MyStrategy`**, resolver being reflection (the easiest and the default) and any parameters that are required for your strategy to work. Reflection will then be used to create an instance of this strategy using the fully qualified class name that you have provided as the identifier.

```
<marshalling-strategy>
 <resolver>reflection</resolver>
 <identifier>com.myCompany.MyStrategy</identifier>
 <parameters>
    <parameter xsi:type="xs:string"
xmlns:xs="http://www.w3.org/2001/XMLSchema">
      param
    </parameter>
  </parameters>
</marshalling-strategy>
```

In case reflection based on resolver is not enough (as demonstrated in the previous example), you can use a resolver based on MVEL expression as the identifier of the object model. While evaluating expressions, you can substitute out-of-the-box parameters. As an example:

```
<marshalling-strategy>
  <resolver>mvel</resolver>
  <identifier>new com.myCompany.CustomStrategy(runtimeManager)
</identifier>
</marshalling-strategy>
```

The Spring based resolver allows you to look up a bean by its identifier from a Spring application context. Whenever JBoss BPM Suite is used with Spring, this resolver helps in deploying kjars into the runtime. As an example (note that the identifier in this case is a named bean in the Spring context):

```
<marshalling-strategy>
 <resolver>spring</resolver>
 <identifier>customStrategy</identifier>
</marshalling-strategy>
```

2.4.2. Managing Deployment Descriptors

Deployment Descriptors can be edited via the Business Central in one of two ways. Either graphically (by clicking on **Authoring** → **Project Authoring** and then selecting **Tools** → **Deployment Descriptor**) or by clicking on **Authoring** → **Administration** menu and then clicking through to the **`META-INF`** folder in the File Explorer. Click on the **`kie-deployment-descriptor.xml`** file to edit it manually.

Every time a project is created, a stock **`kie-deployment-descriptor.xml`** file is generated with default values as described earlier.

Overriding Hierarchical Merge Mode Behavior

To change the default mode of MERGE_COLLECTIONS to one of KEEP_ALL, OVERRIDE_ALL or OVERRIDE_EMPTY you can use the following methods, depending on the requirement.

- Set the system property org.kie.dd.mergemode to one of these values. This merge mode will become default for all kjars deployed in the system, unless you override it at a kjar level via the next method.
- When deploying a new deployment unit via Business Central (**Deploy** → **Deployments**) you can select what merge mode should be used for that particular kjar.
- When deploying via the REST API, you can add mergemode query parameter to the command URL to one of these modes to set the merge mode for that deployment.

Restricting access to the Runtime Engine

One of the configuration items discussed earlier, required-roles, can be edited via the Deployment Descriptors. This property restricts access to the runtime engine on a per kjar or per server level by ensuring that access to certain processes is only granted to users that belong to groups defined by this property.

The security role can be used to restrict access to process definitions or restrict access at runtime.

The default behavior is to add required roles to this property based on repository restrictions. You can of course, edit these properties manually if required, as described above by providing roles that match actual roles defined in the security realm.

2.5. MANAGING DEPLOYMENT OVERRIDE POLICY

If a user tries to deploy an artifact with a GAV (Group-Id, Artifact-Id and Version) that already exists in the system, the deployment will fail and an error message will be displayed in the **`Messages`** panel.

This feature prevents the user from overwriting an existing deployment by mistake.

By default this feature is enabled, that is, by default the system will prevent the user from overwriting an existing installation with the same GAV.

However, there may be cases when the user *may* want to overwrite existing deployments with the same GAV. Although you can't enable overwriting on a per-deployment basis, you can set this up for the system as a whole by using the system setting org.kie.override.deploy.enabled. This setting, is **`false`** by default. Change it to **`true`** to enable overwriting of deployments with the same GAV by providing it at startup time of your server (**`-Dorg.kie.override.deploy.enabled=true`**).

2.6. EXTENDING BUSINESS CENTRAL

Starting with version 6.1 of JBoss BPM Suite, 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** menu and are classified under **`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's add a new screen in the system to show you the basics of this functionality.

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

Let's 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:

- Create a new screen
- Create a new perspective (and add the new screen to it)
- Create a new menu (and add the new perspective to it)
- Apps (optional)

Adding a new Screen

Click on **`New ...`** 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 up, 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 up the Screen plugin editor again if you close it, open up the **`Screen Plugin`** category and scroll past the system generated screens to your manually created plugin and click on it to open up 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.

Adding a 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 **`New...`** button and then selecting, **New Perspective**. This will open up 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 in the right hand side and then drag a **`6 6`** grid on the right hand side.

Next, open up the **`Components`** category and drag a Screen Component to the right hand side (in any grid). This will open up the **`Edit Component`** dialog box where you can enter your screen created in the previous step (**`HelloWorldJS`**). Click the **`OK`** button and then click the button to save this perspective. Enter **`HelloWorldPerspective`**, enter **`Home`** in the tag name field (and click the **`Add Tag`** button), and click the **`OK`** button to finish the save.

If you need to load this perspective again, you will require the name you have given it as this perspective doesn't appear in the list of perspectives. To load, click the button and enter the perspective name.

Adding a 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 **`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 click the **`Save`** button.

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 up 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 up the App directories by clicking on **Extensions** → **Apps**.

The Apps directory provides an alternate way to open up 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 on:

2.6.2. 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.MultiListWorkbenchPanelPr
esenter",
    view: {
        parts: [
            {
                place: "welcome",
                min_height: 100,
                parameters: {}
            }
        ],
        panels: [
            {
                width: 250,
                min_width: 200,
                position: "west",
                panel_type:
"org.uberfire.client.workbench.panels.impl.MultiListWorkbenchPanelPr
esenter",
                parts: [
                    {
                        place: "YouTubeVideos",
                        parameters: {}
                    }
                ]
            },
            {
                position: "east",
                panel_type:
"org.uberfire.client.workbench.panels.impl.MultiListWorkbenchPanelPr
esenter",
                parts: [
                    {
                        place: "TodoListScreen",
                        parameters: {}
                    }
                ]
            },
            {
                height: 400,
                position: "south",
                panel_type:
"org.uberfire.client.workbench.panels.impl.MultiTabWorkbenchPanelPre
senter",
                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
})
```

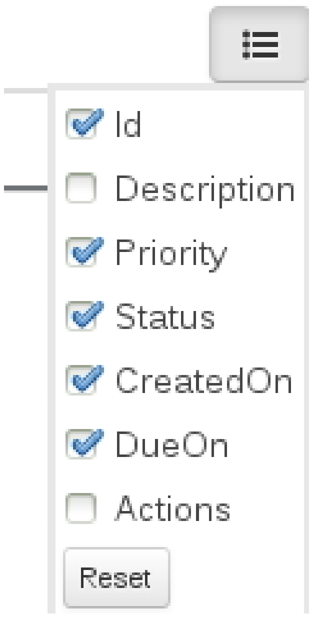
2.7. CONFIGURING TABLE COLUMNS

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

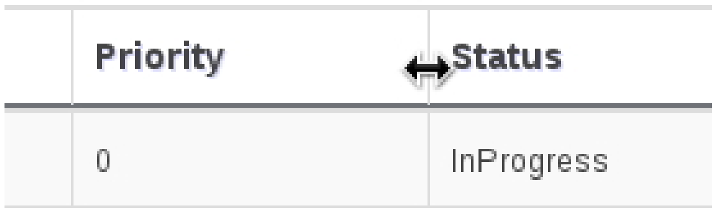
Tables that allow columns to be configured have button in the top right corner. Clicking on this button opens up the list of columns that can added or removed to the current table with a checkbox next



to each column:

Resizing Columns

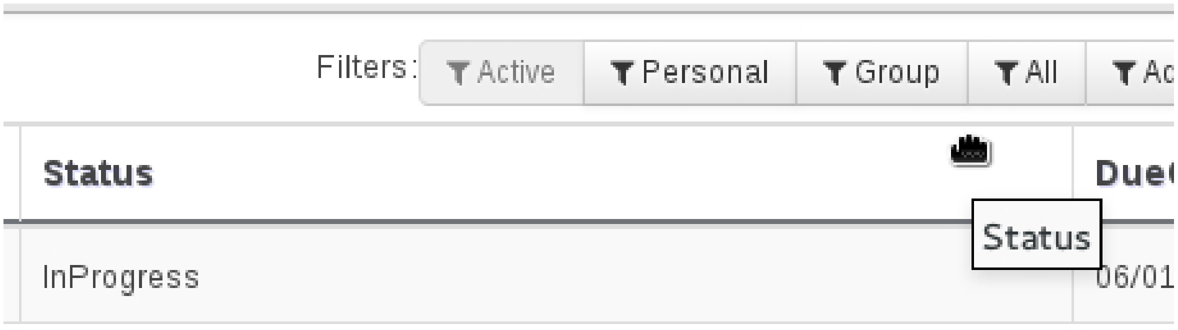
To resize columns, place your cursor between the edges of the column header and move in the direction



that you want:

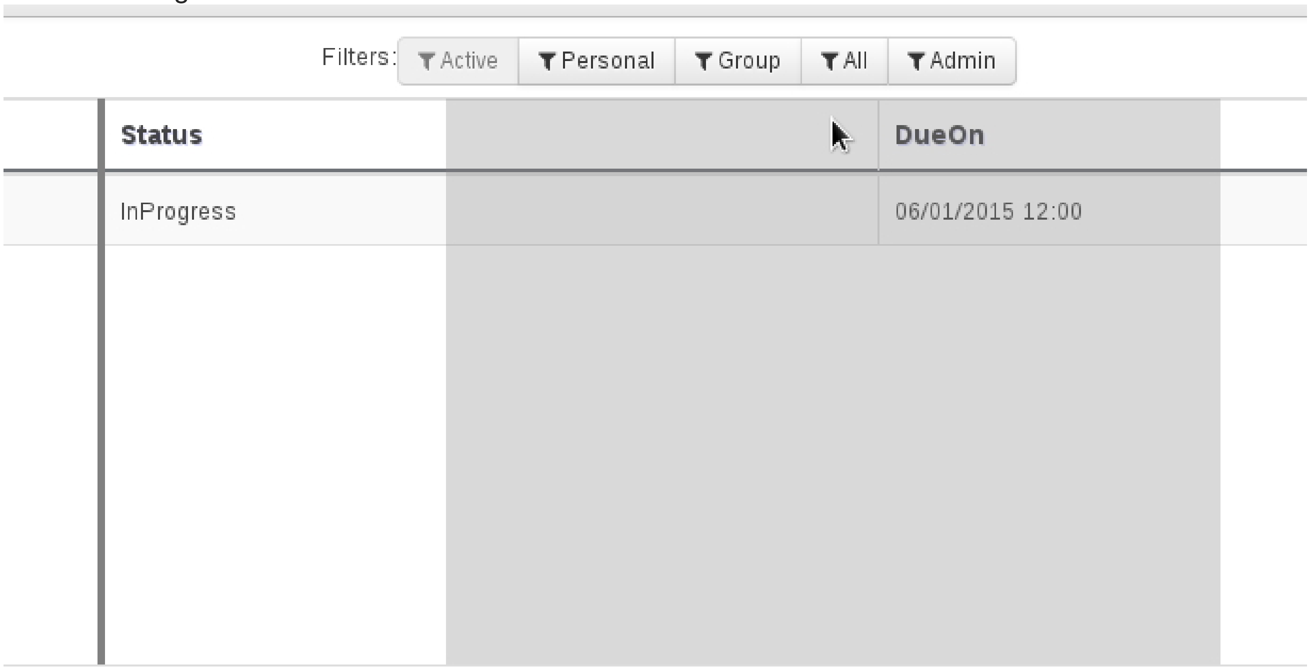
Moving Columns

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. 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.

1. **`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.

2. **`Offline mode`** (a kind of 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** → **Products Downloads**.

3. In the **`Product Downloads`** page that opens, click **`Red Hat JBoss BPM Suite`**.

4. From the **`Version`** drop-down menu, select 6.1.

5. 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`**.

3.1. 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`**

3.2. 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.

3.3. 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 4. MIGRATION

Migrating your projects from Red Hat JBoss BPM Suite 5 to Red Hat JBoss BPM Suite 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 JBoss BPM Suite 6 uses GIT for storing assets, artifacts and code repositories including processes and rules, you should start by creating an empty project in JBoss BPM Suite 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 JBoss BPM Suite 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's look at these steps in more detail in the next few sections.

4.1. DATA MIGRATION

To migrate data from Red Hat JBoss BPM Suite 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 BPM Suite Software Downloads section. Click on **`Red Hat JBoss BPM Suite 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** menu and then click on **Clone Repository** menu to start the process.

NOTE

Assets can also be migrated manually. After all, 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 BPM Suite 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.

4.2. RUNTIME MIGRATION

To run Red Hat JBoss BPM Suite 5 processes in Red Hat JBoss BPM Suite 6, do the following:

1. Set the system property **`jbpm.v5.id.strategy`** to true in the JBoss BPM Suite **`standalone.xml`** file:

```
<property name="jbpm.v5.id.strategy" value="true"/>
```

2. Load the KieSession as shown here:

   ```
   KieSession ksession =
   JPAKnowledgeService.loadStatefulKnowledgeSession(sessionID, kbase,
   sessionConf, env);
   ```

3. Continue the normal execution of the process using KieSession methods:

   ```
   ksession.signalEvent("SomeEvent", null);
   ```

4.3. API AND BACKWARDS COMPATIBILITY

Migrating to Version 6.1

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

Red Hat JBoss BPM Suite no longer provides backward compatibility with the rule, event, and process application programming interface (API) from 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 JBoss BPM Suite 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 JBoss BRMS 5.x and JBoss BPM Suite 6.x, namely the task service API and the REST API.

Migrating to Version 6.0

The JBoss BPM Suite 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 JBoss BPM Suite 5, note that this has changed as well and there is no mechanism in it for backwards compatibility.

4.4. MIGRATING TASK SERVICE

JBoss BPM Suite 6 provides support for a locally running task server only. This means that you do not need to setup any messaging service in your project. This differs from JBoss BPM Suite 5 because it provided a task server that was bridged from the core engine by using, most commonly, the messaging system provided by HornetQ.

To help you bridge the gap until you can migrate this in your current architecture, there is a helper or utility method, **`LocalHTWorkItemHandler`**.

Since the TaskService API is part of the public API you will now need to refactor your imports because of package changes and refactor your methods due to API changes themselves.

CHAPTER 5. DATA MANAGEMENT

5.1. DATA BACKUPS

When applying a backup mechanism to Red Hat JBoss BPM Suite make sure you back up the following resources:

- any customized deployment descriptors (such as, **`web.xml`**, **`jboss-web.xml`**, **`jboss.xml`**)
- any customized properties files

NOTE

Consider backing up the entire **`business-central.war`** and **`dashbuilder.war`** files.

5.2. SETUP INDEXES

Setup foreign key indexes

Some databases, for instance Oracle and Postgres, do not automatically create an index for each foreign key. This can result in deadlocks occurring. To avoid this situation it is necessary to create an index on all foreign keys, especially in the Oracle database.

Setup indexes for Process and Task Dashboard

Process and Task Dashboard in 6.1 has been refactored in order to cope with high volume of task and process instances. In order to get good response times while querying the database the following JBoss BPM Suite tables need to be indexed: **`processinstancelog`** and **`bamtasksummary`**.

Note that *ALL* the columns in these two tables need to be indexed and not just the primary and foreign keys.

5.3. SETTING UP THE DATABASE

Dashbuilder application requires an existing database, previously created before running the application. To create a database you can use any database client tool and run the following commands:

Postgres

The following sql sentence is used to create a Postgres database:

```
CREATE DATABASE dashbuilder
  WITH ENCODING='UTF8'
         OWNER=dashbuilder
         CONNECTION LIMIT=-1
```

NOTE

The database encoding must be UTF8

DB2

DB2 database can be created using the following sql sentence:

```
CREATE DATABASE dashb PAGESIZE 16384
```

NOTE

The default pagesize for DB2 systems is 4k which is not enough for the dashbuilder table columns size. The pagesize should be forced to 16384 as shown in the above sentence.

Once the database is created, the application server datasource must be configured. You must edit the JBoss EAP configuration file and configure the datasource subsystem as the following examples:

```
<datasource jndi-name="java:jboss/datasources/jbpmDS" enabled="true" use-
java-context="true" pool-name="postgresqlDS">
    <connection-url>jdbc:postgresql://localhost/test</connection-url>
    <driver>postgresql</driver>
    <pool></pool>
    <security>
       <user-name>sa</user-name>
       <password>sa</password>
    </security>
</datasource>
<drivers>
    <driver name="postgresql" module="org.postgresql">
        <xa-datasource-class>org.postgresql.xa.PGXADataSource</xa-
datasource-class>
    </driver>
</drivers>
```

5.4. EDITING THE DATABASE

Dashbuilder requires the JBoss BPM Suite to have history log's database tables. It is mandatory to deploy the Human Task console (or a superset, i.e: kie-wb) first. Otherwise, the Dashboard will not be initialized correctly and it will not be possible to display its key performance indicators.

By default, the application is configured to use a datasource with the following JNDI name:

```
java:jboss/datasources/ExampleDS
```

This is specified in JBoss EAP's configuration file; for example, **`standalone.xml`**.

NOTE

This datasource is intended for development/demo purposes; it is present by default in any JBoss installation.

If you want to deploy on a database different from H2 like Oracle, MySQL, Postgres or MS SQL Server, please perform the following steps:

Procedure 5.1. Changing Database

1. Install the database driver on JBoss (refer to JBoss driver documentation).

2. Create an empty database and a JBoss data source which connects to the database driver.

3. Modify the file **`dashbuilder.war/WEB-INF/jboss-web.xml`**:

```
<jboss-web>
   <context-root>/dashbuilder</context-root>
   <resource-ref>
       <res-ref-name>jdbc/dashbuilder</res-ref-name>
       <res-type>javax.sql.DataSource</res-type>
       <jndi-name>java:jboss/datasources/myDataSource</jndi-name>
   </resource-ref>
   ...
```

4. Replace the jndi-name parameter value by the JNDI path of the JBoss data source you've just created.

5. Modify the file **`dashbuilder.war/WEB-INF/jboss-deployment-structure.xml`**

6. Add the following snippet of configuration inside the **`deployment`** tag, where **`jdbcDriverModuleName`** is the name of the JBoss JDBC driver module:

```
  <dependencies>
      <module name="jdbcDriverModuleName" />
  </dependencies>
```

5.5. DDL SCRIPTS

DDL scripts for database tables for both JBoss BRMS and BPM Suite are available for download via the Customer Portal. These scripts allow you to study the tables and use them to create the tables and indexes manually or in databases that are not directly supported.

To download these scripts, login to the Customer Portal and click on Red Hat JBoss BPM Suite. Select the version of the product for your requirements and then click on **`Download`** in the row **`Red Hat JBoss BPM Suite 6.x.x Supplementary Tools`** to download the supplementary tools.

Unzip the file on your machine. The DDL scripts are located in the **`ddl-scripts`** folder. Database scripts are provided for DB2, H2, MySQL5, Oracle, PostgreSQL and SQLServer.

The complete Entity Relationship diagram can be viewed in this Red Hat Solution.

CHAPTER 6. ASSET REPOSITORY

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

Knowledge store is a centralized repository for your business knowledge. It connects with the GIT repository that allows you to store different kinds of knowledge assets and artifacts at a single location. Business Central provides a web front-end that allows users to view and update the stored content. You can access it using the **`Project Explorer`** from the unified environment of Red Hat JBoss BPM Suite.

6.1. CREATING A REPOSITORY

IMPORTANT

Note that only user with the **`ADMIN`** role can create a repository.

Procedure 6.1. Creating a New Repository

1. Open the **`Administration`** perspective: on the main menu, click **Authoring** → **Administration**.
2. On the perspective menu, click **Repositories** → **New Repository**.
3. The **`Create Repository`** pop-up window is displayed.

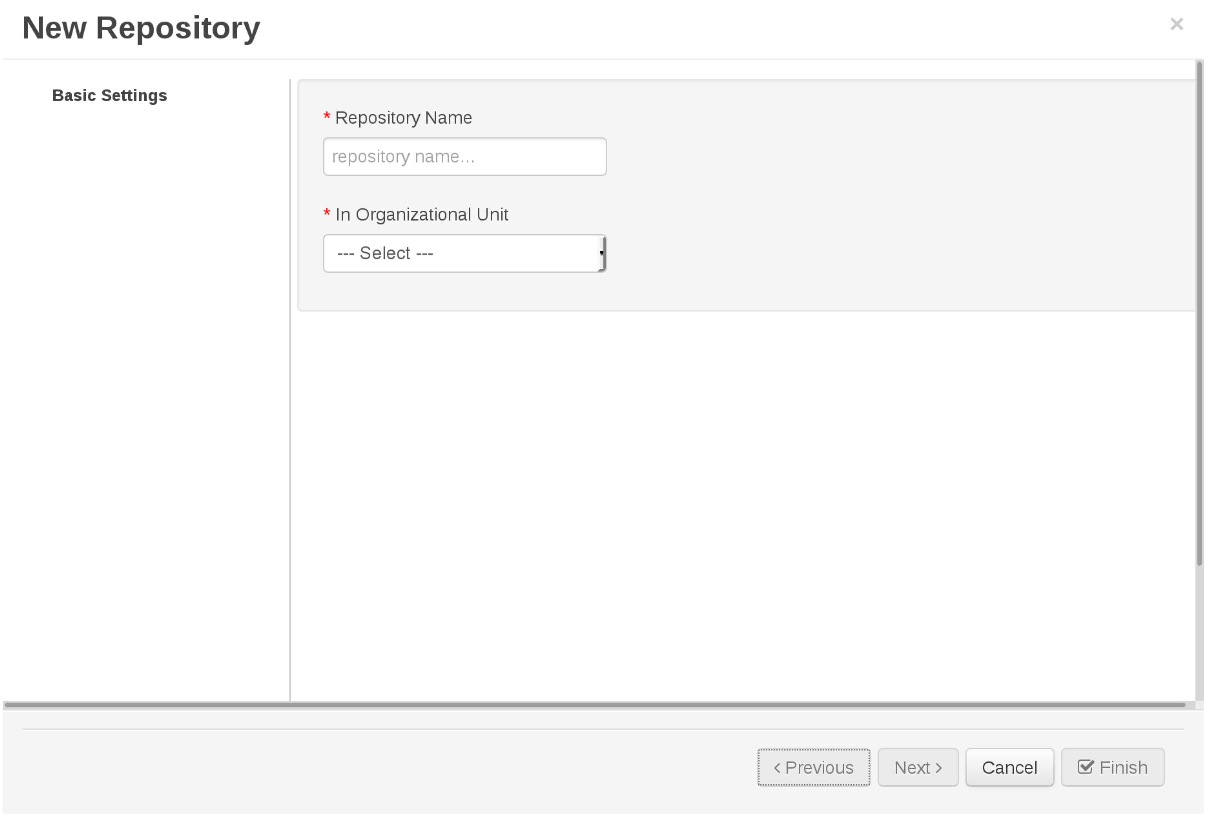


Figure 6.1. Create Repository Pop-up

4. Enter the mandatory details:
   - Repository name.

NOTE

Note that the repository name should be a valid filename. Avoid using a space or any special character that might lead to an invalid folder name.

- Select an organizational unit in which the repository is to be created from the **`Organizational Unit`** drop-down option.

5. Click **`Finish`**

The new repository can be viewed either in the **`File Explorer`** or **`Project Explorer`** views.

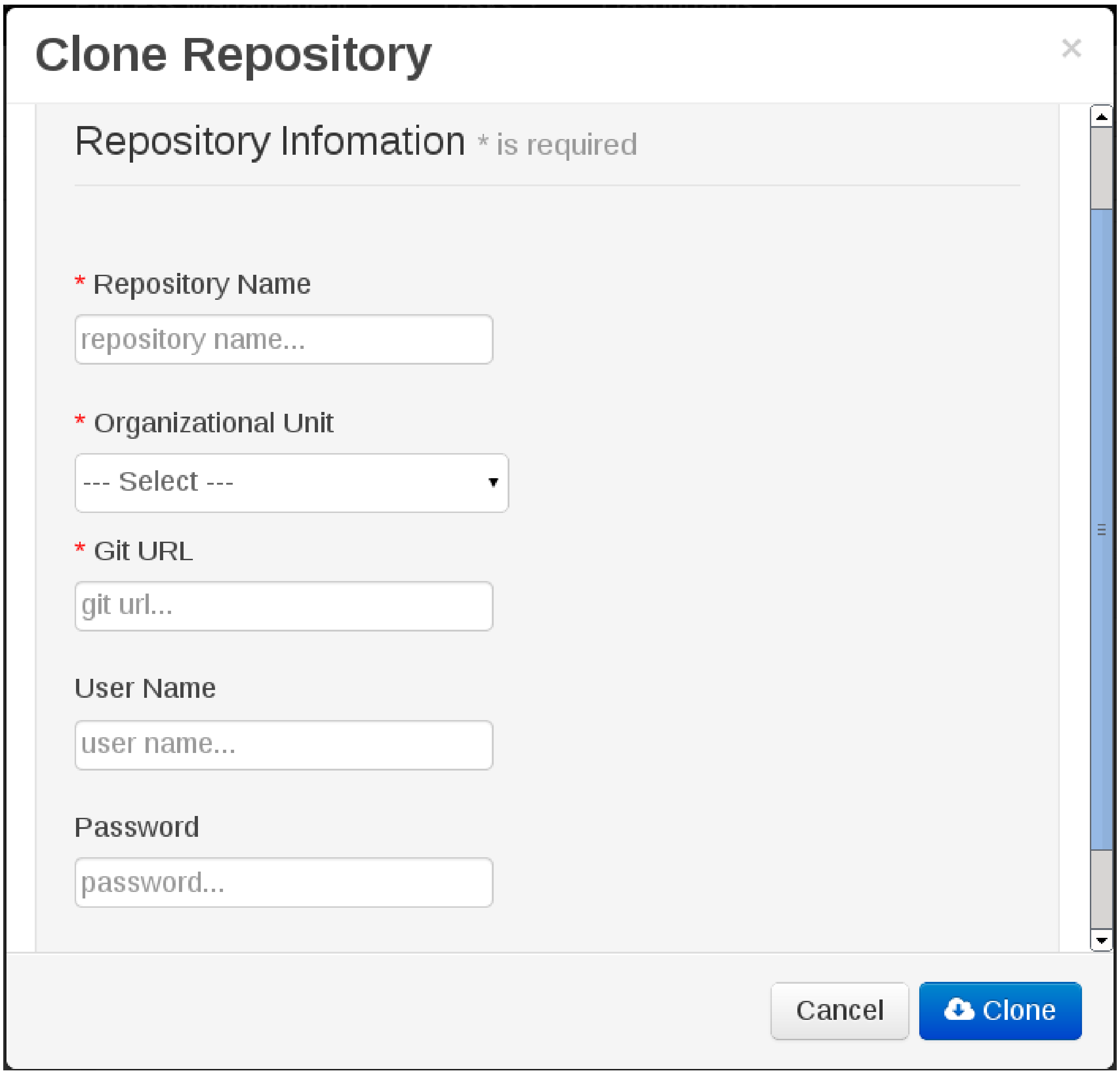
6.2. CLONING A REPOSITORY

IMPORTANT

Note that only user with the **`ADMIN`** role can clone a repository.

Procedure 6.2. Cloning a repository

1. Open the **`Administration`** perspective.
2. On the **`Repositories`** menu, select **Clone repository**.
3. The **`Clone Repository`** pop-up window is displayed.


Clone Repository

Repository Infomation * is required

* Repository Name

repository name...

* Organizational Unit

--- Select ---

* Git URL

git url...

User Name

user name...

Password

password...

Figure 6.2. Clone Repository Pop-up

4. 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: **`file:///`*`path-to-repository/reponame`***

      - For a Remote or preexisting Repository: **`git://`*`hostname/reponame`***

      NOTE

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

   c. If applicable, enter the **`User Name`** and **`Password`** to be used for authentication when cloning the repository.

5. Click **`Clone`**.

6. A confirmation prompt with an **OK** button is displayed which notifies the user that the repository is created successfully. Click **OK**.The repository will be indexed. Some workbench features may be unavailable until indexing has completed.

The cloned repository can be checked either in the **`File Explorer`** or **`Project Explorer`** views.

6.3. REMOVING A REPOSITORY

Repositories can be removed using any of the following procedures.

Removing a Repository from Business Central

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

Procedure 6.3. Using Business Central to Remove a Repository

1. Access the **RepositoryEditor** in Business Central **Authoring** → **Administration**.
2. Select **Repositories** from the tree menu on the left.
3. In the **RepositoryEditor** on the right, locate the repository to be deleted from the list of available repositories.
4. Select **master** from the drop-down menu, and click the **`Delete`** button.
5. The following message will appear:

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

   Press **Ok** to delete.

Removing a Repository using the kie-config-cli Tool

Repositories can be removed using the **`kie-config-cli`** tool via the **`remove-repo`** command.

For more information about the **`kie-config-cli`** tool, see Chapter 3, *Command line configuration*.

Removing a Repository using the REST API

To remove a repository from the Knowledge Store, issue the **DELETE** REST API call. This call relies on the user having created an authenticated HTTP session before issuing this command.

Example 6.1. Removing a repository using curl

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

6.4. MANAGING ASSETS

NOTE

The content in this section is classified as Technical Preview for the 6.1 release of Red Hat JBoss BPM Suite. It is provided as is and no support is provided.

To activate and use the featured described here, you will need to login to Business Central with a user that has been given the special role of **`kiemgmt`**.

To make management of projects easier, Red Hat JBoss BPM Suite 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.

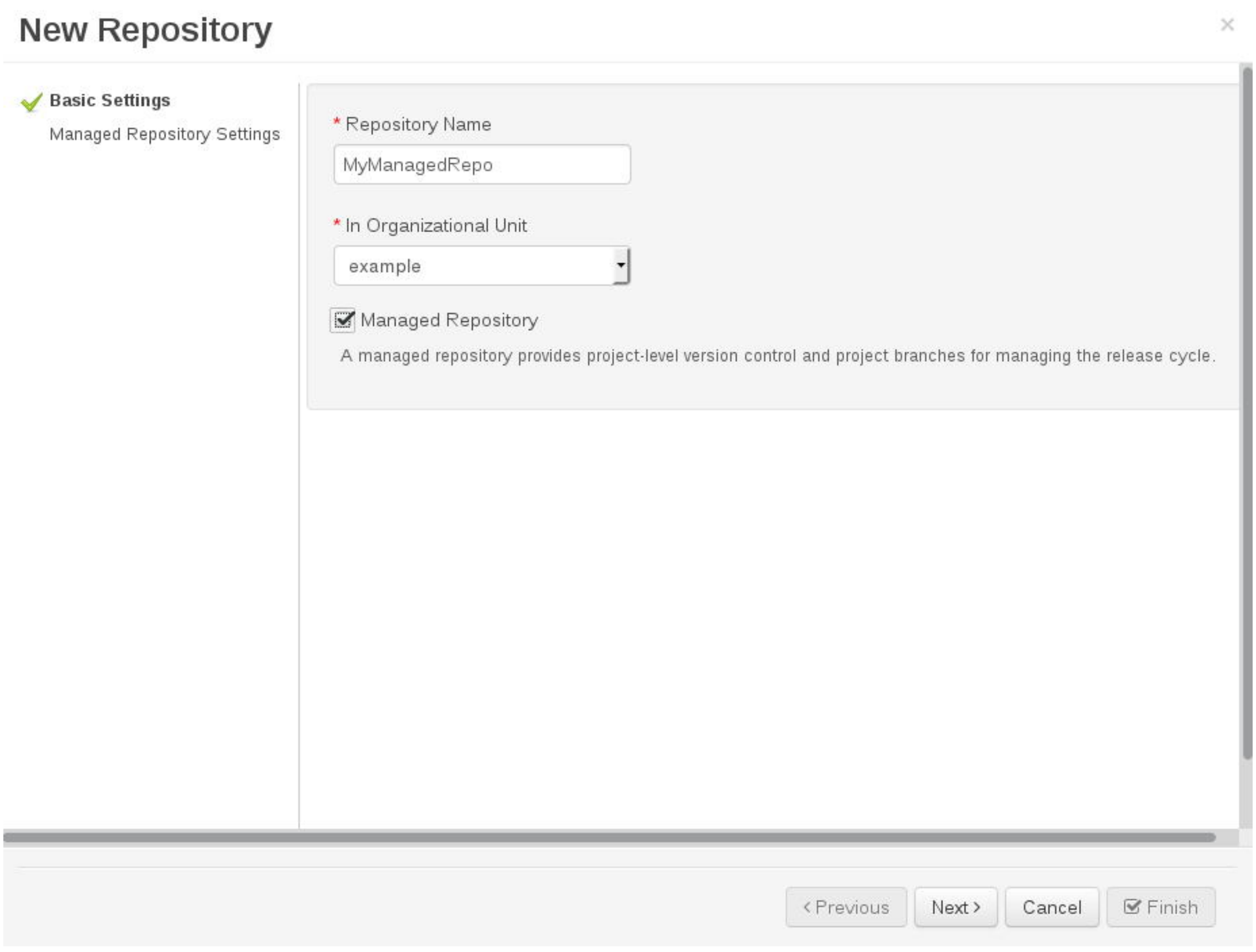
To start with, repositories can now be managed or unmanaged.

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 a Managed Repository is created the asset management configuration process is automatically launched in order to create the repository branches, and the corresponding project structure is also created.

To create a Managed or Unmanaged Repository, open up the screen for creating a new repository. This is achieved by going to **Authoring** → **Administration** and then clicking on **Repositories** → **New Repository**. This will bring up the **`New Repository`** screen.


New Repository

Basic Settings
Managed Repository Settings

* Repository Name: MyManagedRepo

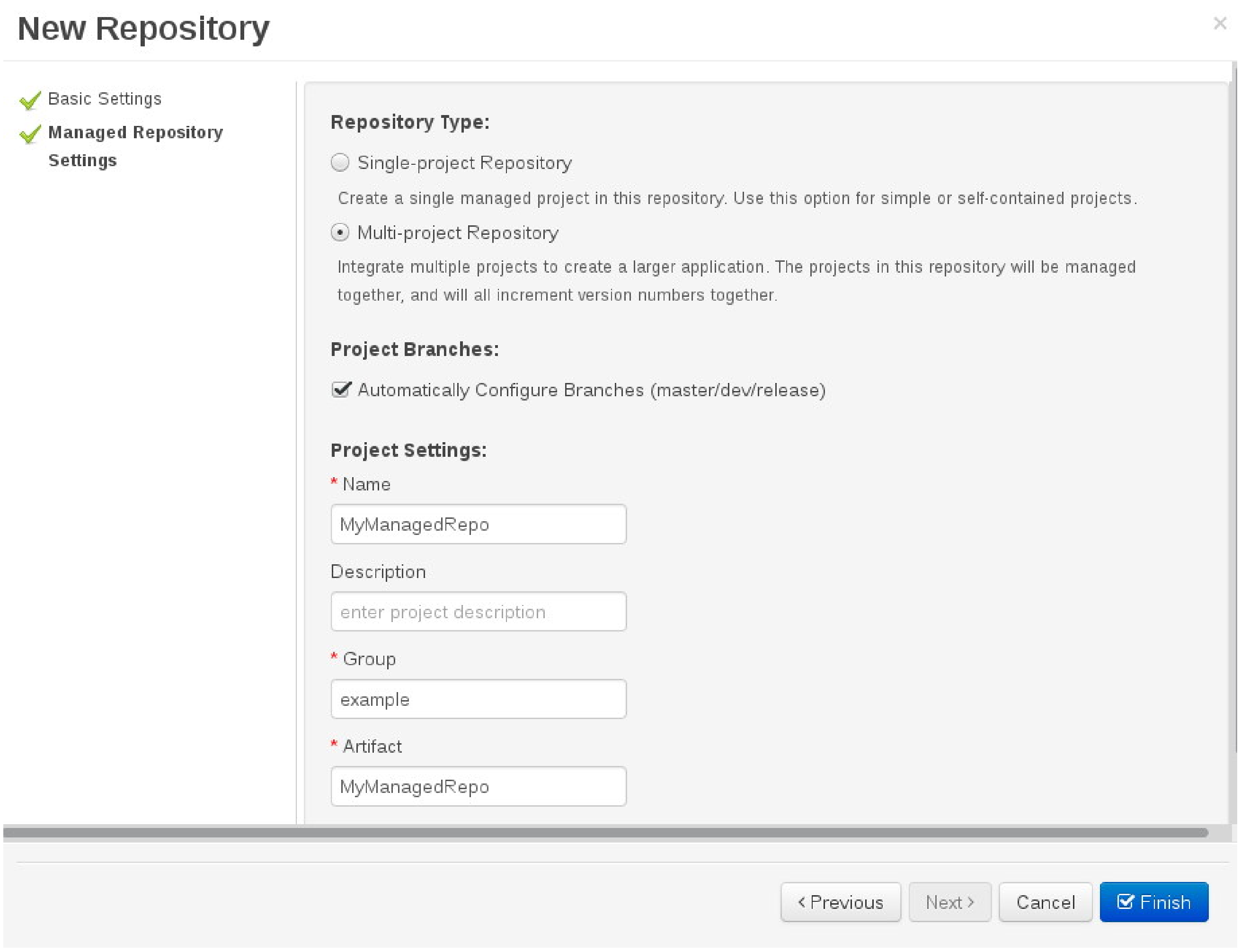
* In Organizational Unit: example

[✓] Managed Repository

A managed repository provides project-level version control and project branches for managing the release cycle.

The Unmanaged Repository creation is the same as before; Enter the name of the repository and select the organizational unit that it belongs to and click the **`Finish`** button.

To create a Managed Repository, select the **`Managed Repository`** checkbox, after giving the repository a name and the organizational unit it belongs to. Click the **`Next`** button to enter details about this Managed Repository.


New Repository

✓ Basic Settings
✓ **Managed Repository Settings**

Repository Type:

( ) Single-project Repository

Create a single managed project in this repository. Use this option for simple or self-contained projects.

(●) Multi-project Repository

Integrate multiple projects to create a larger application. The projects in this repository will be managed together, and will all increment version numbers together.

Project Branches:

[✓] Automatically Configure Branches (master/dev/release)

Project Settings:

* Name: MyManagedRepo

Description: enter project description

* Group: example

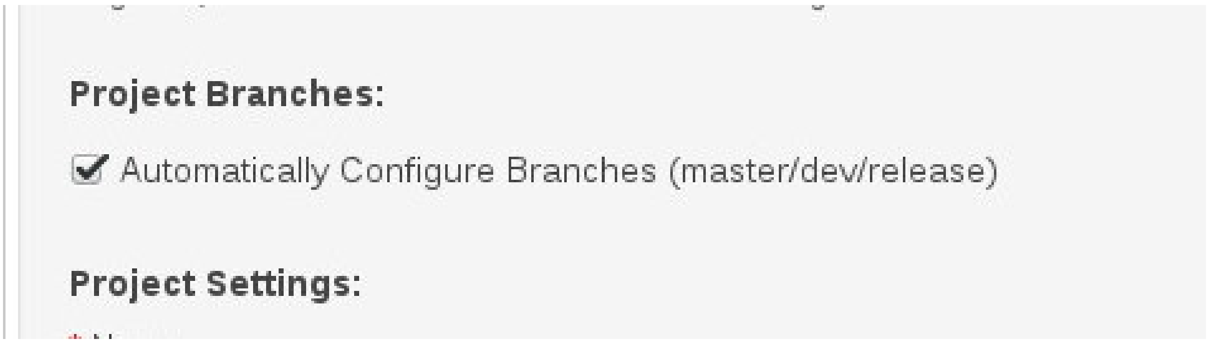
* Artifact: MyManagedRepo

Select the **`Single Project`** label if the project you are creating is a simple project and is self-contained. Enter the details of the managed project, along with the GAV details. 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 other smaller projects, select the **`Multi-Project`** repository. All Projects created in a multi-project repository will be managed together, with their version numbers being incremented together as well. Also enter the details of the parent project and the GAV, which will be inherited by all future projects that you create in this Managed Repository.

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).

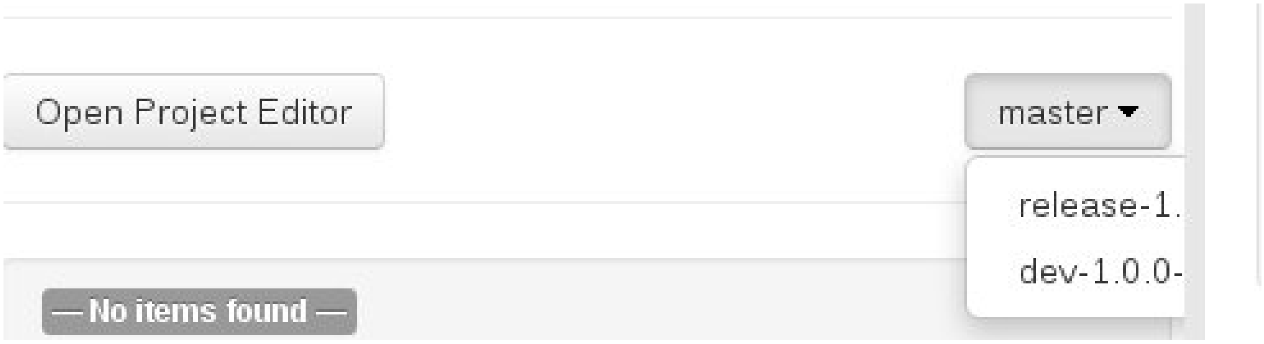


Project Branches:

✓ Automatically Configure Branches (master/dev/release)

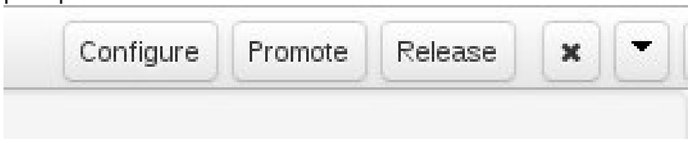
Project Settings:

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

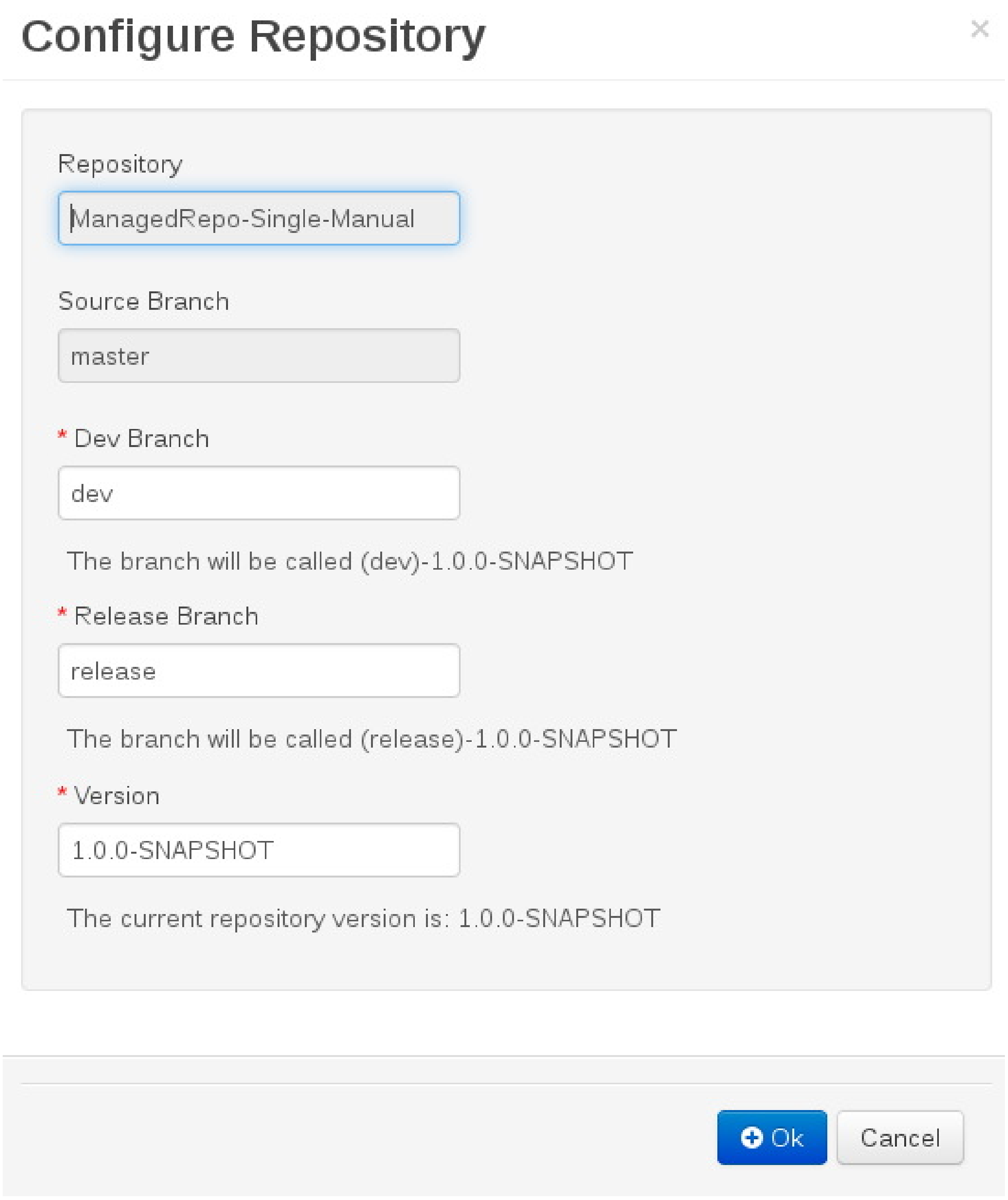


Repository Structure

If you don't 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.


Configure Repository

Repository

ManagedRepo-Single-Manual

Source Branch

master

* Dev Branch

dev

The branch will be called (dev)-1.0.0-SNAPSHOT

* Release Branch

release

The branch will be called (release)-1.0.0-SNAPSHOT

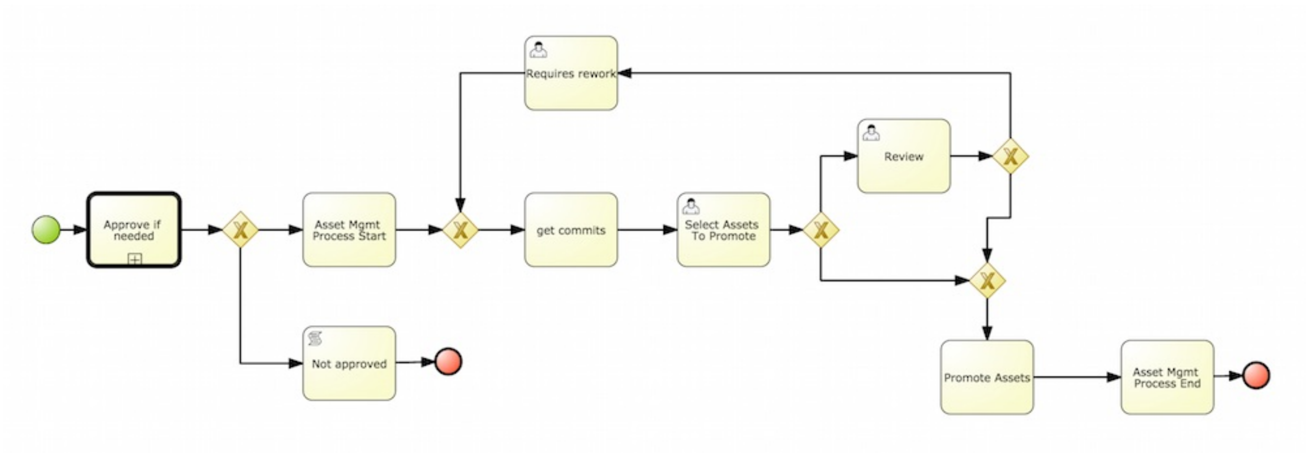
* Version

1.0.0-SNAPSHOT

The current repository version is: 1.0.0-SNAPSHOT

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:

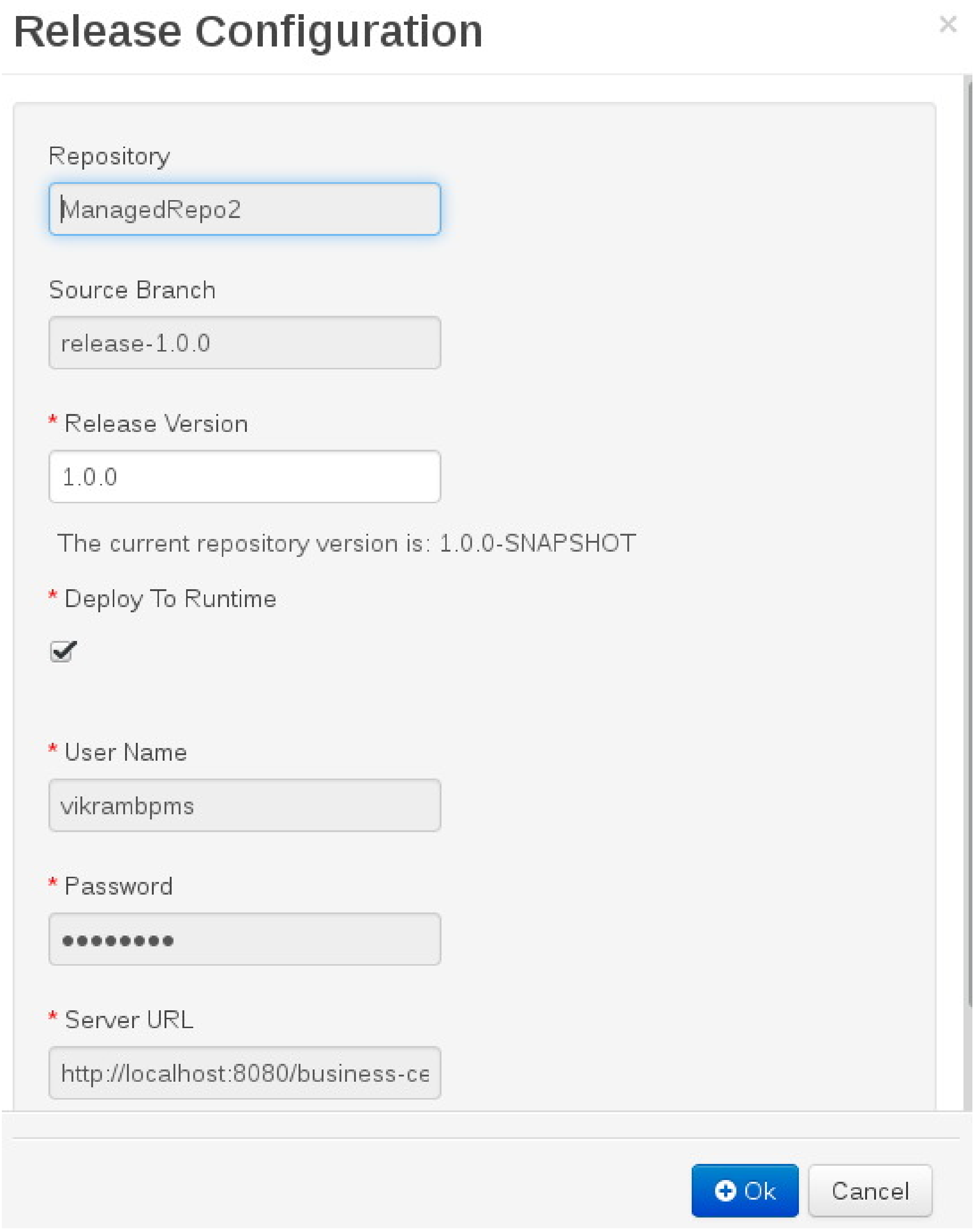


Similarly, when you click on the **`Release`** button, a release process flow is initiated. This process flow builds the project and updates all the Maven artifacts to the next version, and deploys the project to the runtime, if runtime deployment details are supplied.

WARNING

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

.


Release Configuration

Repository

ManagedRepo2

Source Branch

release-1.0.0

* Release Version

1.0.0

The current repository version is: 1.0.0-SNAPSHOT

* Deploy To Runtime

☑

* User Name

vikrambpms

* Password

●●●●●●●●●

* Server URL

http://localhost:8080/business-ce

6.5. 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.

6.6. 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 **`~/.m2`** folder, unless you have changed its location by modifying the kie.maven.settings.custom system property.

```
<servers>
  <server>
    <id>deployment</id>
    <username>admin</username>
    <password>admin.123</password>
  </server>
</servers>
```

With this configuration in place, clicking the **`Build and 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.

6.7. SYSTEM CONFIGURATION

In JBoss EAP, to change a Business Central property, such as the configuration for SSH, do the following:

Procedure 6.4. Changing System Properties

1. Edit the file **`$JBOSS_HOME/domain/configuration/host.xml`**
2. Locate the XML elements server that belong to the main-server-group and add the system property. For example:

   ```
   <system-properties>
    <property name="org.uberfire.nio.git.dir" value="..." boot-
   time="false"/>
    ...
   </system-properties>
   ```

Here is a list of all the available system properties:

- **`org.uberfire.nio.git.dir`**: Location of the directory .niogit. Default: working directory
- **`org.uberfire.nio.git.daemon.enabled`**: Enables/disables GIT daemon. Default: true
- **`org.uberfire.nio.git.daemon.host`**: If GIT daemon enabled, uses this property as the localhost identifier. Default: localhost
- **`org.uberfire.nio.git.daemon.port`**: If GIT daemon is enabled, uses this property as the port number. Default: 9418
- **`org.uberfire.nio.git.ssh.enabled`**: Enables/Disables SSH daemon. Default: true
- **`org.uberfire.nio.git.ssh.host`**: If SSH daemon is enabled, uses this property as the localhost identifier. Default: localhost
- **`org.uberfire.nio.git.ssh.port`**: If SSH daemon is enabled, uses this property as the port number. Default: 8001
- **`org.uberfire.nio.git.ssh.cert.dir`**: Location of the **`.security`** directory where local certificates will be stored. Default: working directory

- **`org.uberfire.metadata.index.dir`**: Location of the **`.index`** folder for Lucene. Default: working directory
- **`org.uberfire.cluster.id`**: Name of the Helix cluster, for example: kie-cluster
- **`org.uberfire.cluster.zk`**: Connection string to Zookeeper. This is of the form **`host1:port1,host2:port2,host3:port3`**. For example: **`localhost:2188`**.
- **`org.uberfire.cluster.local.id`**: Unique id of the Helix cluster node. Note that ':' is replaced with '_'. For example: node1_12345.
- **`org.uberfire.cluster.vfs.lock`**: Name of the resource defined on the Helix cluster, for example: kie-vfs
- **`org.uberfire.cluster.autostart`**: Delays VFS clustering until the application is fully initialized to avoid conflicts when all cluster members create local clones. Default: false
- **`org.uberfire.sys.repo.monitor.disabled`**: Disable configuration monitor (do not disable unless you know what you're doing). Default: false
- **`org.uberfire.secure.key`**: Secret password used by password encryption. Default: org.uberfire.admin
- **`org.uberfire.secure.alg`**: Crypto algorithm used by password encryption. Default: PBEWithMD5AndDES
- **`org.guvnor.m2repo.dir`**: Place where Maven repository folder will be stored. Default: working-directory/repositories/kie
- **`org.kie.example.repositories`**: Folder from where demo repositories will be cloned. The demo repositories need to have been obtained and placed in this folder. This system property takes precedence over org.kie.demo and org.kie.example properties. Default: Not used.
- **`org.kie.demo`**: Enables external clone of a demo application from GitHub. This system property takes precedence over org.kie.example. Default: true.
- **`org.kie.example`**: Enables example structure composed by Repository, Organization Unit and Project. Default: false

CHAPTER 7. PROCESS EXPORT AND IMPORT

7.1. CREATING A PROCESS DEFINITION

Make sure you have logged in to JBoss BPM Suite or you are in JBoss Developer Studio with the repository connected.

To create a Process, do the following:

1. Open the Project Authoring perspective (**Authoring** → **Project Authoring**).
2. In **`Project Explorer`** (**Project Authoring** → **Project Explorer**), navigate to the project where you want to create the Process definition (in the **`Project`** view, select the respective repository and project in the drop-down lists; in the **`Repository`** view, navigate to ***`REPOSITORY/PROJECT/`*****`src/main/resources/`** directory).

   NOTE

   It is recommended to create your resources, including your Process definitions, in a package of a Project to allow importing of resources and their referencing. To create a package, do the following:

   - In the **`Repository`** view of the Project Explorer, navigate to the ***`REPOSITORY/PROJECT/`*****`src/main/resources/`** directory.
   - Go to **New Item** → **Package**.
   - In the **`New resource`** dialog, define the package name and check the location of the package in the repository.

3. From the perspective menu, go to **New Item** → **Business Process**.
4. In the **`New Processes`** dialog box, enter the Process name and click **`OK`**. Wait until the Process Editor with the Process diagram appears.

7.2. IMPORTING A PROCESS DEFINITION

To import an existing BPMN2 or JSON definition, do the following:

1. In the **`Project Explorer`**, select a Project and the respective package to which you want to import the Process definition.
2. Create a new Business Process to work in by going to **New Item** → **Business Process**.
3. In the Process Designer toolbar, click the **`Import`** icon in the editor toolbar and pick the format of the imported process definition. Note that you have to choose to overwrite the existing process definition in order to import.
4. From the **`Import`** window, locate the Process file and click **`Import`**.

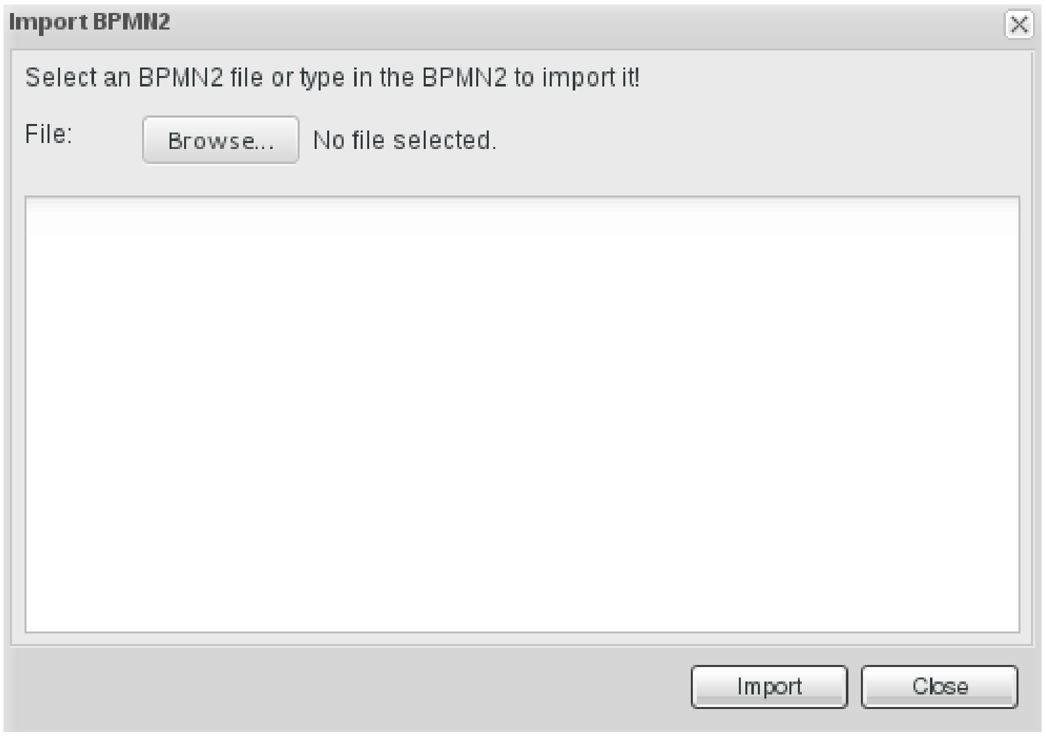


Figure 7.1. Import Window

Whenever a process definition is imported, the existing imported definition is overwritten. Make sure you are not overwriting a process definition you have edited so as not to lose any changes.

A process can also be imported to the git repository in the filesystem by cloning the repository, adding the process files, and pushing the changes back to git. In addition to alternative import methods, you can copy and paste a process or just open a file in the import dialog.

When importing processes, the Process Designer provides visual support for Process elements and therefore requires information on element positions on the canvas. If the information is not provided in the imported Process, you need to add it manually.

7.3. IMPORTING JPDL 3.2 TO BPMN2

To migrate and import a jPDL definition to BPMN2, in the Process Designer, click on the import button then scroll down and select **`Migrate jPDL 3.2 to BPMN2`**.

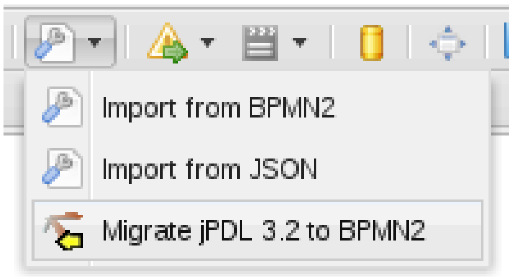


Figure 7.2. Migrate jPDL 3.2 to BPMN2

In the **`Migrate to BPMN2`** dialog box, select the process definition file and the name of the**`gpd`** file. Confirm by clicking the **`Migrate`** button.

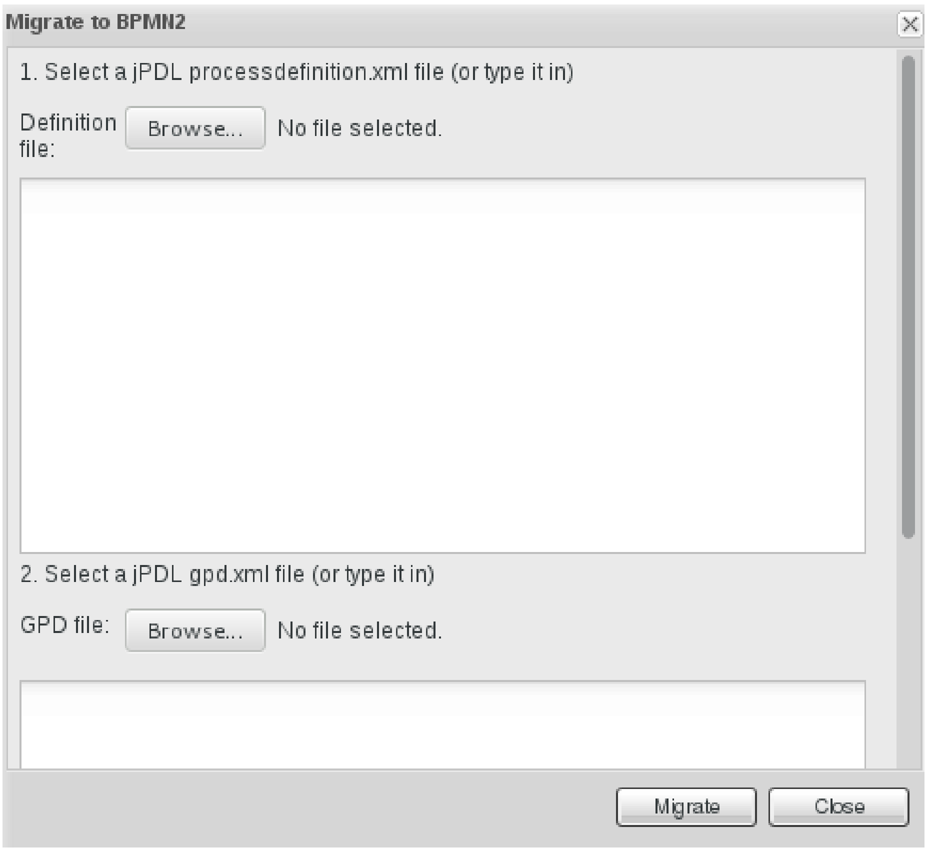


Figure 7.3. Migrate to BPMN2 dialog box

IMPORTANT

The migration tool for jPDL 3.2 to BPMN2 is a technical preview feature, and therefore not currently supported in Red Hat JBoss BPM Suite.

7.4. EXPORTING A PROCESS

Procedure 7.1. Exporting a business process

To export a business process, do the following:

1. Open the **`Project Authoring`** perspective: on the main menu, click **Authoring** → **Project Authoring**.

2. Select the business process which is to be exported, to view it in the Process Designer.

3. Click on the ( ) button of the process designer toolbar and select **`View Process Sources`** from the drop-down options.

4. The **`Process Sources`** window is displayed

5. Click on the **`Download BPMN2`** button and save the business process at the desired location.

PART III. INTEGRATION

CHAPTER 8. DEPLOYING RED HAT JBOSS BPM SUITE ARTIFACTS TO SOA REPOSITORY ARTIFACT MODEL AND PROTOCOL (S-RAMP) REPOSITORY

While Red Hat JBoss BPM Suite and S-RAMP are two independent products, it is possible to move artifacts between them. You can move artifacts from JBoss BPM Suite to S-RAMP using Maven or via a user interface.

This section provides information about these two processes.

8.1. DEPLOYING RED HAT JBOSS BPM SUITE ARTIFACTS TO SOA REPOSITORY ARTIFACT MODEL AND PROTOCOL (S-RAMP) USING MAVEN

Before you can deploy Red Hat JBoss BPM Suite artifacts to S-RAMP using Maven, you will need to enable the S-RAMP Maven Wagon. The Maven Wagon is a key feature that supports the S-RAMP Atom based REST API protocol. By enabling the S-RAMP Maven Wagon, users will be able to access artifacts from the S-RAMP repository as dependencies in a Maven project.

Enable the S-RAMP Maven Wagon by making an edit in the **`pom.xml`** file as shown below:

```
<build>
  <extensions>
    <extension>
      <groupId>org.overlord.sramp</groupId>
      <artifactId>s-ramp-wagon</artifactId>
      <version>${s-ramp-wagon.version}</version>
    </extension>
  </extensions>
</build>
```

Once the S-RAMP Maven Wagon is enabled, you can deploy the JBoss BPM Suite artifacts to that S-RAMP repository. To do this, follow the steps below:

1. Clone the git repository where you have saved the BPM Suite project by running this command:

   ```
   git clone http://localhost:8001/REPOSITORY_NAME
   ```

2. On the command line, move into the folder that contains the project.

3. Follow the instructions in *Red Hat JBoss Fuse Service Works 6 Development Guide, Volume 3: Governance*, section Deploying to S-RAMP. Use the URL from the example below:

   ```
   <distributionManagement>
     <repository>
       <id>local-sramp-repo</id>
       <name>S-RAMP Releases Repository</name>
       <url>sramp://S-RAMP_SERVER_URL/s-ramp-server/</url>
     </repository>
     <snapshotRepository>
       <id>local-sramp-repo-snapshots</id>
       <name>S-RAMP Snapshots Repository</name>
   ```

```
    <url>sramp://S-RAMP_SERVER_URL/s-ramp-server/</url>
  </snapshotRepository>
</distributionManagement>
```

With these settings, Maven deployments are sent directly to the S-RAMP repository using the S-RAMP API. Note that artifacts are added to the S-RAMP repository with an artifact type based on the Maven type of the project. You can override this behavior by adding a query parameter to the repository URL in the **pom.xml** file. For example:

```
<distributionManagement>
  <repository>
    <id>local-sramp-repo</id>
    <name>S-RAMP Releases Repository</name>
    <url>sramp://S-RAMP_SERVER_URL/s-ramp-server/?
artifactType=KieJarArchive</url>
  </repository>
</distributionManagement>
```

The above example causes the Maven artifact to be uploaded with an S-RAMP artifact type of KieJarArchive.

4. Amend the maven plug-in in file **pom.xml** and add a dependency to it as follows in case the project does not contain decision tables:

```
<plugins>
    <plugin>
      <groupId>org.kie</groupId>
      <artifactId>kie-maven-plugin</artifactId>
      <version>6.0.2-redhat-6</version>
      <extensions>true</extensions>
      <dependencies>
          <dependency>
              <groupId>org.jbpm</groupId>
              <artifactId>jbpm-bpmn2</artifactId>
              <version>6.0.2-redhat-6</version>
          </dependency>
      </dependencies>
     </plugin>
    </plugins>
```

If the project contains decision tables, use this dependency for the kie-maven-plugin instead:

```
<plugins>
    <plugin>
      <groupId>org.kie</groupId>
      <artifactId>kie-maven-plugin</artifactId>
      <version>6.0.2-redhat-6</version>
      <extensions>true</extensions>
      <dependencies>
          <dependency>
               <groupId>org.drools</groupId>
               <artifactId>drools-decisiontables</artifactId>
               <version>6.0.2-redhat-6</version>
           </dependency>
```

```
        </dependencies>
      </plugin>
    </plugins>
```

5. Run a clean Maven deployment using the following command:

```
mvn -s sramp-settings.xml deploy
```

.

NOTE

For the Maven deployment to the S-RAMP repository, it is necessary to have credentials set in the **`settings.xml`** file. For further details on the credentials, refer to *Red Hat JBoss Fuse Service Works* (FSW) documentation on Authentication.

8.2. DEPLOYING RED HAT JBOSS BPM SUITE ARTIFACTS TO SOA REPOSITORY ARTIFACT MODEL AND PROTOCOL (S-RAMP) USING GRAPHICAL USER INTERFACE (GUI)

To deploy Red Hat JBoss BPM Suite artifacts to a S-RAMP repository using the user interface, do the following:

1. In a web browser, navigate to http://localhost:8080/s-ramp-ui/. If the user interface has been configured to run from a domain name, substitute **`localhost`** for the domain name. For example http://www.example.com:8080/s-ramp-ui/.

2. Click on **`Artifacts`**.

3. In the **`Manage Artifacts`** section, select **`Import`**.

4. Locate the kie archive you want to deploy. In the dialog that opens, fill out **`KieJarArchive`** as the type, and select **`Import`**.

5. The deployment then creates these entries in the S-RAMP repository:

   `KieJarArchive`, from which it derives:

   - **`KieXmlDocument`** (if the archive contains **`kmodule.xml`**)
   - **`BpmnDocument`** (if the archive contains **`bpmn`** definitions)
   - **`DroolsDocument`** (if the archive contains **`drl`** definitions)

CHAPTER 9. INTEGRATING RED HAT JBOSS BPM SUITE WITH RED HAT JBOSS FUSE

Red Hat JBoss Fuse integration allows users of JBoss Fuse to complement their integration solution with additional features provided by JBoss BPM Suite and JBoss BRMS. Red Hat JBoss BPM Suite integration is provided by two **`features.xml`** files: one providing core JBoss BPM Suite and JBoss BRMS features, which defines the OSGi features that can be deployed into JBoss Fuse, and the other providing additional support for integration with SwitchYard and Camel.

NOTE

For users of JBoss Fuse 6.1, only the core JBoss BPM Suite and JBoss BRMS features, provided by the following features file, are supported. Customers who are using an earlier version of this file must update it.

IMPORTANT

SwitchYard integration is a Technical Preview in JBoss Fuse 6.2, and is therefore not currently supported.

Core JBoss BPM Suite and JBoss BRMS features are provided by **`jboss-brms-bpmsuite<version>-redhat<version>fuse-features.zip`**:

- 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 9.1. Features and Use Case Examples

Feature	Use Case
`drools-module`	Use the JBoss BRMS engine for rules evaluation, without requiring persistence, processes, or decision tables.
`drools-jpa`	Use the 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 JBoss BRMS engine with decision tables.
`jbpm`	Use the JBoss BPM Suite (or 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 JBoss BPM Suite (or JBoss BRMS engine with processes) with Human Task.
Core engine jars and **`kie-ci`**.	Use JBoss BRMS or JBoss BPM Suite with KieScanner (KIE-CI) to download kJARs from a Maven repository.
`kie-spring`	Use KIE-Spring integration.
`kie-spring` and **`kie-aries-blueprint`**.	Use KIE-Aries-Blueprint integration.

The following additional features for integration with SwitchYard and Camel on 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

This file (and supporting repositories) is located in http://repository.jboss.org/nexus/content/repositories/public, which is already configured for use on JBoss Fuse 6.2 out of the box in ***`installDir`*`/etc/org.ops4j.pax.url.mvn.cfg`**.

The file can also be downloaded from either the JBoss Fuse 6.2 or JBoss BPM Suite product page in the Red Hat Customer Portal.

9.1. INSTALL/UPDATE CORE INTEGRATION FEATURES

NOTE

This section refers to features in the **`jboss-brms-bpmsuite<version>-redhat<version>fuse-features.zip`** file. For additional integration features, refer to Section 9.2, “Install Additional Integration Features”.

If you have already installed an older version of the core JBoss BPM Suite and 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 9.1. Removing an Existing `drools-karaf-features` Installation

1. Start the Fuse console using:

```
$ ./installDir/bin/fuse
```

2. Unistall old features/apps 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/jbpm and remove them:

```
list -t 0 -s | grep drools
list -t 0 -s | grep kie
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-<old-version>/xml/features
```

5. Restart Fuse

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

Procedure 9.2. Install core JBoss BPM Suite and JBoss BRMS features

1. Configure required repositories

a. Edit the ***installDir*/etc/org.ops4j.pax.url.mvn.cfg** file in your JBoss Fuse installation and add the following entry to the **org.ops4j.pax.url.mvn.repositories** variable, noting that entries are separated by '**, **':

   - http://maven.repository.redhat.com/ga/@id=bxms-product-repo

2. Start JBoss Fuse:

```
$ ./installDir/bin/fuse
```

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

```
JBossFuse:karaf@root> features:addurl mvn:org.drools/drools-karaf-
features/6.2.0.Final-redhat-<version>/xml/features
```

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
```

9.2. INSTALL ADDITIONAL INTEGRATION FEATURES

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

IMPORTANT

SwitchYard integration is a Technical Preview in JBoss Fuse 6.2, and is therefore not currently supported.

Procedure 9.3. SwitchYard and Camel Integration

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

   NOTE

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

2. Add the Remote Maven Repository that contains the fuse dependencies to your **karaf** instance:

   - Edit the ***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/<SWITCH
YARD_VERSION>/xml/core-features
JBossFuse:karaf@root> features:addurl mvn:org.drools/drools-karaf-
features/<DROOLS_VERSION>/xml/features
```

4. Add a reference to the features file for additional integration with SwitchYard and Camel by running the following console command:

   ```
   JBossFuse:karaf@root> features:addurl
   mvn:org.jboss.integration.fuse/karaf-
   features/<FUSE_INTEGRATION_PACKAGE_VERSION>/xml/features
   ```

5. You can now install the features provided for SwitchYard and 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
   ```

9.3. INSTALL JBOSS FUSE INTEGRATION QUICKSTART APPLICATIONS

The following features for JBoss Fuse integration quickstart applications are provided by **`org/jboss/integration/fuse/quickstarts/karaf-features/<FUSE_INTEGRATION_PACKAGE_VERSION>/karaf-features-<FUSE_INTEGRATION_PACKAGE_VERSION>-features.xml`**:

- fuse-bxms-switchyard-quickstart-bpm-service
- fuse-bxms-switchyard-quickstart-rules-camel-cbr
- fuse-bxms-switchyard-quickstart-rules-interview
- fuse-bxms-switchyard-quickstart-rules-interview-container
- fuse-bxms-switchyard-quickstart-rules-interview-dtable
- fuse-bxms-switchyard-demo-library
- fuse-bxms-switchyard-demo-helpdesk
- fuse-bxms-camel-blueprint-drools-decision-table
- fuse-bxms-camel-spring-drools-decision-table
- fuse-bxms-jbpm-workitems-camel-quickstart
- 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 JBoss Fuse 6.2 out of the box in **`installDir/etc/org.ops4j.pax.url.mvn.cfg`**.

Procedure 9.4. Installing the Quickstart Application

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-620137/xml/features
```

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

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

Procedure 9.5. Downloading and Installing the Quickstart ZIP Files

1. Download the quickstart application ZIP file.
2. Unpack the contents of the quickstarts directory into your existing ***`installDir/`*****`quickstarts`** directory.
3. Unpack the contents of the system directory into your existing ***`installDir/`*****`system`** directory.

9.3.1. Testing Your First Quickstart Application

Procedure 9.6. Testing the Quickstart Application

1. Start JBoss Fuse:

```
$ ./installDir/bin/fuse
```

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

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

NOTE

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

3. Submit a webservice request to invoke the SOAP gateway.

   a. Open a terminal window and navigate to the associated quickstart directory that was unpacked from the quickstart 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:sub
mitOrderResponse 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 10. INTEGRATING WITH SPRING

10.1. CONFIGURING RED HAT JBOSS BPM SUITE WITH SPRING

Refer to the Red Hat JBoss BPM Suite Installation Guide to download the Spring module. You will need to download the generic deployable version of JBoss BPM Suite.

The Spring module is present in the **`jboss-bpms-engine.zip`** file and is called **`kie-spring-VERSION-redhat-MINORVERSION.jar`**.

How you intend to use the Spring modules in your application affects how you configure them.

As a Self Managed Process Engine

This is the standard way to start using JBoss BPM Suite in your Spring application. You only configure it once and run as part of the application. Using **`RuntimeManager`** API, perfect synchronization between process engine and task service is managed internally and the end user does not have to deal with the internal code to make these two work together.

As a Shared Task Service

When you use a single instance of a **`TaskService`**, you have more flexibility in configuring the task service instance as it is independent of the **`RuntimeManager`**. Once configured it is then used by the RuntimeManager when requested.

To create a RuntimeEnvironment from your Spring application, you can use the **`org.kie.spring.factorybeans.RuntimeEnvironmentFactoryBean`** class. This factory class is responsible for producing instances of **`RuntimeEnvironment`** that are consumed by RuntimeManager upon creation. Illustrated below is a configured RuntimeEnvironment with the entity manager, transaction manager, and resources for the class **`org.kie.spring.factorybeans.RuntimeEnvironmentFactoryBean`**:

```
<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>
```

The following RuntimeEnvironment can be created or configured:

- DEFAULT - default (most common) configuration for RuntimeManager
- EMPTY - completely empty environment to be manually populated
- DEFAULT_IN_MEMORY - same as DEFAULT but without persistence of the runtime engine
- DEFAULT_KJAR - same as DEFAULT but knowledge asset are taken from KJAR identified by releaseid or GAV

- DEFAULT_KJAR_CL - build directly from classpath that consists kmodule.xml descriptor

Depending upon the selected type, there are different mandatory properties that are required. However, at least one of the following knowledge properties must be provided:

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

Finally, for DEFAULT, DEFAULT_KJAR, DEFAULT_KJAR_CL types, persistence needs to be configured in the form of values for **`entity manager factory`** and **`transaction manager`**. Illustrated below is an example RuntimeManager for **`org.kie.spring.factorybeans.RuntimeManagerFactoryBean`**:

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

CHAPTER 11. CDI INTEGRATION

11.1. CDI INTEGRATION

To make use of jbpm-kie-services in your system, you will need to provide some mbeans to satisfy all dependencies of the services. There are several mbeans that depend on actual scenarios.

- entity manager and entity manager factory
- user group callback for human tasks
- identity provider to pass authenticated user information to the services

When running in JEE environment, like JBoss Application Server, the mbean should satisfy all requirements of the jbpm-kie-services

```
public class EnvironmentProducer {

    @PersistenceUnit(unitName = "org.jbpm.domain")
    private EntityManagerFactory emf;

    @Inject
    @Selectable
    private UserGroupCallback userGroupCallback;

    @Produces
    public EntityManagerFactory getEntityManagerFactory() {
        return this.emf;
    }

    @Produces
    @RequestScoped
    public EntityManager getEntityManager() {
        EntityManager em = emf.createEntityManager();
        return em;
    }

    public void close(@Disposes EntityManager em) {
        em.close();
    }

    @Produces
    public UserGroupCallback produceSelectedUserGroupCalback() {
        return userGroupCallback;
    }
    @Produces

    public IdentityProvider produceIdentityProvider {
        return new IdentityProvider() {
             // implement IdentityProvider
        };
    }
}
```

Then **`deployments/business-central.war/WEB-INF/beans.xml`** file may be configured to change the current settings of the new **`usergroupcallback`** implementation.

```
<beans xmlns="http://java.sun.com/xml/ns/javaee"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://java.sun.com/xml/ns/javaee
http://docs.jboss.org/cdi/beans_1_0.xsd">

<alternatives>
  <class>org.jbpm.services.task.identity.JAASUserGroupCallbackImpl</class>
</alternatives>

</beans>
```

NOTE

`org.jbpm.services.task.identity.JAASUserGroupCallbackImpl` is just an example here to demonstrate the settings of the application server regardless of what it actually is (LDAP, DB, etc).

CHAPTER 12. PERSISTENCE

The runtime data of the Process Engine can be persisted in data stores. The persistence mechanism saves the data using marshalling: the runtime data is converted into a binary dataset and the dataset is saved in the data storage.

Note that persistence is not configured by default and the engine runs without persistence.

NOTE

The runtime data is saved using marshalling (binary persistence). The marshalling mechanism is a custom serialization mechanism.

Red Hat JBoss BPM Suite will persist the following when persistence is configured:

- Session state: this includes the session ID, date of last modification, the session data that business rules would need for evaluation, state of timer jobs.
- Process instance state: this includes the process instance ID, process ID, date of last modification, date of last read access, process instance start date, runtime data (the execution status including the node being executed, variable values, etc.)and the eventtypes.
- Work item runtime state: this includes the work item ID, creation date, name, process instance ID, and the work item state itself.

Based on the persisted data, it is possible to restore the state of execution of all running process instances in case of failure or to temporarily remove running instances from memory and restore them later. By default, no persistence is configured.

To allow persistence, you need to add the jbpm-persistence jar files to the classpath of your application and configure the engine to use persistence. The engine automatically stores the runtime state in the storage when the engine reaches a safe point. Safe points are points where the process instance has paused. When a process instance invocation reaches a safe point in the engine, the engine stores any changes to the process instance as a snapshot of the process runtime data. However, when a process instance is completed, the persisted snapshot of process instance runtime data is automatically deleted.

If a failure occurs and you need to restore the engine runtime from the storage, the process instances are automatically restored and their execution resumes so there is no need to reload and trigger the process instances manually.

The runtime persistence data is to be considered internal to the engine. You should not access persisted runtime data or modify them directly as this might have unexpected side effects.

To obtain information about the current execution state, refer to the history log. Query the database for runtime data only if absolutely necessary.

12.1. SESSION

Sessions are persisted as `SessionInfo` entities. These persist the state of the runtime KIE session, and store the following data:

Table 12.1.

Field	Description	Nullable
id	primary key	false
lastmodificationdate	last saved to data store	N/A
rulesbytearray	binary dataset with session state (binary blob	false
startdate	session start	
optlock	version number used to lock value for optimistic locking	

12.2. PROCESS INSTANCE

Process instances are persisted as **`ProcessInstanceInfo`** entities, which persist the state of a process instance on runtime and store the following data:

Table 12.2.

Field	Description	Nullable
instanceid	primary key	false
lastmodificationdate	last saved to data store	N/A
lastreaddate	last read from data store	N/A
processid	ID of the process the instance is based on	false
processinstancebytearray	binary dataset with process instance state (binary blob)	false
startdate	Process instance start date	
optlock	version number used lock value for optimistic locking	
state	Process instance state	false

`ProcessInstanceInfo` has a 1:N relationship to the **`EventTypes`** entity.

The **`EventTypes`** entity contains the following data:

Table 12.3.

Field	Description	Nullable
instanceid	reference to the Process instance (foreign key to the **`processinstanceinfo`**)	false
element	text field related to an event the Process instance has undergone	

Pessimistic Locking Support

The default locking mechanism for persistence of processes is *optimistic*. With multi-thread high concurrency to the same process instance, this locking strategy can result in bad performance.

With the release of the 6.1 version of Red Hat JBoss BPM Suite, this can be changed at runtime to allow the user to set locking on a per process basis and to allow it to be *pessimistic* (the change can be made at a per KIE Session level or Runtime Manager level as well and not just at the process level).

To set a process to use pessmistic locking, do this in the runtime environment:

```
import org.kie.api.runtime.Environment;
import org.kie.api.runtime.EnvironmentName;
import org.kie.api.runtime.manager.RuntimeManager;
import org.kie.api.runtime.manager.RuntimeManagerFactory;

...

// here env is an instance of org.kie.api.runtime.Environment
env.set(EnvironmentName.USE_PESSIMISTIC_LOCKING, true);

// now create your Runtime Manager using this enviornment
RuntimeManager manager =
RuntimeManagerFactory.Factory.get().newPerRequestRuntimeManager(environmen
t);
```

12.3. WORK ITEM

Work Items are persisted as **`workiteminfo`** entities, which persist the state of the particular work item instance on runtime and store the following data:

Table 12.4.

Field	Description	Nullable
workitemid	primary key	false
name	work item name	
processinstanceid	parent Process instance id	false
state	integer representing work item state	false

Field	Description	Nullable
optlock	version number used lock value for optimistic locking	
workitembytearray	binary dataset with work item state (binary blob)	false
creationDate	timestampe on which the work item was created	false

12.4. PERSISTENCE CONFIGURATION

12.4.1. Persistence configuration

Although persistence is not used by default, the dependencies needed are available in the runtime directory as jar files .

Persistence is defined per session and you can define it either using the **`JBPMHelper`** class after you create a session or using the **`JPAKnowledgeService`** to create your session. The latter option provides more flexibility, while **`JBPMHelper`** has a method to create a session, and uses a configuration file to configure this session.

12.4.2. Configuring persistence using JBPMHelper

To configure persistence of your session using JBPMHelper, do the following:

1. Define your application to use an appropriate JBPMHelper session construtor:
    - **`KieSession ksession = JBPMHelper.newKieSession(kbase);`**
    - **`KieSession ksession = JBPMHelper.loadKieSession(kbase, sessionId);`**
2. Configure the persistence in the **`jBPM.properties`** file.

Example 12.1. A sample jBPM.properties file with persistence for the in-memory H2 database

```
# for creating a datasource
persistence.datasource.name=jdbc/jbpm-ds
persistence.datasource.user=sa
persistence.datasource.password=
persistence.datasource.url=jdbc:h2:tcp://localhost/~/jbpm-db
persistence.datasource.driverClassName=org.h2.Driver

# for configuring persistence of the session
persistence.enabled=true
persistence.persistenceunit.name=org.jbpm.persistence.jpa
persistence.persistenceunit.dialect=org.hibernate.dialect.H2Dialec
t

# for configuring the human task service
```

```
taskservice.enabled=true
taskservice.datasource.name=org.jbpm.task
taskservice.transport=mina
taskservice.usergroupcallback=org.jbpm.task.service.DefaultUserGro
upCallbackImpl
```

Any invocations on the session will now trigger the persistance process.

Make sure the datasource is up and running on engine start. If you are running the in-memory H2 database, you can start the database from your application using the **`JBPMHelper.startH2Server();`** method call and register it with the engine using **`JBPMHelper.setupDataSource();`** method call.

12.4.3. Configuring persistence using JPAKnowledgeService

To create your knowledge session and configure its persistence using JPAKnowledgeService, do the following:

1. Define your application to use the knowledge session created by JPAKnowledgeService:

   - Define the session based on a knowledge base, a knowledge session configuration, and an environment. The environment must contain a reference to your Entity Manager Factory:

     ```
     // create the entity manager factory and register it in the
     environment
     EntityManagerFactory emf =
     Persistence.createEntityManagerFactory(
     "org.jbpm.persistence.jpa" );
     Environment env = KnowledgeBaseFactory.newEnvironment();
     env.set( EnvironmentName.ENTITY_MANAGER_FACTORY, emf );

     // create a new knowledge session that uses JPA to store the
     runtime state
     KieSession ksession = JPAKnowledgeService.newKieSession( kbase,
     null, env );
     int sessionId = ksession.getId();

     // invoke methods on your method here
     ksession.startProcess( "MyProcess" );
     ksession.dispose();
     ```

   - Define the session based on a specific session id.

     ```
     // recreate the session from database using the sessionId
     ksession = JPAKnowledgeService.loadKieSession(sessionId, kbase,
     null, env );
     ```

2. Configure the persistence in the **`META-INF/persistence.xml`** file: configure JPA to use Hibernate and the respective database.

   Information on how to configure data source on your application server should be available in the documentation delivered with the application server. For this information for JBoss Enterprise Application Platform, see the *Administration and Configuration Guide* for this product.

Example 12.2. A sample persistence.xml file with persistence for an H2 data source `jdbc/jbpm-ds`

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence
  version="1.0"
  xsi:schemaLocation=
    "http://java.sun.com/xml/ns/persistence
     http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd
     http://java.sun.com/xml/ns/persistence/orm
     http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
  xmlns:orm="http://java.sun.com/xml/ns/persistence/orm"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/persistence">
  <persistence-unit name="org.jbpm.persistence.jpa" transaction-
type="JTA">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>jdbc/jbpm-ds</jta-data-source>
    <mapping-file>META-INF/JBPMorm.xml</mapping-file>
    <class>org.drools.persistence.info.SessionInfo</class>

<class>org.jbpm.persistence.processinstance.ProcessInstanceInfo</c
lass>
    <class>org.drools.persistence.info.WorkItemInfo</class>
    <properties>
      <property name="hibernate.dialect"
value="org.hibernate.dialect.H2Dialect"/>
      <property name="hibernate.max_fetch_depth" value="3"/>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.transaction.manager_lookup_class"

value="org.hibernate.transaction.BTMTransactionManagerLookup"/>
    </properties>
  </persistence-unit>
</persistence>
```

Any invocations on the session will now trigger the persistance process.

Make sure the datasource is up and running on engine start. If you are running the in-memory H2 database, you can start the database from your application using the **`JBPMHelper.startH2Server();`** method call and register it with the engine using **`JBPMHelper.setupDataSource();`** method call.

NOTE

If you are running JBoss BPM Suite in a simple Java environment, your data source configuration will be similar to the following:

```
PoolingDataSource ds = new PoolingDataSource();
ds.setUniqueName("jdbc/jbpm-ds");
ds.setClassName("bitronix.tm.resource.jdbc.lrc.LrcXADataSource"
);
ds.setMaxPoolSize(3);
ds.setAllowLocalTransactions(true);
ds.getDriverProperties().put("user", "sa");
ds.getDriverProperties().put("password", "sasa");
ds.getDriverProperties().put("URL",
"jdbc:h2:tcp://localhost/~/jbpm-db");
ds.getDriverProperties().put("driverClassName",
"org.h2.Driver");
ds.init();
```

CHAPTER 13. TRANSACTIONS

13.1. TRANSACTIONS

The Process Engine supports JTA transactions: local transactions are only supported when using Spring. Pure local transactions are not supported.

By default, each method invocation is considered a transaction. To change this behavior, for example, to combine multiple commands into one transaction, you will need to specify transaction boundaries.

13.2. DEFINING TRANSACTIONS

To define a transaction, do the following:

1. Register the transaction manager in your environment.

   Example 13.1. Code with transaction manager registration

   ```
   // create the entity manager factory
   EntityManagerFactory emf =
   EntityManagerFactoryManager.get().getOrCreate("org.jbpm.persistenc
   e.jpa");
   TransactionManager tm =
   TransactionManagerServices.getTransactionManager();
   Environment env = EnvironmentFactory.newEnvironment();
   env.set(EnvironmentName.ENTITY_MANAGER_FACTORY, emf);
   env.set(EnvironmentName.TRANSACTION_MANAGER, tm);

   // setup the runtime environment
   RuntimeEnvironment environment =
   RuntimeEnvironmentBuilder.Factory.get()
   .newDefaultBuilder()
   .addAsset(ResourceFactory.newClassPathResource("MyProcessDefinitio
   n.bpmn2"), ResourceType.BPMN2)
       .addEnvironmentEntry(EnvironmentName.TRANSACTION_MANAGER, tm)

   .addEnvironmentEntry(EnvironmentName.PERSISTENCE_CONTEXT_MANAGER,
   new JpaProcessPersistenceContextManager(env))

   .addEnvironmentEntry(EnvironmentName.TASK_PERSISTENCE_CONTEXT_MANA
   GER, new JPATaskPersistenceContextManager(env))
       .get();
   ```

2. Initialize the KieSession:

   ```
   // get the KieSession
   RuntimeManager manager =
   RuntimeManagerFactory.Factory.get().newPerProcessInstanceRuntimeMana
   ger(environment);
   RuntimeEngine runtime =
   manager.getRuntimeEngine(ProcessInstanceIdContext.get());
   KieSession ksession = runtime.getKieSession();
   ```

3. Define the transaction manager in **`jndi.properties`**.

Example 13.2. Definition of Bitronix transaction manager in jndi.properties

```
java.naming.factory.initial=bitronix.tm.jndi.BitronixInitialContex
tFactory
```

NOTE

To use a different JTA transaction manager, edit the hibernate.transaction.manager_lookup_class, the transaction manager property, in the **`persistence.xml`** file to load your transaction manager.

Example 13.3. JBoss Transaction Manager set as transaction manager

```
<property
name="hibernate.transaction.manager_lookup_class"
value="org.hibernate.transaction.JBossTransactionManage
rLookup"/>
```

4. Define the start and the end of the transaction.

```
 // start the transaction
UserTransaction ut =
InitialContext.doLookup("java:comp/UserTransaction");
ut.begin();

// perform multiple commands inside one transaction
ksession.insert( new Person( "John Doe" ) );
ksession.startProcess("MyProcess");

// commit the transaction
ut.commit();
```

13.3. CONTAINER MANAGED TRANSACTIONS

In cases where JBoss BPM Suite is embedded inside an application that is in a container that can manage transactions by itself (Container Managed Transactions - CMT), a special dedicated transaction manager is provided using the **`org.jbpm.persistence.jta.ContainerManagerTransactionManager`** class. This is because the default implementation of the transaction manager in JBoss BPM Suite is based on the **`UserTransaction`** class getting the transaction status. However, some application servers in a CMT mode do not allow accessing the **`UserTransaction`** instance from JNDI.

Operations executed on this manager are all no-op because they can't affect the underlying CMT. The **`ContainerManagedTransactionManager`** class expects that the transaction is always active (returning **`ACTIVE`** to the **`getStatus()`** method).

NOTE

Even though the container manages transactions, the container should be made aware of any exceptions that happen during process instance execution. Exceptions thrown by the engine should be propagated up to the container to properly rollback transactions.

Configuring the Transaction Manager

To configure and use the **`ContainerManagedTransactionManager`**, it needs to be inserted into the environment before you create or load a session:

```
    Environment env = EnvironmentFactory.newEnvironment();
    env.set(EnvironmentName.ENTITY_MANAGER_FACTORY, emf);
    env.set(EnvironmentName.TRANSACTION_MANAGER, new
ContainerManagedTransactionManager());
    env.set(EnvironmentName.PERSISTENCE_CONTEXT_MANAGER, new
JpaProcessPersistenceContextManager(env));
```

Next setup the JPA Provider in your **`persistence.xml`** file. For example if using IBM WebSphere:

```
<property name="hibernate.transaction.factory_class"
value="org.hibernate.transaction.CMTTransactionFactory"/>
<property name="hibernate.transaction.manager_lookup_class"
value="org.hibernate.transaction.WebSphereExtendedJTATransactionLookup"/>
```

Disposing the KSession in a CMT

In a CMT, you should not dispose the ksession directly (by using the **`dispose()`** method). Doing so will cause exceptions on transaction completion as the Process Engine needs to clean up the state after the invocation has finished.

Instead, use the specialized class **`org.jbpm.persistence.jta.ContainerManagedTransactionDisposeCommand`**'s **`execute()`** method. Using this command ensures that the ksession will be disposed when the transaction is actually complete.

This method checks to see if the transaction is active. If it is, it delegates the actual disposal to the **`afterDisposal`** phase of the transaction instead of executing it directly. If there is no active transaction or if there is no active transaction, the ksession is disposed immediately.

CHAPTER 14. LOGGING

The logging mechanism allows you to store information about the execution of a process instance. It is provided by a special event listener that listens to the Process Engine for any relevant events to be logged, so that the information can be stored separately from other non-log information stored either in the server built-in database (h2) or a connected data source using JPA or Hibernate.

The jbpm-audit module provides the event listener and also allows you to store process-related information directly in a database using JPA or Hibernate. The data of the following entities is stored as follows:

- Process instance as **`processinstancelog`**
- Element instance as **`nodeinstancelog`**
- Variable instance as **`variableinstancelog`**

Table 14.1. Fields of the ProcessInstanceLog table

Field	Description	Nullable
id	The primary key of the log entity	No
end_date	The end date of the process instance	Yes
processid	The name (id) of the underlying process	Yes
processinstanceid	The id of the process instance	No
start_date	The start date of the process instance	Yes
status	The status of the process instance	Yes
parentProcessInstanceId	The process instance id of the parent process instance if applicable	Yes
outcome	The outcome of the process instance (details on the process finish, such as error code)	Yes

Table 14.2. Fields of the NodeInstanceLog table

Field	Description	Nullable
id	The primary key of the log entity	No
log_date	The date of the event	Yes

Field	Description	Nullable
nodeid	The node id of the underlying Process Element	Yes
nodeinstanceid	The id of the node instance	Yes
nodename	The name of the underlying node	Yes
processid	The id of the underlying process	Yes
processinstanceid	The id of the parent process instance	No
type	The type of the event (**`0`** = enter event, **`1`** = exit event)	No

Table 14.3. Fields of the VariableInstanceLog table

Field	Description	Nullable
id	The primary key of the log entity	No
log_date	The date of the event	Yes
processid	The name (id) of the underlying process	Yes
processinstanceid	The id of the process instance	No
value	The value of the variable at log time	Yes
variableid	The variable id as defined in the process definition	Yes
variableinstanceid	The id of the variable instance	Yes
outcome	The outcome of the process instance (details on the process finish, such as error code)	Yes

If neccessary, define your own data model of custom information and use the process event listeners to extract the information.

14.1. LOGGING EVENTS TO DATABASE

To log an event that occurs on runtime in a Process instance, an Element instance, or a variable instance, you need to do the following:

1. Map the Log classes to the data source, so that the given data source accepts the log entries. On Red Hat JBoss EAP, edit the data source properties in the **`persistence.xml`** file.

Example 14.1. The ProcessInstanceLog, NodeInstanceLog and VariableInstanceLog classes enabled for processInstanceDS

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<persistence  version="1.0"  xsi:schemaLocation=
    "http://java.sun.com/xml/ns/persistence
     http://java.sun.com/xml/ns/persistence/persistence_1_0.xsd
     http://java.sun.com/xml/ns/persistence/orm
     http://java.sun.com/xml/ns/persistence/orm_1_0.xsd"
  xmlns:orm="http://java.sun.com/xml/ns/persistence/orm"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://java.sun.com/xml/ns/persistence">

  <persistence-unit name="org.jbpm.persistence.jpa">
    <provider>org.hibernate.ejb.HibernatePersistence</provider>
    <jta-data-source>jdbc/processInstanceDS</jta-data-source>
    <class>org.drools.persistence.info.SessionInfo</class>

<class>org.jbpm.persistence.processinstance.ProcessInstanceInfo</c
lass>
    <class>org.drools.persistence.info.WorkItemInfo</class>
    <class>org.jbpm.process.audit.ProcessInstanceLog</class>
    <class>org.jbpm.process.audit.NodeInstanceLog</class>
    <class>org.jbpm.process.audit.VariableInstanceLog</class>

    <properties>
      <property name="hibernate.dialect"
value="org.hibernate.dialect.H2Dialect"/>
      <property name="hibernate.max_fetch_depth" value="3"/>
      <property name="hibernate.hbm2ddl.auto" value="update"/>
      <property name="hibernate.show_sql" value="true"/>
      <property name="hibernate.transaction.manager_lookup_class"

value="org.hibernate.transaction.BTMTransactionManagerLookup"/>
    </properties>
  </persistence-unit>
</persistence>
```

2. Register a logger on your Kie Session.

Example 14.2. Import the Loggers

```
import org.jbpm.process.audit.AuditLogService;
import org.jbpm.process.audit.AuditLoggerFactory;
import org.jbpm.process.audit.AuditLoggerFactory.Type;
import org.jbpm.process.audit.JPAAuditLogService;
...
```

Example 14.3. Registering a Logger to a Kie Session

```
@PersistenceUnit(unitName = PERSISTENCE_UNIT_NAME)
  private EntityManagerFactory emf;

  private AuditLogService auditLogService;
@PostConstruct
  public void configure() {

  auditLogService = new JPAAuditLogService(emf);
  ((JPAAuditLogService)
auditLogService).setPersistenceUnitName(PERSISTENCE_UNIT_NAME);

  if( emf == null ) {
  ((JPAAuditLogService)
auditLogService).setPersistenceUnitName(PERSISTENCE_UNIT_NAME);
  }

  RuntimeEngine runtime =
singletonManager.getRuntimeEngine(EmptyContext.get());
  KieSession ksession = runtime.getKieSession();
  AuditLoggerFactory.newInstance(Type.JPA, ksession, null);

  }
```

3. Optionally, call the method **`addFilter`** on the logger to filter out irrelevant information. Only information accepted by all filters appears in the database.

4. Logger classes can be viewed in the Audit View:

```
<dependency>
 <groupId>org.jbpm</groupId>
 <artifactId>jbpm-audit</artifactId>
 <version>6.0.1.Final</version>
</dependency>
```

14.2. LOGBACK FUNCTIONALITY

Red Hat JBoss BPM Suite 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're 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.

14.3. 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 15. LOCALIZATION AND CUSTOMIZATION

15.1. AVAILABLE LANGUAGES

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

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

NOTE

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

15.2. 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`

Changing the User Interface Language in Dashbuilder

To change the user interface language in dashbuilder, do the following:

1. Log into the dashbuilder after the server has been successfully started by navigating to http://localhost:8080/dashbuilder in a web browser.

2. Select the language of your choice by clicking on the available locales on the top center of the dashbuilder user interface to change the language.

Setting a Default User Interface Language in Dashbuilder

Following is an example to set the default user interface language in dashbuilder:

Procedure 15.1. Setting the default language as French

1. Navigate to **`jboss-eap-6.4/standalone/configuration`** and define the following in the **`standalone.xml`** file.

```
<system-properties>
    <property
name="org.jboss.dashboard.LocaleManager.installedLocaleIds"
```

```
value="en,es,de,fr,ja,pt,zh"/>
    <property
name="org.jboss.dashboard.LocaleManager.defaultLocaleId"
value="fr"/>
</system-properties>
```

2. The default user interface language of the dashbuilder is now set to French.

Defining the Installed Locales in Dashbuilder

Following is an example to define the installed locales in dashbuilder:

Procedure 15.2. Defining the installed locale

- Navigate to **`jboss-eap-6.4/standalone/configuration`** and define the following in the **`standalone.xml`** file.

```
<system-properties>
    <property
name="org.jboss.dashboard.LocaleManager.installedLocaleIds"
value="en,es,de,fr,ja,pt"/>
    <property
name="org.jboss.dashboard.LocaleManager.defaultLocaleId"
value="fr"/>
</system-properties>
```

In this example, the Chinese language (zh) has been removed from the list of installed locales so users will not be able to switch the dashbuilder to Chinese. Dashbuilder will show content in French, which is the default locale. Users will be able to select other languages that are defined (en, es, de, ja, pt) in this file.

NOTE

Within Business Central, the application server does not need to be restarted after changing locale if you append the "locale" parameter to the URL of Business Central. However, with Dashbuilder, the application server should be restarted after the configuration files have been changed.

15.3. RUNNING THE JVM WITH UTF-8 ENCODING

Red Hat JBoss BPM Suite 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".

PART IV. EXECUTION

CHAPTER 16. EXECUTION SERVER

16.1. ASSIGNMENT RULES

Assignment rules are rules executed automatically when a Human Task is created or completed. This mechanism can be used, for example, to assign a Human Task automatically to a particular user of a group or prevent a user from completing a Task if data is missing.

16.1.1. Defining assignment rules

To define assignment rules, do the following:

1. Create a file that will contain the rule definition on the Business Central classpath (the recommended location is ***$DEPLOY_DIR*/standalone/deployments/business-central.war/WEB-INF/classes/**):
   - **default-add-task.drl** with the rules to be checked when the Human Task is created
   - **default-complete-task.drl** with the rules to be checked when the Human Task is completed
2. Define the rules in the file.

Example 16.1. The default-add-task.drl content

```
package defaultPackage

import org.kie.api.task.model.Task;
import org.kie.api.task.model.User;
import org.kie.api.task.model.Status;
import org.kie.api.task.model.PeopleAssignments;
import org.jbpm.services.task.rule.TaskServiceRequest;
import org.jbpm.services.task.exception.PermissionDeniedException;
import org.jbpm.services.task.impl.model.*;
import java.util.HashMap;
import java.util.List;


global TaskServiceRequest request;

rule "Don't allow Mary to complete task when rejected"
    when
        $task : Task()
        $actualOwner : User( id == 'mary') from
$task.getTaskData().getActualOwner()
        $params : HashMap(this["approved"] == false)
    then
        request.setAllowed(false);
        request.setExceptionClass(PermissionDeniedException.class);
        request.addReason("Mary is not allowed to complete task with
approved false");
end
```

If the potential owners of a Human Task will contain the user **Mary**, the task will be automatically assigned to the user **mary**.

Example 16.2. The `default-complete-task.drl` content

```
package defaultPackage

import org.kie.api.task.model.Task;
import org.kie.api.task.model.User;
import org.kie.api.task.model.Status;
import org.kie.api.task.model.PeopleAssignments;
import org.jbpm.services.task.rule.TaskServiceRequest;
import org.jbpm.services.task.exception.PermissionDeniedException;
import org.jbpm.services.task.impl.model.*;
import java.util.HashMap;
import java.util.List;


global TaskServiceRequest request;

rule "Don't allow Mary to complete task when rejected"
    when
        $task : Task()
        $actualOwner : User( id == 'mary') from
$task.getTaskData().getActualOwner()
        $params : HashMap(this["approved"] == false)
    then
        request.setAllowed(false);
        request.setExceptionClass(PermissionDeniedException.class);
        request.addReason("Mary is not allowed to complete task without
approval.");
end
```

If the potential owners of a Human Task will contain the user **Mary**, the task will be automatically assigned to the user **mary**.

16.2. MAIL SESSION

Mail session defines the mail server properties that are used for sending emails if required by the application, such as, escalation or notification mechanisms (refer to the *Red Hat JBoss BPM Suite User Guide*).

16.2.1. Setting up mail session

To set up the mail session for your execution engine, do the following:

1. Open the respective profile configuration file (**`standalone.xml`** or **`host.xml`**) for editing.

2. Add the mail session to the **`urn:jboss:domain:mail:1.1`** subsystem.

Example 16.3. New mail session on localhost

```
<subsystem xmlns="urn:jboss:domain:mail:1.1">
    <!-- omitted code -->

    <mail-session jndi-name="java:/mail/bpmsMailSession"
debug="true" from="bpms@company.com">
        <smtp-server outbound-socket-binding-ref="bpmsMail"/>
    </mail-session>
</subsystem>
```

3. Define the session outbound socket in the profile configuration file.

Example 16.4. Outbound socket definition

```
<outbound-socket-binding name="bpmsMail">
    <remote-destination host="localhost" port="12345"/>
</outbound-socket-binding>
```

CHAPTER 17. PLUG-IN FOR RED HAT JBOSS DEVELOPER STUDIO

17.1. PLUG-IN

PART V. MONITORING

CHAPTER 18. PROCESS MONITORING

18.1. JBOSS OPERATIONS NETWORK

A JBoss Operations Network plug-in can be used to monitor rules sessions for Red Hat JBoss . The plug-in uses Java Management Extensions (JMX) to monitor rules sessions.

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 JBoss application via the **`pom.xml`** configuration file.

Please refer to the **JBoss Operations Network** *Installation Guide* for installation instructions for the JBoss ON server.

18.2. 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.

The following procedure guides a user to copy the plug-in JAR files to the JBoss Operations Network plug-in directory

Procedure 18.1. 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-3.2.0.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.GA1/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, following is the procedure

Procedure 18.2. 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 **`Server Plugins`** link.

4. At the bottom of the list of loaded server 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.

18.3. 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 occured in the updated version.

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

CHAPTER 19. MANAGING SECURITY FOR RED HAT JBOSS BPM SUITE DASHBUILDER

19.1. ACCESSING RED HAT JBOSS BPM SUITE DASHBUILDER

Dashbuilder is the Red Hat JBoss BPM Suite web-based user interface for Business Activity Monitoring. To access the Dashbuilder from Business Central, go to **Dashboards → Process & Task Dashboards**.

The displayed dashboard provides statistics on runtime data selected on the left. You can create your own dashboard in the Dashbuilder. To do so, display the Dashbuilder by clicking **Dashboards → Business Dashboards**.

19.2. MANAGING SECURITY

To manage security, you can define custom authorization policies to grant or deny access to workspace, page, or panel instances per role.

Defined below is a list of the available roles for Dashbuilder:

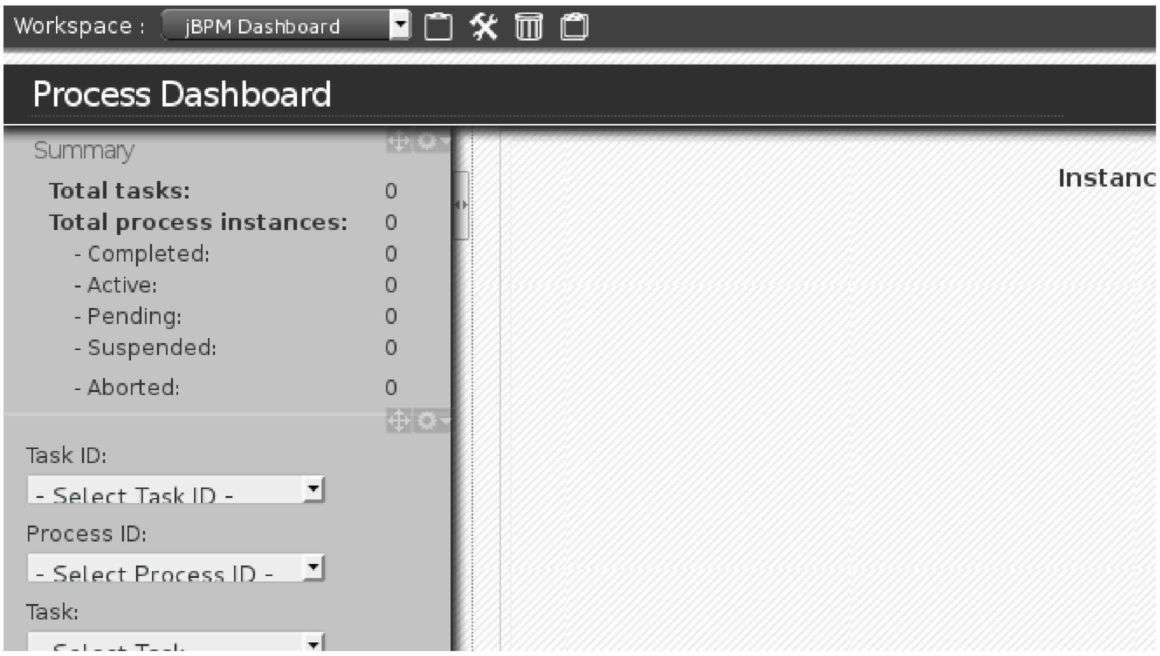
- admin - Administrates the Red Hat JBoss BPM Suite system. Has full access rights to make any changes necessary. Also has the ability to add and remove users from the system.
- developer - Implements code required for process to work. Mostly uses the JBDS connection to view processes, but may use the web tool occasionally.:
- analyst - Responsible for creating and designing processes into the system. Creates process flows and handles process change requests. Needs to test processes that they create. Also creates forms and dashboards.
- user - Daily user of the system to take actions on business tasks that are required for the processes to continue forward. Works primarily with the task lists.
- manager - Viewer of the system that is interested in statistics around the business processes and their performance, business indicators, and other reporting of the system and people who interact with the system.

Thanks to the permissions system, you can build a workspace structure with several pages, menus, and panels and define what pages and panels within a page will be visible for each role. You can also define special types of users and give them restricted access to certain tooling features, or even restricted access to a page subset.

19.3. WORKSPACE PERMISSIONS

Procedure 19.1. Accessing Workspace Permissions

1. Log into Business Dashboards from Business Central (as described in the Accessing Red Hat JBoss BPM Suite Dashbuilder topic).
2. Select the appropriate Dashboard from the Wokspace drop-down.


Workspace : jBPM Dashboard

Process Dashboard

Summary

Total tasks:	0
Total process instances:	0
- Completed:	0
- Active:	0
- Pending:	0
- Suspended:	0
- Aborted:	0

Task ID: - Select Task ID -

Process ID: - Select Process ID -

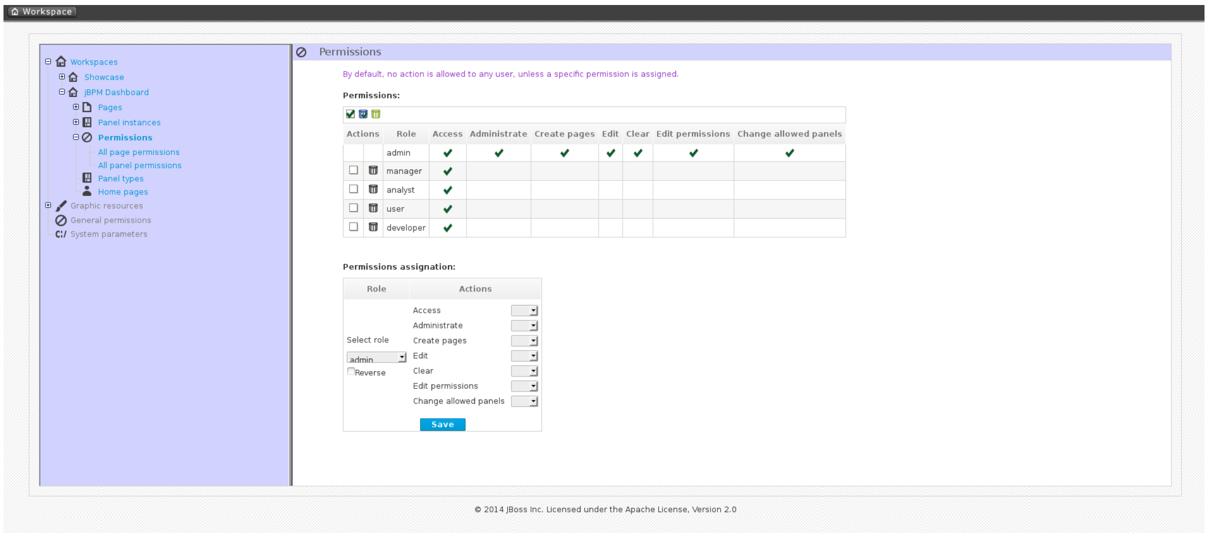
Task:

Instanc

Figure 19.1. Dashbuilder Workspace

3. Click the **`Edit selected workspace properties`** button to access the Workspace Dashboard.

4. Click the **`Permissions`** label to view the permission management screen.



Permissions

By default, no action is allowed to any user, unless a specific permission is assigned.

Permissions:

Actions		Role	Access	Administrate	Create pages	Edit	Clear	Edit permissions	Change allowed panels
		admin	✓	✓	✓	✓	✓	✓	✓
☐		manager	✓						
☐		analyst	✓						
☐		user	✓						
☐		developer	✓						

Permissions assignation:

Role	Actions
Select role: admin; ☐ Reverse	Access, Administrate, Create pages, Edit, Clear, Edit permissions, Change allowed panels

Save

© 2014 JBoss Inc. Licensed under the Apache License, Version 2.0

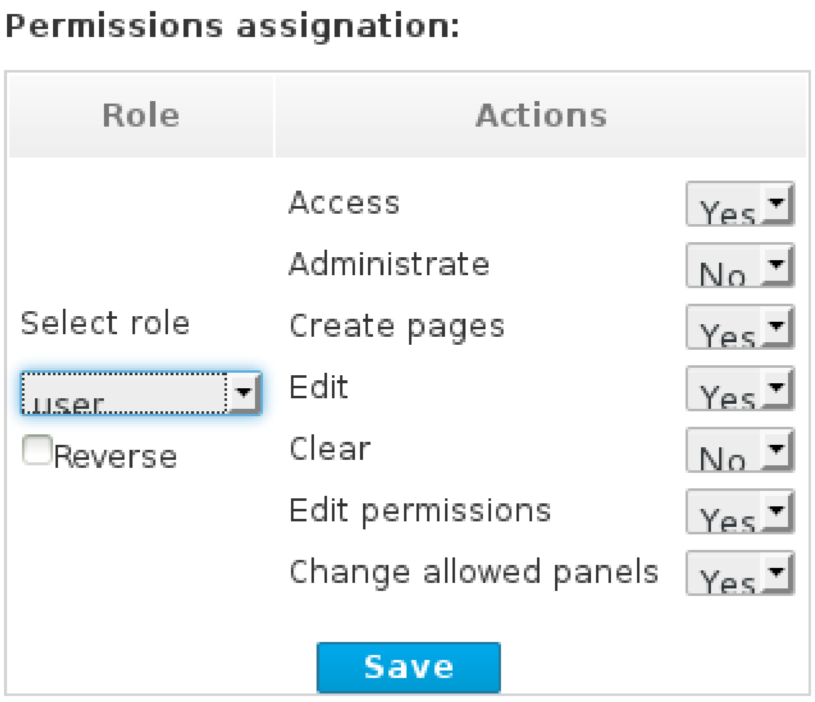
Figure 19.2. Permissions Screen

Under the **`Permissions assignation`** section is a list of allowed actions that are applied to the selected role:

- **Access**: permission to login into the application.
- **Administrate**: permission to access the toolbar and system configuration features.
- **Create pages**: ability to create new project pages.
- **Edit**: permission to change the workspace properties.
- **Clear**: ability to delete the workspace.

- **Edit permissions**: ability to grant/deny permissions.
- **Change allowed panels**: permission to restrict the type of panels that can be used in this workspace.

To assign a permission you must select the target role and the list of actions allowed over the selected resource.


Permissions assignation:

Role	Actions	
	Access	Yes
	Administrate	No
Select role	Create pages	Yes
user	Edit	Yes
☐ Reverse	Clear	No
	Edit permissions	Yes
	Change allowed panels	Yes

Figure 19.3. Permissions Assignation

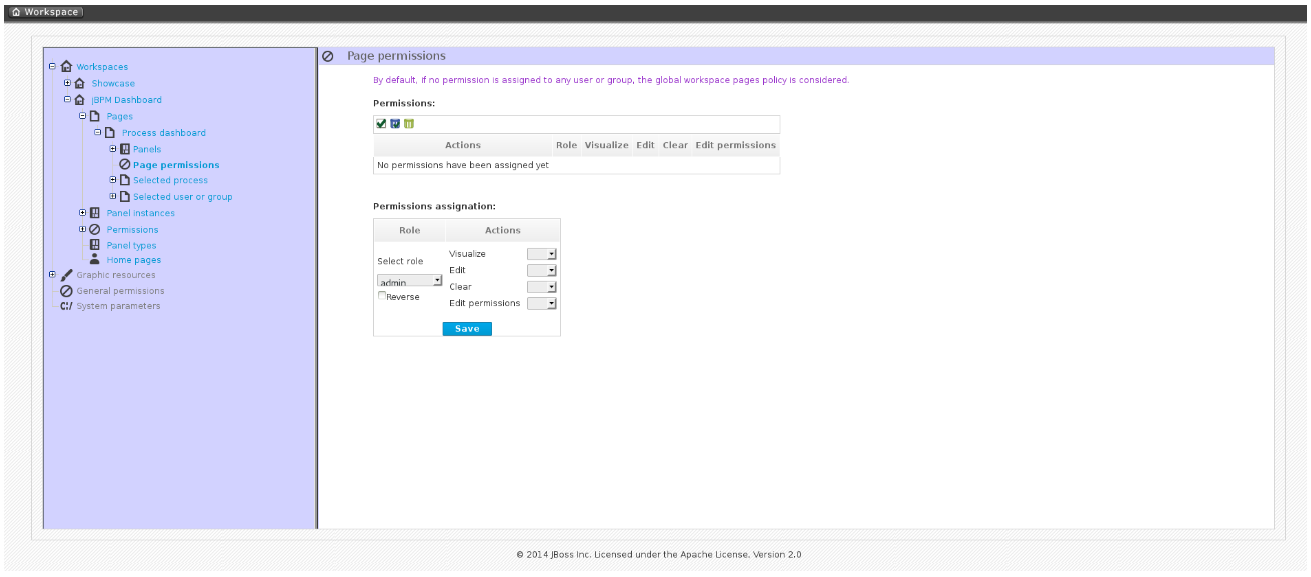
- *Target roles (who)*: What user will be granted/denied with the permissions defined.
- *Allowed actions*: depending on the type of the resource we can enable/disable what the user can do on this resource.
- *Reverse (optional)*: when we have a set of roles and we want to grant/deny a permission to all the roles but one.

NOTE

By default, the full set of permissions go to the role *admin*. This makes it easy to create a user that can do everything as long as the role *admin* is assigned.

19.4. PAGE PERMISSIONS

1. To access **`Page permissions`**, locate the **`Pages`** drop-down under the jBPM Dashboard (or whatever Dashboard you selected).
2. After expanding **`Pages`**, expand the **`Process dashboard`** option.
3. Select the **`Page permissions`** option.



Page permissions

By default, if no permission is assigned to any user or group, the global workspace pages policy is considered.

Permissions:

Actions	Role	Visualize	Edit	Clear	Edit permissions
No permissions have been assigned yet					

Permissions assignation:

Role	Actions
Select role: admin; Reverse	Visualize; Edit; Clear; Edit permissions

Save

© 2014 JBoss Inc. Licensed under the Apache License, Version 2.0

Figure 19.4. Page Permissions

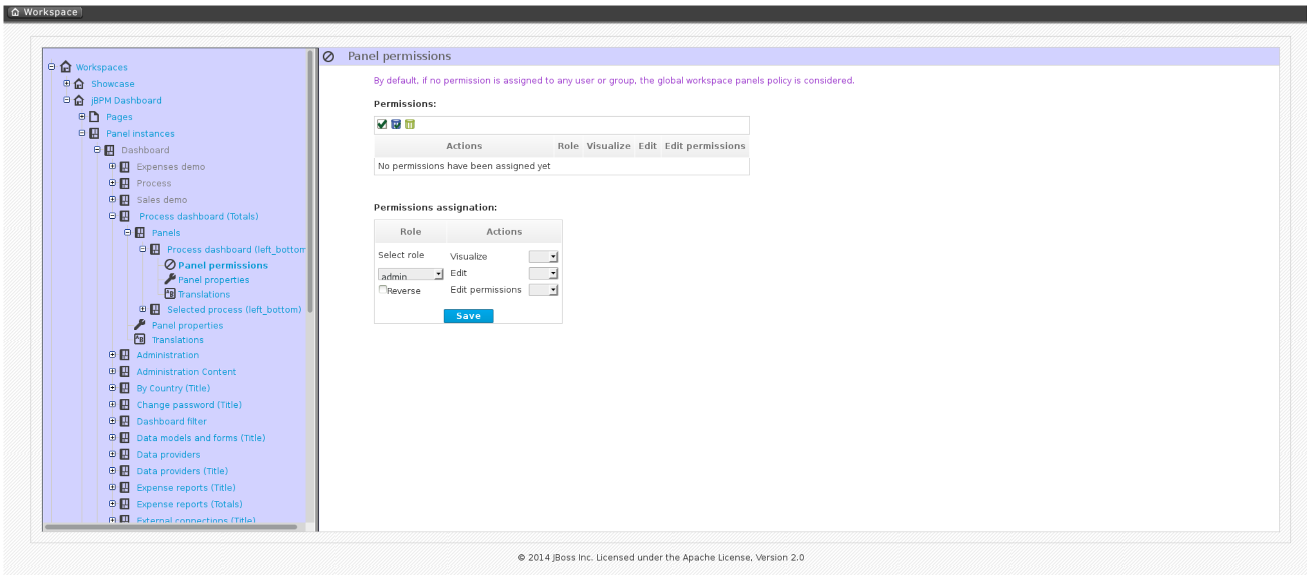
Under the **`Permissions assignation`** section is a list of allowed actions that are applied to the selected role:

- **Visualize**: permission to make the page visible.
- **Edit**: ability to change the page properties.
- **Clear**: ability to delete the page.
- **Edit permissions**: ability to grant/deny permissions for the page.

19.5. PANEL PERMISSIONS

1. To access the **`Panel permissions`** page, expand the **`Panel instances`** option under the jBPM Dashboard (or whatever Dashboard you are using).
2. Expand the **`Dashboard`** option and then expand the **`Process dashboard`**.
3. Expand the **`Panels`** choice and select the appropriate process.
4. Open the **`Panel permissions`** page.

Below is a screenshot of the permission management screen for a given panel (in this example, the Process dashboard):



Panel permissions

By default, if no permission is assigned to any user or group, the global workspace panels policy is considered.

Permissions:

Actions	Role	Visualize	Edit	Edit permissions
No permissions have been assigned yet				

Permissions assignation:

Role	Actions	
Select role	Visualize	
admin	Edit	
Reverse	Edit permissions	

© 2014 JBoss Inc. Licensed under the Apache License, Version 2.0

Figure 19.5. Panel permissions configuration screen

Allowed actions are the following:

- **Visualize**: make the panel visible.
- **Edit**: change the panel properties.
- **Edit permissions**: ability to grant/deny permissions for the panel.

APPENDIX A. REVISION HISTORY

Revision 1.0.0-56	**Thu Dec 17 2015**	**Vidya Iyengar**

Build includes various enhancements and fixes