Red Hat JBoss BPM Suite 6.3

User Guide

The User Guide for Red Hat JBoss BPM Suite

Last Updated: 2017-11-16

Red Hat JBoss BPM Suite 6.3 User Guide

The User Guide for Red Hat JBoss BPM Suite

Red Content Services

Gemma Sheldon
gsheldon@redhat.com

Klara Kufova
kkufova@redhat.com

Marek Czernek
mczernek@redhat.com

Tomas Radej
tradej@redhat.com

Vidya Iyengar
viyengar@redhat.com

Legal Notice

Abstract

A guide to defining and managing business processes with Red Hat JBoss BPM Suite.

Table of Contents

CHAPTER 1. INTRODUCTION

Red Hat JBoss BPM Suite is an open source business process management suite that combines Business Process Management and Business Rules Management and enables business and IT users to create, manage, validate, and deploy Business Processes and Rules.

To accommodate Business Rules component, JBoss BPM Suite includes integrated Red Hat JBoss BRMS.

Red Hat JBoss BRMS and Red Hat JBoss BPM Suite use a centralized repository where all resources are stored. This ensures consistency, transparency, and the ability to audit across the business. Business users can modify business logic and business processes without requiring assistance from IT personnel.

Business Resource Planner is also included with this release.

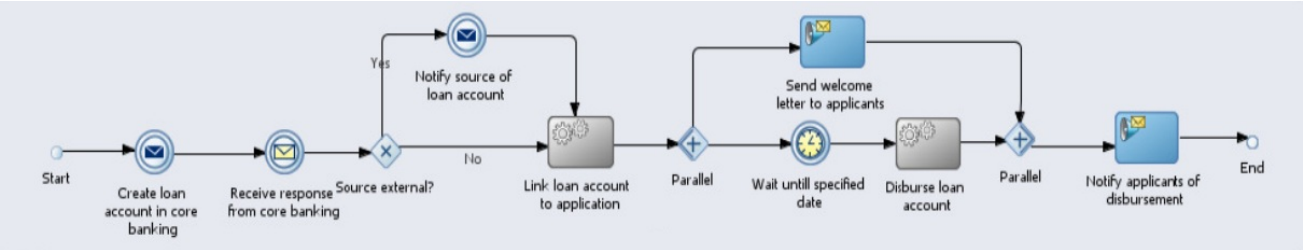
1.1. USE CASE: PROCESS-BASED SOLUTIONS IN THE LOAN INDUSTRY

This section describes a use case of deploying JBoss BPM Suite to automate business processes (such as loan approval process) at a retail bank. This use case is a typical process-based specific deployment that might be the first step in a wider adoption of JBoss BPM Suite throughout an enterprise. It leverages features of both business rules and processes of JBoss BPM Suite.

A retail bank offers several types of loan products each with varying terms and eligibility requirements. Customers requiring a loan must file a loan application with the bank. The bank then processes the application in several steps, such as verifying eligibility, determining terms, checking for fraudulent activity, and determining the most appropriate loan product. Once approved, the bank creates and funds a loan account for the applicant, who can then access funds. The bank must be sure to comply with all relevant banking regulations at each step of the process, and has to manage its loan portfolio to maximize profitability. Policies are in place to aid in decision making at each step, and those policies are actively managed to optimize outcomes for the bank.

Business analysts at the bank model the loan application processes using the BPMN2 authoring tools (Process Designer) in JBoss BPM Suite. Here is the process flow:

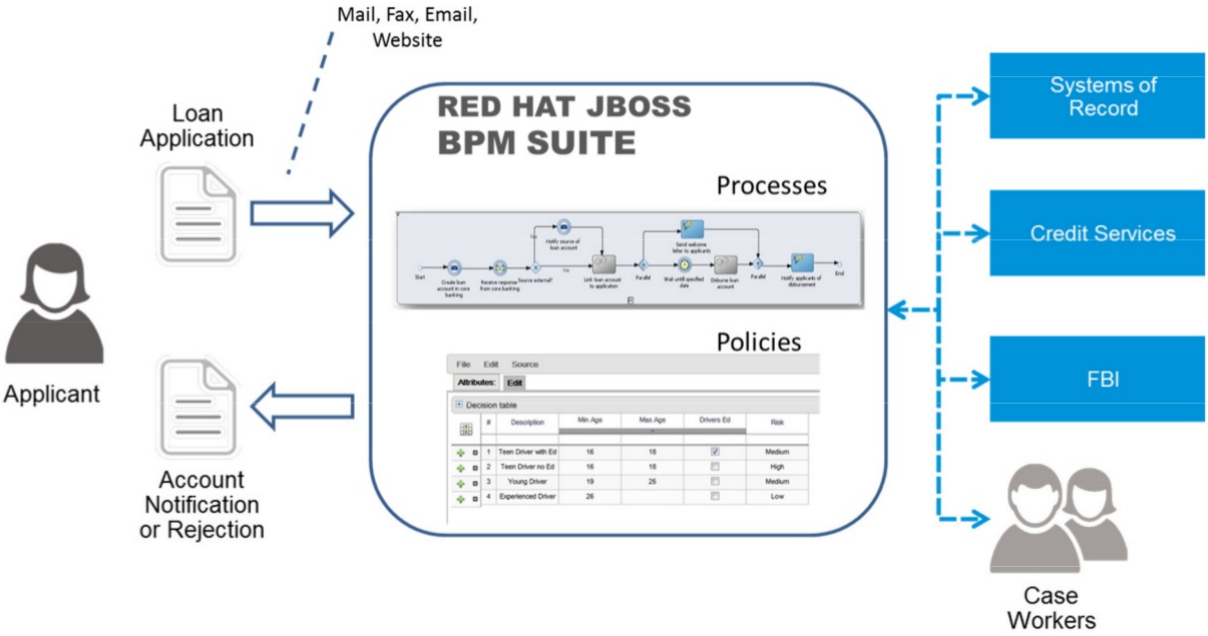
Figure 1.1. High-level loan application process flow



Business rules are developed with the rule authoring tools in JBoss BPM Suite to enforce policies and make decisions. Rules are linked with the process models to enforce the correct policies at each process step.

The bank’s IT organization deploys the JBoss BPM Suite so that the entire loan application process can be automated.

Figure 1.2. Loan Application Process Automation



The entire loan process and rules can be modified at any time by the bank's business analysts. The bank is able to maintain constant compliance with changing regulations, and is able to quickly introduce new loan products and improve loan policies in order to compete effectively and drive profitability.

1.2. COMPONENTS

Red Hat JBoss BPM Suite has the following components:

- *Business Central*, which is a web-based application (**`business-central.war`** and **`dashbuilder.war`**) and provides tools for creating, editing, building, managing, and monitoring of business assets as well as a Task client
- *Artifact repository* (Knowledge Store), which is the set of data the application operates over and is accessed by the Execution Server
- *Execution Server*, which provides the runtime environment for business assets

A more detailed description of components is available in the *Red Hat JBoss BPM Suite Administration and Configuration Guide* .

1.3. RED HAT JBOSS BPM SUITE AND BRMS

Red Hat JBoss BPM Suite comes with integrated Red Hat JBoss BRMS, a rule engine and rule tooling, so you can define rules governing Processes or Tasks. Based on a Business Rule Task call, the Process Engine calls the Rule Engine to evaluate the rule based on specific data from the Process instance. If the defined rule condition is met, the action defined by the rule is taken (see Section A.9.3.7, "Business Rule Task" and the Red Hat JBoss BRMS documentation for further information).

1.4. BUSINESS CENTRAL

Business Central is a web console that allows you to operate over individual components in a unified web-based environment: to create, manage, and edit your Processes, to run,

manage, and monitor Process instances, generate reports, and manage the Tasks produced, as well as create new Tasks and notifications.

- Process management capabilities allow you to start new process instances, acquire the list of running process instances, inspect the state of a specific process instances, etc.
- User Task management capabilities allow you to work with User Tasks; claim User Tasks, complete Tasks through Task forms, etc.

Business Central integrates multiple tools:

- *Process Designer and other editors* for modeling Processes and their resources (form item editor, work item editor, data model editor, etc.), as well as process model simulation tools (see Chapter 4, *Process Designer*).
- *Rules Modeler* for designing Business Rules models and their resources (see the Red Hat JBoss BRMS documentation).
- *Task client* for managing and creating User Tasks (seeSection 11.4, “Task Management”).
- *Process Manager* for managing process instances (seeSection 11.2.2, “Process Instances”).
- *Dashboard Builder*, the BAM component, for monitoring and reporting (see Chapter 14, *Red Hat JBoss Dashboard Builder*).
- *Business Asset Manager* for accessing the Knowledge Repository resources, building and deploying business assets (see Chapter 3, *Project*).
  Artifact repository (Knowledge Store) is the set of data over which Business Central operates. It provides a centralized store for your business knowledge, which can consist of multiple repositories with business assets and resources.

  Apart from the project assets, you can also manage your pom artifacts (such as parent **`pom.xml`** files for kjars) from Business Central’s**Artifact repository**, in case you do not have a separate repository to manage your artifacts. You can further create a child project to extend the uploaded pom artifact by adding the <parent>PARENT_GAV</parent> tag to **`pom.xml`** of the given child project. Here, **`PARENT_GAV`** denotes group, artifact and version of the previously uploaded pom artifact.

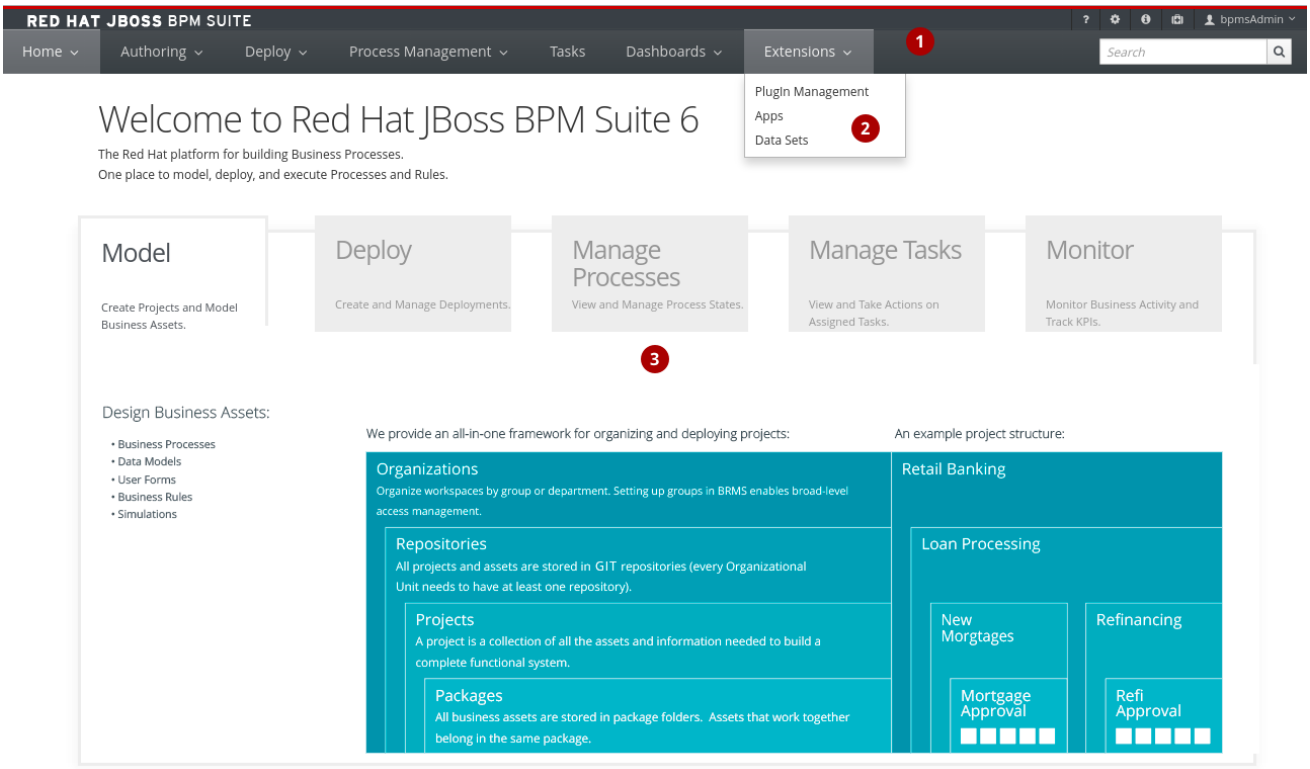
Business Central can be accessed from your web browser on **`https://$HOSTNAME/business-central`** (for instances running on localhost **`https://localhost:8080/business-central`**).

The tools are accessible from the *Views* and *BPM* menus on the main menu:

- **Process Definitions** displays the **Process Definition List** with the Process definitions available in the connected repository.
- **Process Instances** displays the **Process Instance List** with the Process instances currently running on the Process Engine.
- **Tasks** displays a view of the Tasks list for the currently logged-in user. You can call a Task List in the grid view or in the calendar view from the menu:*BPM* menu.

1.4.1. Business Central Environment

Figure 1.3. Home page



The main menu contains the links to the **`Home`** page and all available perspectives.

The perspective menu contains menus for the selected perspective.

The perspective area contains the perspective tools (here the home page with links to individual perspectives and their views), such as views and editors.

1.4.2. Perspectives

Business Central provides the following groups of perspectives accessible from the main menu:

- **Authoring** group:
  - **Project Authoring** perspective contains the **Project Explorer** view (by default on the left) with the overview of available repository structure, and information on available resources, such as, business process definitions, form definitions, etc.; the editor area on the right, where the respective editor appears when a resource is opened; and the **Messages** view with validation messages.
  - **Artifact Repository** perspective contains a list of jars which can be added as dependencies. The available operations in this perspective are upload/download project artifacts and pom-packaged artifacts (**`pom.xml`**).
  - **Administration** perspective (available only for users with the **`ADMIN`** role) contains the **File Explorer** view (by default on the left) with available asset repositories; the editor area on the right, where the respective editor appears

when a resource is opened. The perspective allows an administrator to connect Knowledge Store to a repository with assets and to create a new repository (see the *Red Hat JBoss BPM Suite Administration and Configuration Guide*).

- **Deploy** group:
  - **Deployments** perspective contains a list of the deployed resources and allows you to build and deploy an undeploy new units.
- **Process Management** group:
  - **Process Definitions** perspective contains a list of the deployed Process definitions. It allows you to instantiate and manage the deployed Processes.
  - **Process Instances** perspective contains a list of the instantiated Processes. It allows you to view their execution workflow and its history.
- **Tasks** group:
  - **Task List** perspective contains a list of Tasks produced by Human Task of the Process instances or produced manually. Only Tasks assigned to the logged-in user are visible. It allows you to claim Tasks assigned to a group you are a member of.
- **Dashboards** group (the BAM component):
  - **Process & Task Dashboard** perspective contains a prepared dashboard with statistics on runtime data of the Execution Server
  - **Business Dashboards** perspective contains the full BAM component, the Dashbuilder, including administration features available for users with the **`ADMIN`** role.

1.4.3. Embedding Business Central

Business Central provides a set of editors to author assets in different formats. A specialized editor is used according to the asset format.

Business Central provides the ability to embed it in your own (Web) Applications using standalone mode. This allows you to edit rules, processes, decision tables, et cetera, in your own applications without switching to Business Central.

In order to embed Business Central in your application, you will need the Business Central application deployed and running in a web/application server and, from within your own web applications, an iframe with proper HTTP query parameters as described in the following table.

Table 1.1. HTTP Query Parameters for Standalone Mode

Parameter Name	Explanation	Allow Multiple Values	Example
standalone	This parameter switches Business Central to standalone mode.	no	(none)

Parameter Name	Explanation	Allow Multiple Values	Example
path	Path to the asset to be edited. Note that asset should already exists.	no	git://master@uf-playground/todo.md
perspective	Reference to an existing perspective name.	no	org.guvnor.m2repo.client.perspectives.GuvnorM2RepoPerspective
header	Defines the name of the header that should be displayed (useful for context menu headers).	yes	ComplementNavArea

The following example demonstrates how to set up an embedded Author Perspective for Business Central.

```
===test.html===
 <html>
  <head>
    <title>Test</title>
  </head>
  <body>
    <iframe id="ifrm" width="1920" height="1080"
src='http://localhost:8080/business-central?
standalone=&perspective=AuthoringPerspective&header=AppNavBar'></iframe>
  </body>
</html>
```

X-frame options can be set in **`web.xml`** of business-central. The default value for**`x-frame-options`** is as follows:

```
<param-name>x-frame-options</param-name>
  <param-value>SAMEORIGIN</param-value>
```

CHAPTER 2. BASIC CONCEPTS

Red Hat JBoss BPM Suite provides tools for creating, editing, running, and runtime management of BPMN process models. The models are defined using the BPMN2 language, either directly in its XML form or using visual BPMN Elements that represent the Process workflow (see Chapter 4, *Process Designer*). Alternatively, you can create Processes from your Java application using the JBoss BPM Suite API. Some of these capabilities can be used also via REST API (See *Red Hat JBoss BPM Suite Developer Guide*).

Process models serve as templates for Process instances. To separate the static Process models from their dynamic runtime versions (Process instances), they live in two different entities: Process models live in a Kie Base (or Knowledge Base) and their data cannot be changed by the Process Engine; Process instances live in a Kie Session(or Knowledge Session) which exists in the Process Engine and contains the runtime data, which are changed during runtime by the Process Engine.

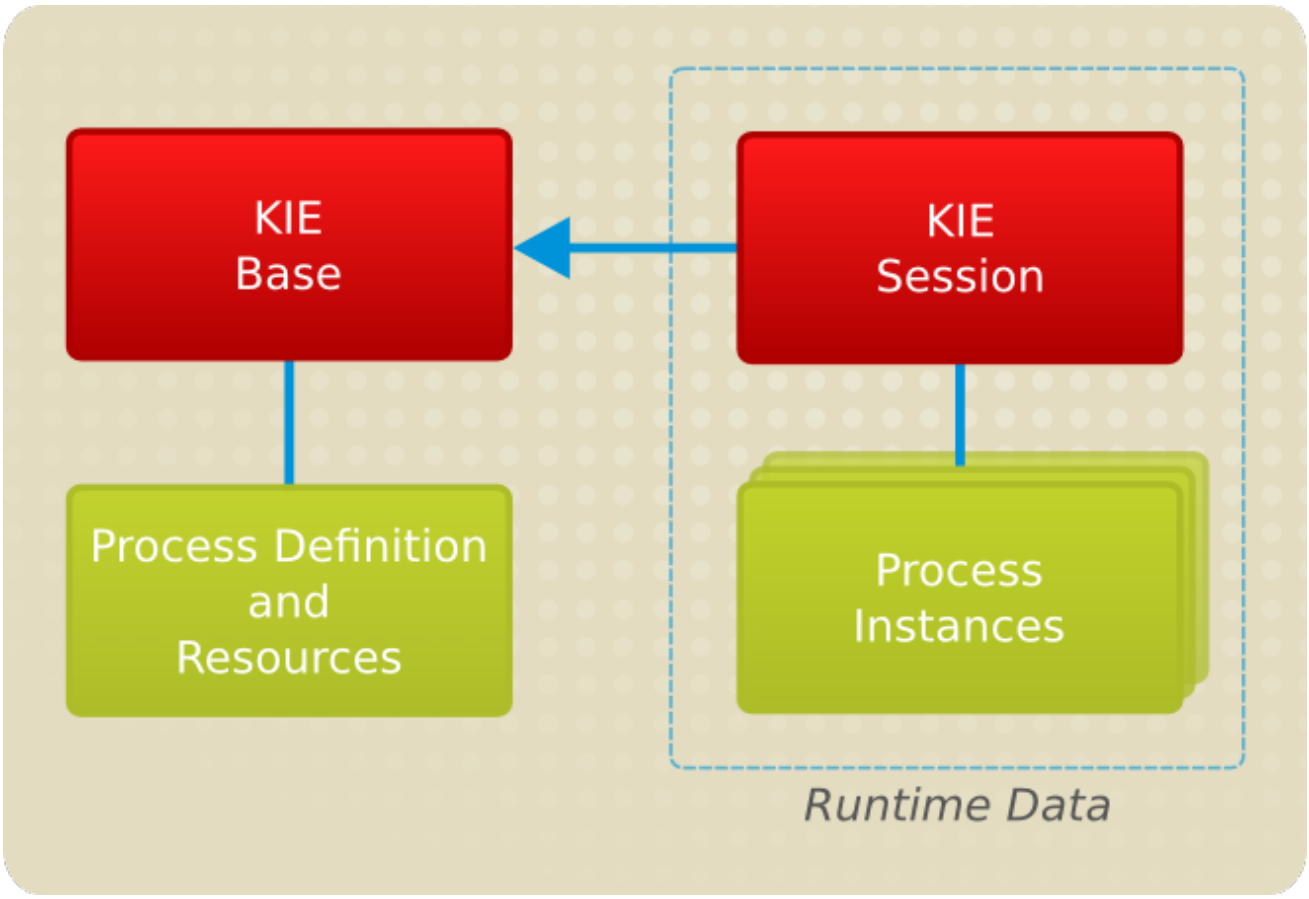
You can define a Kie Base and its Kie Session in the Project Editor of the GUI application (see Section 3.3, “Defining KIE Bases and Sessions”).

Note that a single Kie Base can be shared across multiple Kie Sessions. When instantiating a Kie Base using the respective API call it is usual to create one Kie Base at the start of your application as creating a Kie Base can be rather heavy-weight as it involves parsing and compiling the process definitions. From the Kie Base, you can then start multiple Kie Sessions. The underlying Kie Bases can be changed at runtime so you can add, remove, or migrate process definitions.

To have multiple independent processing units, it might be convenient to create multiple Kie Sessions on the particular Kie Base (for example, if you want all process instances from one customer to be independent from process instances for another customer; multiple Sessions might be useful for scalability reasons as well).

A Kie Session can be either stateful or stateless. Stateful sessions are long-living sessions with explicit call to dispose them; if the **`dispose()`** call is not issued, the session remains alive and causes memory leaks. Also note that the FireAllRules command is not automatically called at the end of a stateful session.

Figure 2.1. Kie Base and Kie Session relationship



PART I. MODELING

CHAPTER 3. 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.

3.1. CREATING A PROJECT

It is possible to create a project either in the **Project Authoring** perspective of Business Central or using the REST API calls.

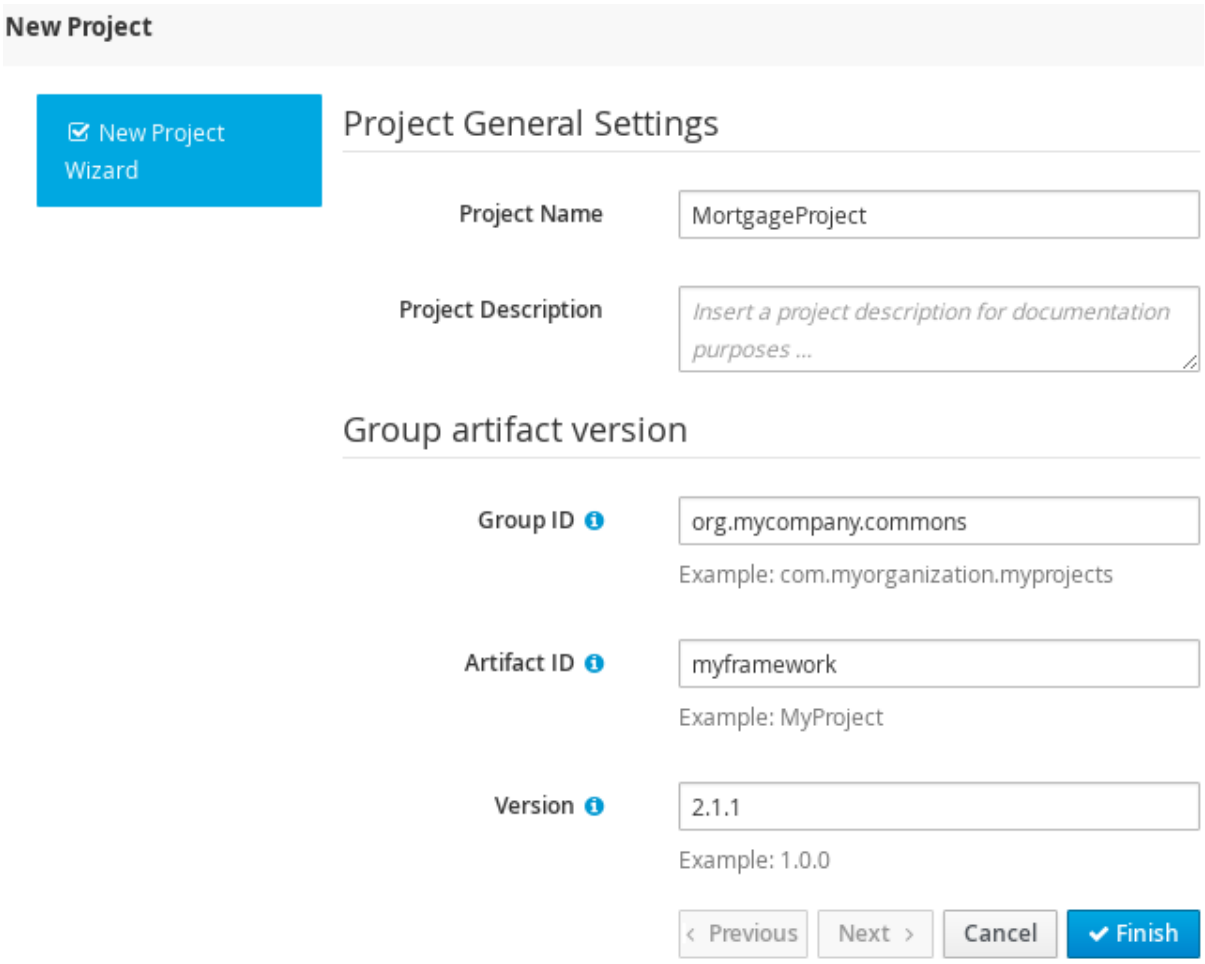
Creating a Project in Business Central

IMPORTANT

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

Procedure: Using Business Central to Create a Project

1. In Business Central, go to **Authoring** → **Project Authoring**.
2. In the **Project Explorer**, select the organizational unit and the repository in which you want to create the project.
3. On the perspective menu, click **New Item** → **Project**.
   The **New Project** dialog window opens.

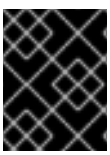


4. Define the **Project General Settings** and **Group artifact version** details of the new project. These parameters are stored in the **pom.xml** Maven configuration file. See the detailed description of the parameters:

   - **Project Name**: name of the project (for example **MortgageProject**).
   - **Project Description**: description of the project, which may be useful for the project documentation purposes.
   - **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 other special character that might lead to an invalid name.
   - **Version**: version of the project (for example **2.1.1**).

5. Click **Finish**.
   The project screen view is updated with the new project details as defined in the **pom.xml** file. You can switch between project descriptor files and edit their content by clicking the **Project Settings: Project General Settings** button at the top of the project screen view.

Creating a Project Using the REST API



IMPORTANT

Note that only users with the **rest-all** or **rest-project** role can create projects.

To create a project in the repository, issue the **POST** REST API call. Details of the project are defined by the corresponding JSON entity.

Input parameter of the call is an **Entity** instance. The call returns a **CreateProjectRequest** instance.

Example 3.1. Creating a Project Using the Curl Utility

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

```
{
 "name"        : "MortgageProject",
 "description" :  null,
 "groupId"     : "org.mycompany.commons",
 "version"     : "2.1.1"
}
```

Execute the following command:

```
curl -X POST 'localhost:8080/business-
central/rest/repositories/REPOSITORY_NAME/projects/' -u
USERNAME:PASSWORD -H 'Accept: application/json' -H 'Content-Type:
application/json' -d
'{"name":"MortgageProject","description":null,"groupId":"org.mycompany.c
ommons","version":"2.1.1"}'
```

For further information, see the *Repository Calls* section of the *Knowledge Store REST API* chapter in the *Red Hat JBoss BPM Suite Development Guide*

3.2. 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 **Open Project Editor** 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):
   a. When adding a maven dependency, a user has to define the **Group ID**, **Artifact ID**, and the **Version ID** in the **Dependency** dialogue window.
   b. 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.

Additionally, you can use the **Package white list** when working with dependencies. When you add a repository, you can click the gear icon and select **Add all** or **Add none**, which results in including all or none of the packages from the added dependency.

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.

3.3. DEFINING KIE BASES AND SESSIONS

A *KIE base* is a repository of the application's knowledge definitions. It contains rules, processes, functions, and type models. A KIE base does not contain runtime data, instead sessions are created from the KIE base into which data can be inserted and process instances started.

A *KIE session* stores runtime data created from a KIE base. See the KIE Sessions chapter of the *Red Hat JBoss BPM Suite Development Guide* for more information.

You can create KIE bases and sessions by editing the **`kmodule.xml`** project descriptor file of your project. You can do so through Business Central or by editing **`kmodule.xml`** in the **`src/main/resources/META-INF/`** folder by navigating through the **`Repository`** view.

Defining KIE Bases and Sessions in the Project Editor

To define a KIE base or session in Business Central, do the following:

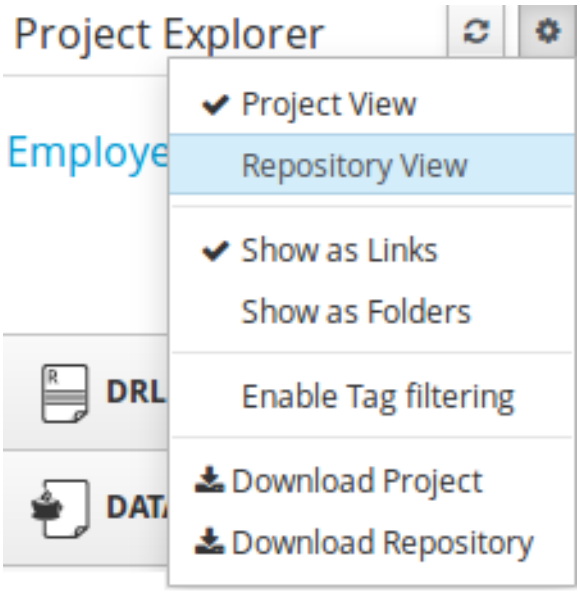
1. Click **Authoring** → **Project Authoring** and navigate to your project.
2. In the **Project Explorer** window, click **Open Project Editor**.
3. Click **Project Settings: Project General Settings** → **Knowledge bases and sessions**. This view provides a user interface for changing **`kmodule.xml`**.
4. Click **Add** to define and add your bases.
    a. After you enter a name for your Knowledge Base, add Packages. For including all packages, click **Add** below **Packages** and enter asterisk *****.
5. Below **Knowledge Sessions**, click **Add** and enter the name of your session.
6. Mark it **`Default`** and select appropriate state. For Red Hat JBoss BPM Suite, use **`stateful`** sessions.
7. Click **Save** in the top right corner once you are done.

Defining KIE Bases and Sessions in kmodule.xml

To define a KIE base or session by editing **`kmodule.xml`**, do the following:

1. Open the repository view for your project.

Figure 3.1. Changing to Repository View



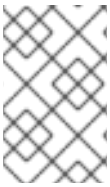
2. Navigate to **/src/main/resources/META-INF**. Click on **kmodule.xml** to edit the file directly.

3. Define your **kbases** and **ksessions**. For example:

```
<kmodule xmlns="http://www.drools.org/xsd/kmodule"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <kbase name="myBase" default="true" eventProcessingMode="stream"
equalsBehavior="identity" packages="*">
    <ksession name="mySession" type="stateless" default="true"
clockType="realtime"/>
  </kbase>
</kmodule>
```

4. Click **Save** in the top right corner.

You can switch between the Project Editor view and the Repository view to look at the changes you make in each view. To do so, close and reopen the view each time a change is made.



NOTE

If you have more than one knowledge base, one of them must be marked *default*. You also must define one default stateful knowledge session amongst all the bases and sessions. Alternatively, you can define no knowledge bases.

3.4. CREATING A RESOURCE

A Project may contain an arbitrary number of packages, which contain files with resources, such as Process definition, Work Item definition, Form definition, Business Rule definition, etc.

To create a resource, select the Project and the package in the **Project Explorer** and click **New Item** on the perspective menu and select the resource you want to create.

CREATING PACKAGES

It is recommended to create your resources, such as Process definitions, Work Item definitions, Data Models, etc., inside a package of a Project to allow importing of resources and referencing their content.

To create a package, do the following:

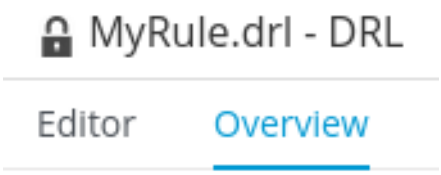
1. In the **Repository** view of the Project Explorer, navigate to the **`REPOSITORY/PROJECT/src/main/resources/`** directory.
2. Go to **New Item** → **Package**.
3. In the **New resource** dialog, define the package name and check the location of the package in the repository.

3.5. ASSET METADATA AND VERSIONING

Most assets within Business Central have some metadata and versioning information associated with them. In this section, we will go through the metadata screens and version management for one such asset (a DRL asset). Similar steps can be used to view and edit metadata and versions for other assets.

Metadata Management

To open up the metadata screen for a DRL asset, click on the **Overview** tab. If an asset does not have an **Overview** tab, it means that there is no metadata associated with that asset.



The **Overview** section opens up in the**Version history** tab, and you can switch to the actual metadata by clicking on the **Metadata** tab.

The metadata section allows you to view or edit the **`Categories`**, **`Subject`**, **`Type`**, **`External Link`** and **`Source metadata`** for that asset. However, the most interesting metadata is the description of the asset that you can view/edit in the description field and the comments that you and other people with access to this asset can enter and view.

Comments can be entered in the text box provided in the comments section. Once you have finished entering a comment, press enter for it to appear in the comments section.

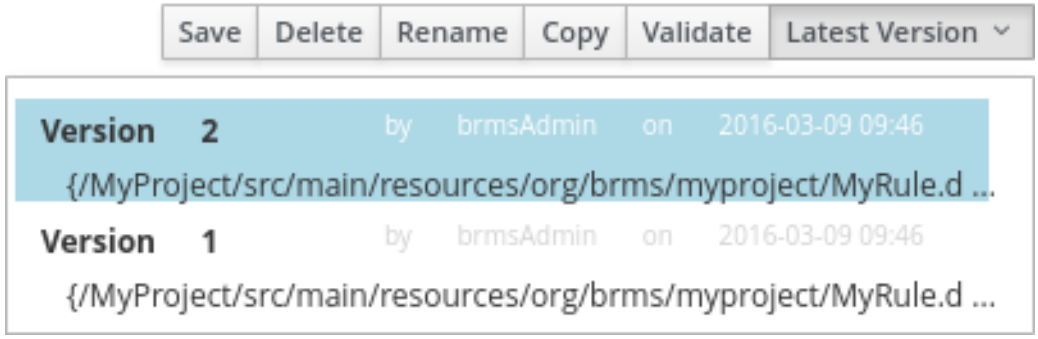
IMPORTANT

You must hit the **Save** button for all metadata changes to be persisted, including the comments.

Version Management

Every time you make a change in an asset and save it, a new version of the asset is created. You can switch between different versions of an asset in one of two ways:

- Click the **Latest Version** button in the asset toolbar and select the version that you are interested in. Business Central will load this version of the asset.



- Alternatively, open up the **Overview** section. The **Version history** section shows you all the available versions. **Select** the version that you want to restore.

In both cases, the **Save** button will change to **Restore** . Click this button to persist changes.

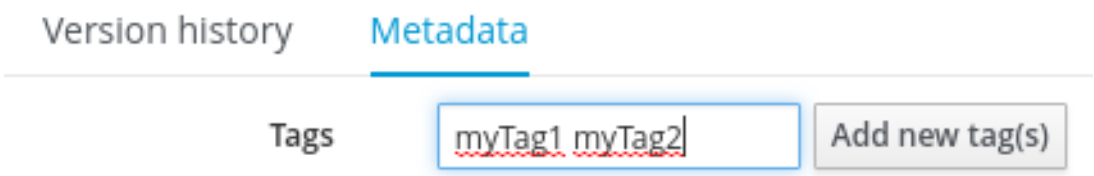
3.6. FILTERING ASSETS BY TAG

It is possible to group assets of similar categories in the project explorer. This feature helps you search through assets of a specific category quickly. To enable this, the metadata management feature provides creating tags to filter assets by category.

Procedure: Create tags and filter assets by tags

1. Open the **Overview** tab of an asset and click the **Metadata** screen.
2. In the **Tags** field, enter a name of your new tag and click **Add a new tag(s)** button. You can assign multiple tags to an asset at once by separating tag names by space.

   Figure 3.2. Creating Tags



   The assigned tags are displayed as buttons next to the Tags field:

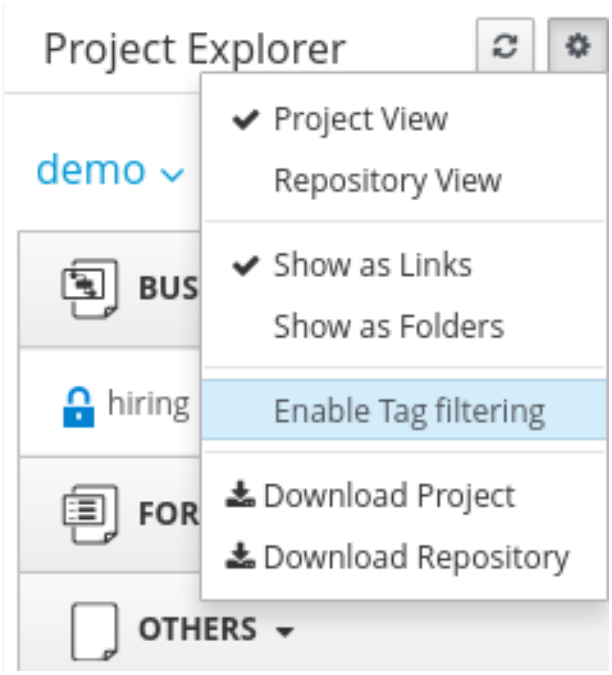
   Figure 3.3. Tags in Metadata View

   In order to delete any tag, click the respective tag button.

3. Click **Save** button to save your metadata changes.

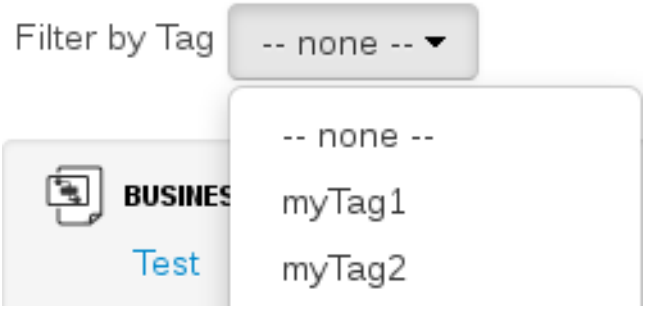
4. Once you are done assigning tags to your assets, click the (Customize View) button in the Project Explorer and select the **Enable Tag filtering** option:

Figure 3.4. Enable Tag Filtering



This displays a **Filter by Tag** drop-down list in the Project Explorer.

Figure 3.5. Filter by Tag



You can sort your assets through this filter to display all service tasks that include the selected metadata tag.

3.7. ASSET LOCKING SUPPORT

The default locking mechanism for locking a BPM and BRMS asset while updating it in Business Central is pessimistic. Whenever you open and modify an asset in Business Central, it automatically locks the asset for your exclusive use, in order to avoid conflicts in a multi-user setup. The pessimistic lock is automatically released when your session ends or when you save or close the asset.

The pessimistic lock feature is provided in order to help prevent users from overwriting each other’s changes. However, there may be cases when you may want to edit a file locked by another user. Business Central allows you to force unlock a locked asset. To do this:

Procedure: Unlocking assets

1. Open the asset.

2. Click on the **Overview** tab and open up the**Metadata** screen.
   If the asset is already being edited by another user, the following will be displayed in the **Lock status** field: **Locked by <user_name>**.

3. To edit the asset locked by another user, click **Force unclock asset** button.
   The following confirmation popup message is displayed:

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

4. Click **Yes** to confirm.
   The asset goes back to unlocked state.

3.8. PROCESS DEFINITION

A Process definition is a BPMN 2.0-compliant file that serves as container for a Process and its BPMN Diagram. A Process definition itself defines the **import** entry, imported Processes, which can be used by the Process in the Process definition, and **relationship** entries. We refer to a Process definition as a business process.

Example 3.2. BPMN2 source of a Process definition

```
<definitions id="Definition"
               targetNamespace="http://www.jboss.org/drools"
               typeLanguage="http://www.java.com/javaTypes"
               expressionLanguage="http://www.mvel.org/2.0"
               xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"Rule
Task
               xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.omg.org/spec/BPMN/20100524/MODEL
BPMN20.xsd"
               xmlns:g="http://www.jboss.org/drools/flow/gpd"
               xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
               xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
               xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
               xmlns:tns="http://www.jboss.org/drools">

    <process>
      PROCESS
    </process>

    <bpmndi:BPMNDiagram>
     BPMN DIAGRAM DEFINITION
    </bpmndi:BPMNDiagram>

    </definitions>
```

3.8.1. 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).

CREATING PACKAGES

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:

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

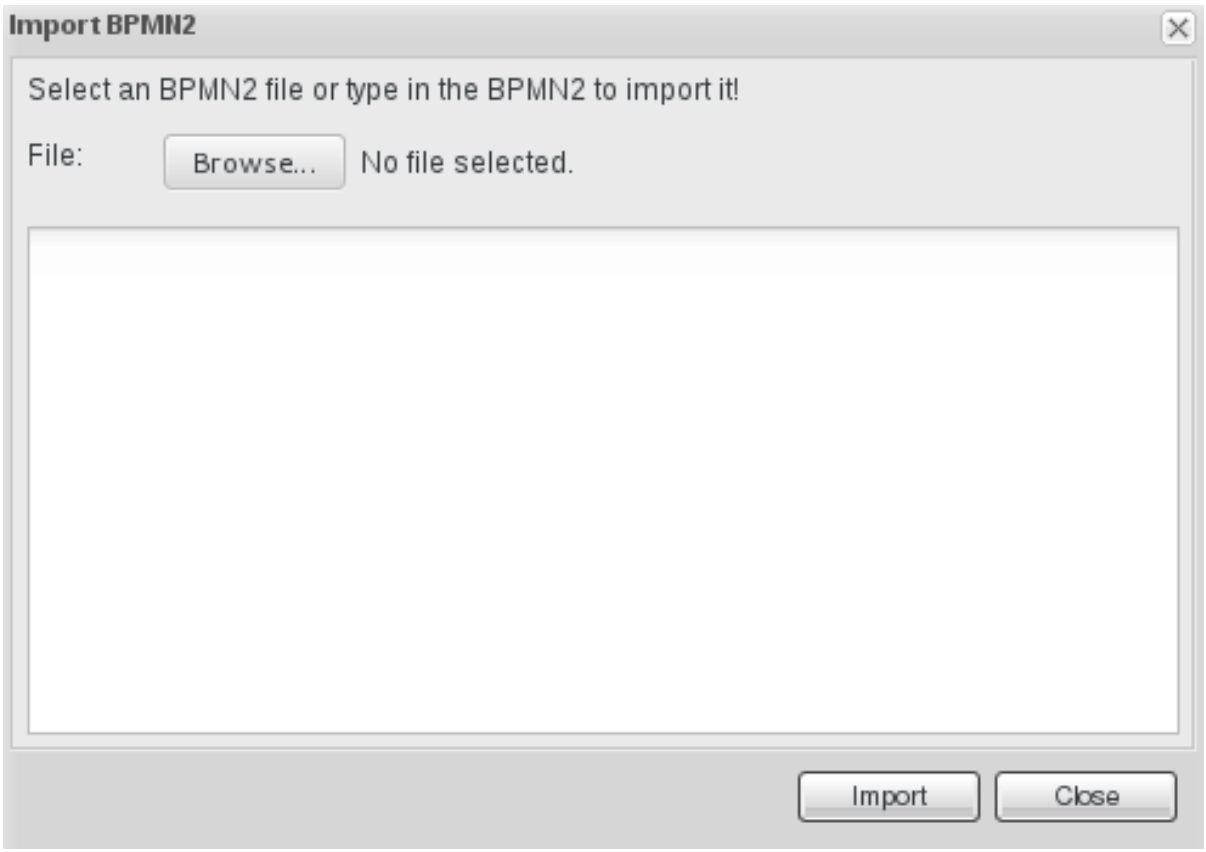
3. From the perspective menu, go to **New Item → Business Process**.
4. In the **New Processes** dialog box, enter the Process name and click**OK**. Wait until the Process Editor with the Process diagram appears.

3.8.2. 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 3.6. 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.

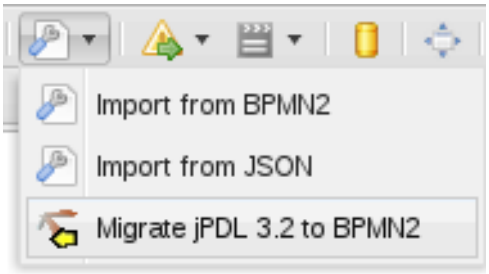
A process can also be imported to the git repository in the filesystem by cloning the repository, adding the process files, and pushing the changes back to git. In addition to alternative import methods, you can copy and paste a process or just open a file in the import dialog.

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.

3.8.3. Importing jPDL 3.2 to BPMN2

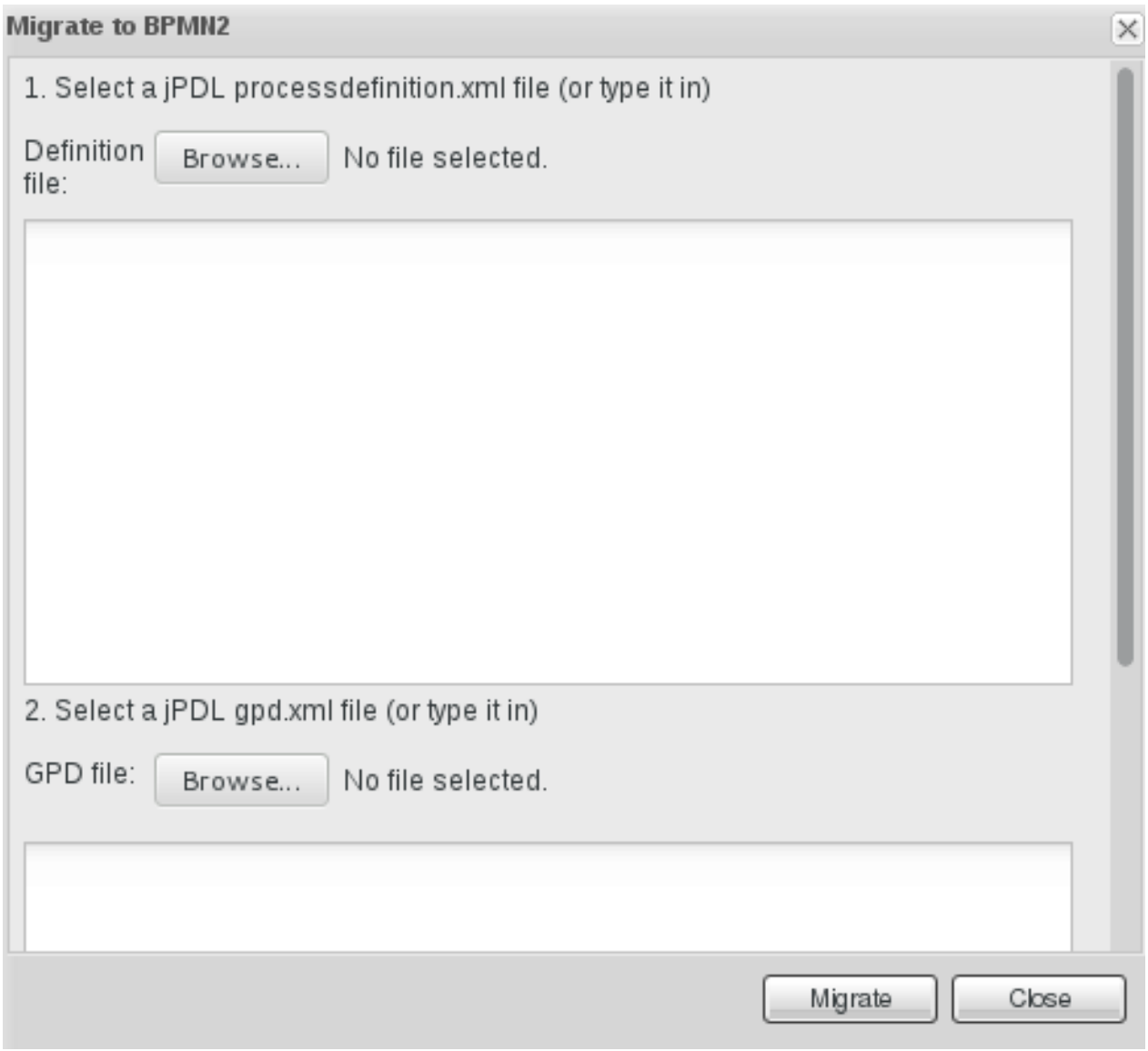
To migrate and import a jPDL definition to BPMN2, in the Process Designer, click on the import button then scroll down and select **Migrate jPDL 3.2 to BPMN2**.

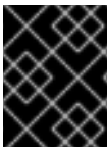
Figure 3.7. Migrate jPDL 3.2 to BPMN2



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 3.8. 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.

CHAPTER 4. PROCESS DESIGNER

The *Process Designer* is the Red Hat JBoss BPM Suite process modeler. The output of the modeler is a BPMN 2.0 process definition file, which is saved in the Knowledge Repository, under normal circumstances with a package of a project. The definition then serves as input for JBoss BPM Suite Process Engine, which creates a Process instance based on the definition.

The editor is delivered in two variants:

JBoss Developer Studio Process Designer

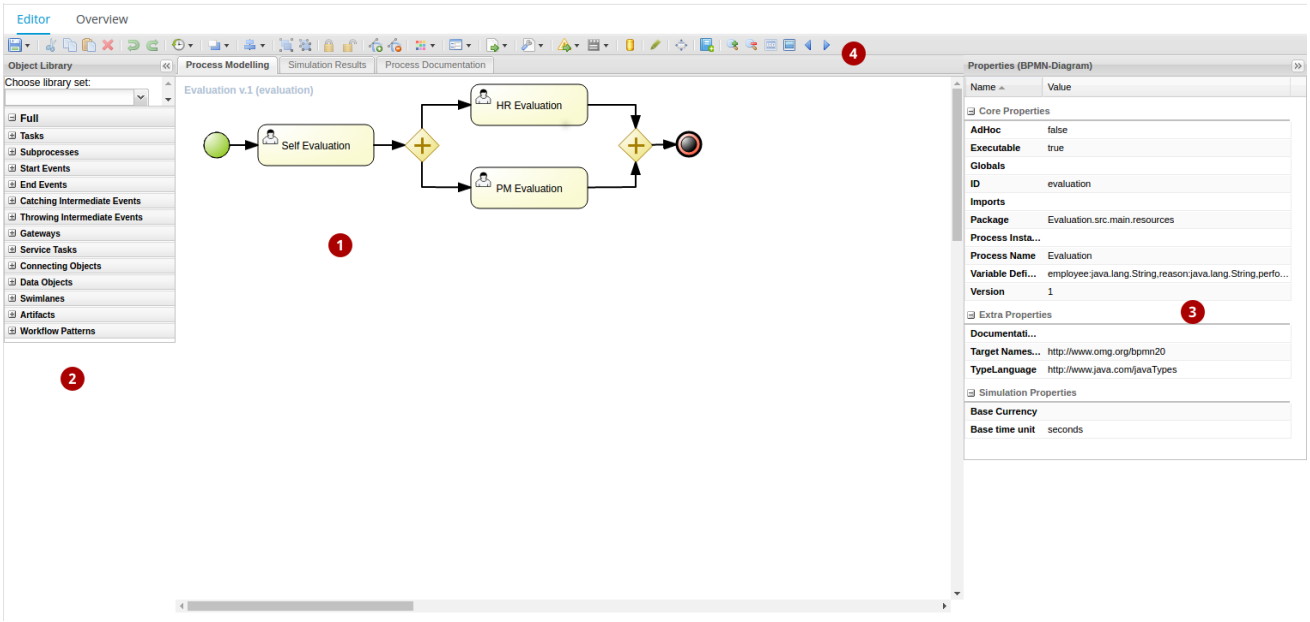
Thick-client version of the Process Designer integrated in the JBoss Developer Studio plug-in

Web Process Designer

Thin-client version of the Process Designer integrated in BPM Central

The graphical user interface of the Process Designer is the same for both the JBoss Developer Studio Process Designer and the Web Process Designer.

Figure 4.1. Process Designer environment

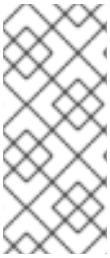


The canvas represents the process diagram. Here you can place the elements from the palette which will constitute the Process. Note that one Process definition may contain exactly one process diagram; therefore a Process definition equals to a Process diagram (this may differ in other products).

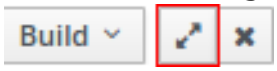
The Object Library (palette) contains groups of BPMN2 elements. Details on execution semantics and properties of individual BPMN2 shapes are available in Appendix A, *Process Elements*.

The Properties panel displays the properties of the selected element. If no element is selected, the panel contains Process properties.

The editor toolbar allows you to select an operation to be applied to the Elements on the canvas.



NOTE

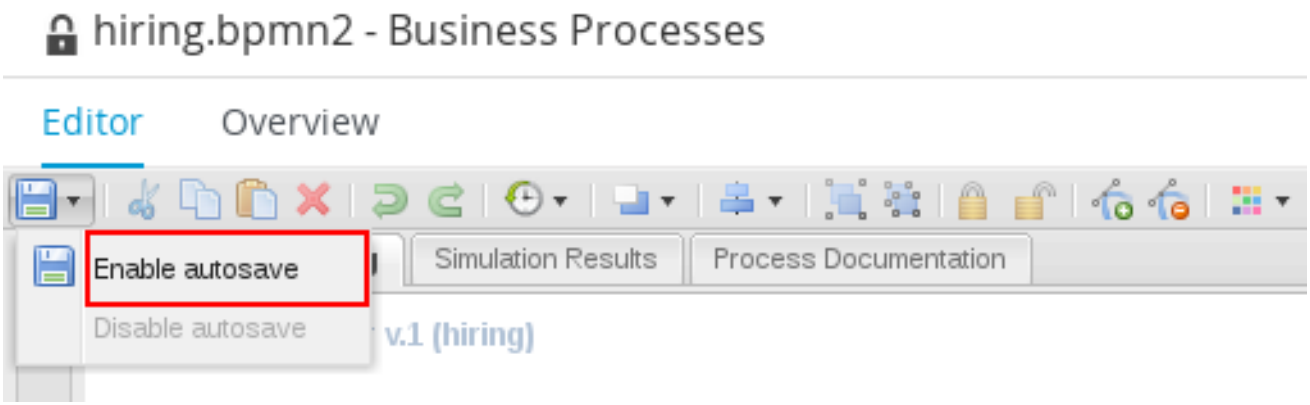
To enlarge the Process Designer screen (or any screen while working in Business Central), click on the button shown here: 

. This will make your current editor fill the entire Business Central screen. To go back, simply click the button again.

4.1. CONFIGURING AUTOMATIC SAVING

The automatic saving feature periodically commits every change in Process Designer into a Git repository. To set an automatic saving, click the  button in Process Designer and choose **Enable autosave**.

Figure 4.2. Enable Autosave Option in Process Designer



4.2. DEFINING PROCESS PROPERTIES

To define Process properties, do the following:

1. Open your process in the Process Designer.
2. Click anywhere on the canvas. Make sure that no process element is selected.

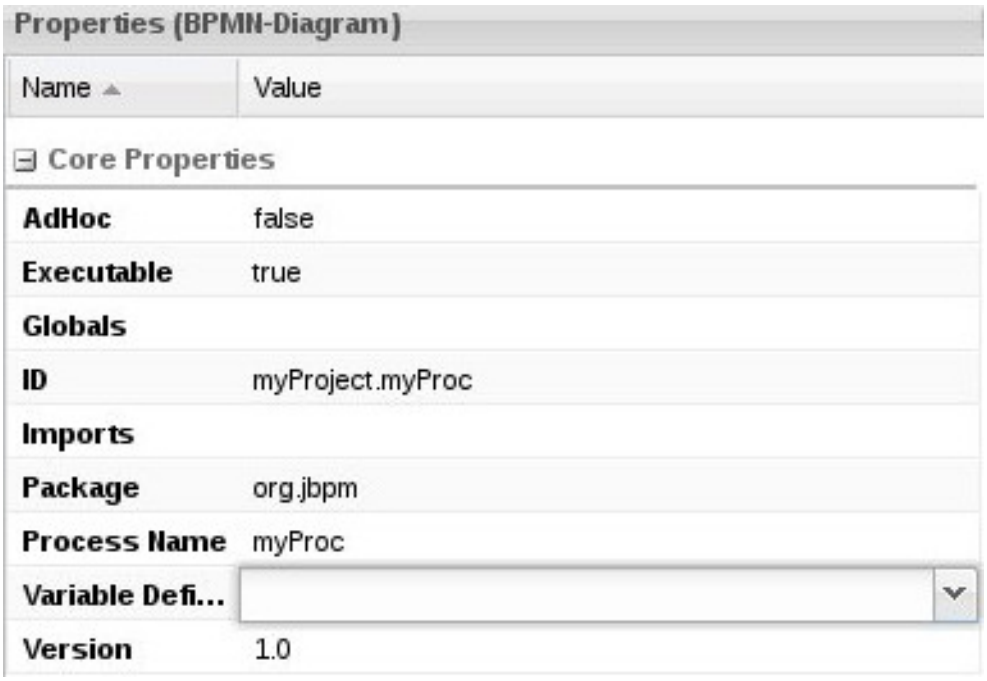
PROCESS PROPERTIES RESTRICTIONS

When creating a new process or copying an existing process with a name that uses a multibyte encoding (for example in Japanese, Chinese, Russian, or other), these characters are converted to their URL equivalent when the editor generates the process ID property.

Due to BPMN2 type restrictions, it is *not* recommended to use multibyte encodings when manually changing the process ID.

3. Click  to expand the **Properties (BPMN-Diagram)** panel.

Figure 4.3. Opening Variable Editor



4. Define the process properties on the tab by clicking individual entries. For entries that require other input that just string input, the respective editors can be used by clicking the arrow icon. Note that editors for complex fields mostly provide validation and auto-completion features.

5. To save your changes, click **Save** in the top right corner.

4.3. DESIGNING PROCESS

To model a process, do the following:

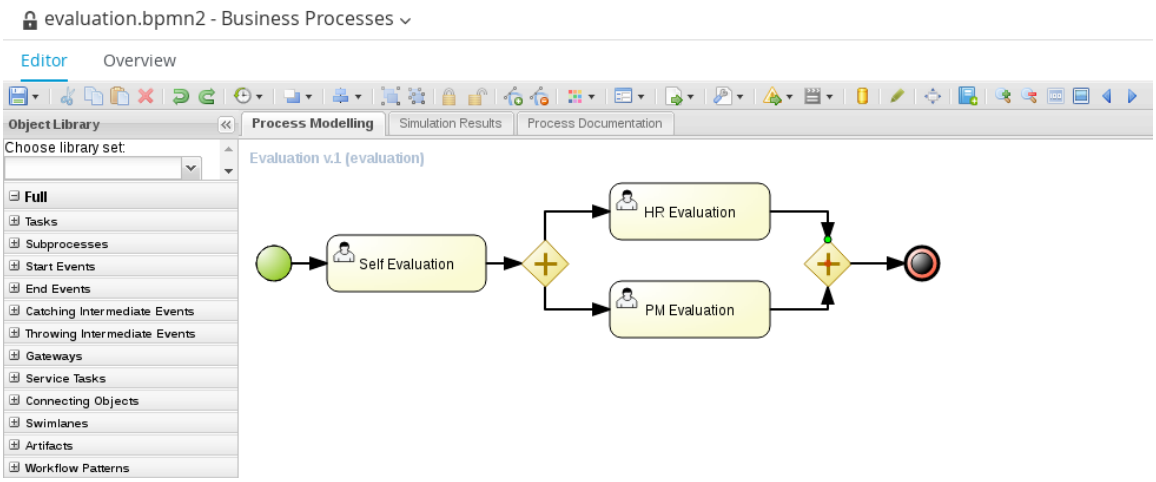
1. In Business Central, go to **Authoring → Project Authoring**. Locate your project in the **Project Explorer** and choose the respective process under**Business Processes**.
   Alternatively, you can locate the process definition in the **Repository View** of the **Project Explorer**. To show the**Repository View**, click the  button.

   The Process Designer opens.

2. Add and edit the required shapes to the process diagram on the canvas.

   a. Drag and drop the shapes from the **Object Library** palette to the required position on the canvas.

Figure 4.4. Object Library in the Process Designer



b. The quick linker menu appears after you select a shape already placed on the canvas. The quick linker feature displays only the elements that can be connected to the selected shape and connects them with a valid association element.

NOTE

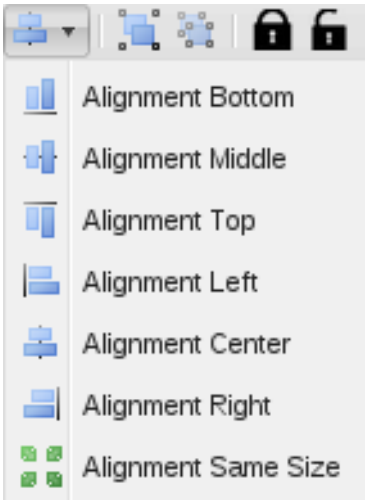
It is possible to change the type of an already placed element. To do so, select the element and click the **Morph shape** () icon from the quick linker menu.

3. Double-click an element to provide or change its name. Also, consider defining the element properties in the **Properties** view on the right side of the Process Designer.

4. Repeat the previous steps until the process diagram defines the required workflow.

4.3.1. Aligning Elements

To align diagram Elements, select the elements and click the respective button in the alignment toolbar:



- **Bottom**: the selected elements will be aligned with the element located at the lowest position

- **Middle**: the selected elements will be aligned to the middle relative to the highest and lowest element
- **Top**: the selected elements will be aligned with the element located at the highest position
- **Left**: the selected elements will be aligned with the leftmost element
- **Center**: the selected elements will be aligned to the center relative to the leftmost and rightmost element
- **Right**: the selected elements will be aligned with the rightmost element

Note that dockers of Connection elements are not influenced by aligning and you might need to remove them.

4.3.2. Changing Element Layering

To change the element layering, select the required element or a group of elements and click the  button in the Process Designer toolbar. Choose one of the following options:

-  **Bring To Front**: bring the selected element to the foreground of the uppermost layer.
-  **Bring To Back**: send the selected element to the background of the lowest layer.
-  **Bring Forward**: bring the selected element to the foreground by one layer.
-  **Bring Backward**: send the selected element to the background by one layer.

Note that the connection elements are not influenced by the layering and remain always visible.

4.3.3. Bending Connection Elements

When moving an Element with incoming or outgoing Connection elements, dockers are automatically added to accommodate the appropriate Connection shape. To create a docker manually, click and pull the respective point of the Connection object. To delete a docker, click the **Delete a Docker**  button in the toolbar and then click the respective Docker. Once you delete dockers of a Connection object, no more dockers will be created automatically.

4.3.4. Resizing Elements

To resize Elements on the canvas, select the element, and click and pull the blue arrow displayed in the upper left or lower right corner of the element.

To make the size of multiple elements identical, select the Elements and then click the  icon in the toolbar and then click on **Alignment Same Size**: all Elements will be resized to the size of the largest selected Element.

Note that only Activity Elements can be resized.

4.3.5. Grouping Elements

To create and manage an element group:

1. Select the elements on the canvas.
2. Click **Groups all selected shapes** () to group the elements.
3. Click **Deletes the group of all selected shapes** () to ungroup the elements.

4.3.6. Locking Elements

When you lock process model elements, the elements cannot be edited or moved.

- To lock the elements, select the elements and click **Lock Elements** ().
- To unlock the elements, select the elements and click **Unlock Elements** ().

4.3.7. Changing Color Scheme

Color schemes define the colors used for individual process elements in a diagram.

Color schemes are stored in the **`themes.json`** file, which is located in the**`global`** directory of each repository.

Procedure: Adding New Color Scheme

1. Locate your project in the **Project Explorer** and switch to the**Repository View** by clicking the  button.
2. Open the **`global`** directory.
3. Locate and open the **`themes.json`** file.
4. Click **Download**.
   The file is downloaded to your computer. You can now open the file in a text editor and update it locally. Note that it is not possible to update the file directly in Business Central.
5. Upload the updated file. Click **Choose file...** (), select the **`themes.json`** file and click **Upload** ().
   In order to be able to use the new color schemes, you have to reload the browser.

To apply a new color scheme or any other defined scheme, click the  button in the Process Designer toolbar and select one of the available color schemes from the drop-down menu.

4.3.8. Recording local history

Local history keeps track of any changes, you apply to your Process model so as to allow you to restore any previous status of the Process model. By default, this feature is turned off.

To turn on local history recording, click the **Local History** button and select **Enable Local History** entry. From this menu, you can also display the local history records and apply the respective status to the Process as well as disable the feature or clear the current local history log.

4.3.9. Enlarging and shrinking canvas

To change the size of the canvas, click the respective yellow arrow on the canvas edge.

4.3.10. Validating a Process

Process validation can be set up to be continuous or to be only immediate.

To validate your Process model continuously, click the **Validate** ( ) button in the toolbar of the Process Designer with the Process and click **Start Validating**. If validation errors have been detected, the elements with errors are highlighted in orange. Click on the invalid element on the canvas to display a dialog with the summary of its validation errors.

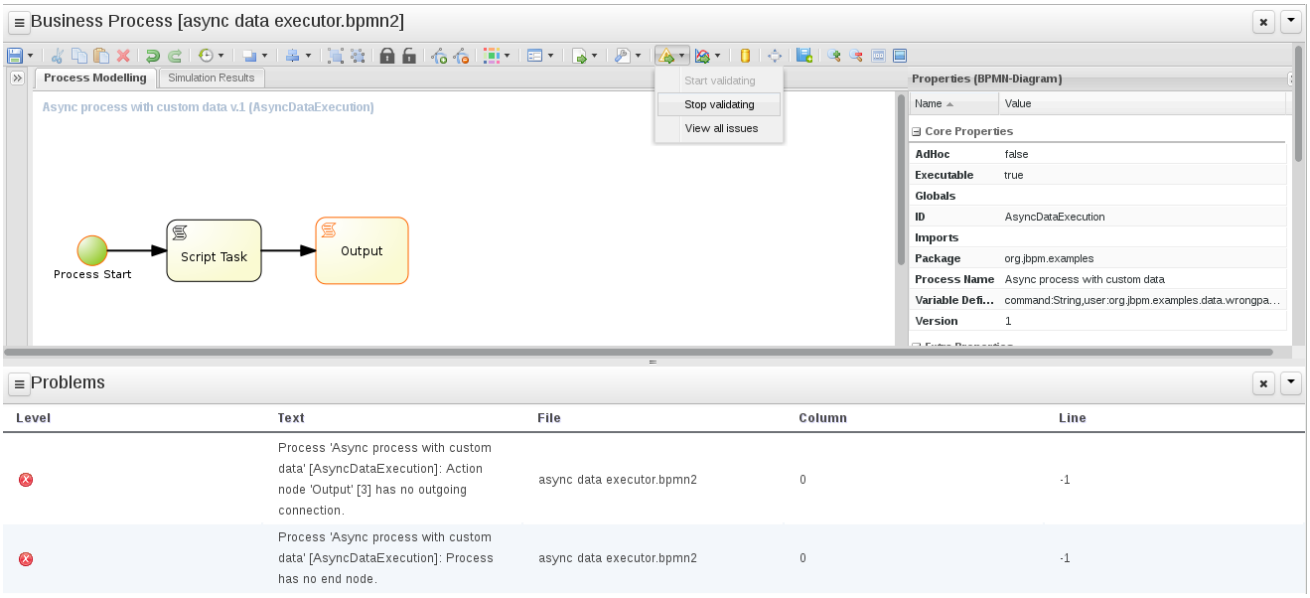
To disable continuous validation, click the **Validate** ( ) button in the toolbar of the Process Designer with the Process and click **Stop Validating**.

Also note that errors on the element properties are visualized in further details in the Properties view of the respective element.

If you want to display the validation errors and not to keep the validation feature activated, click the **Validate** ( ) button in the toolbar of the Process Designer with the Process and click **View all issues**.

Additionally after you save your Process, any validation errors are also displayed in the **Messages** view.

Figure 4.5. Stopping continuous validation

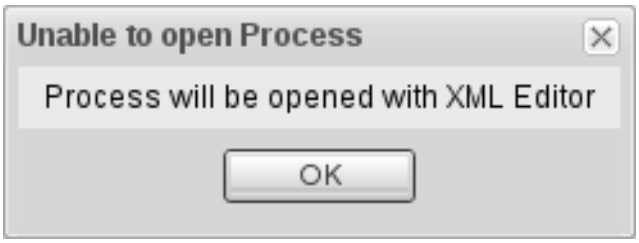


4.3.11. Correcting Invalid Processes

If your process is invalid and the Process Designer is unable to render it in the designer canvas, you can open the process in XML format and make the necessary corrections.

1. In the Project view of the Project Explorer, select your Project and open the process.
   If the process is valid, the Process Designer opens process diagram on the canvas.

   If the process is invalid, you will see the following prompt:

2. Click **OK**.
   The invalid process opens as XML in a text editor in the Process Designer.

3. You can restore previous correct version of the process by selecting the version either from the **Latest Version** drop-down menu or from the**Overview** tab.
   Alternatively, you can edit the XML to correct the business process and click **Save**.

   You can now open the valid process and view it as a diagram on the canvas.

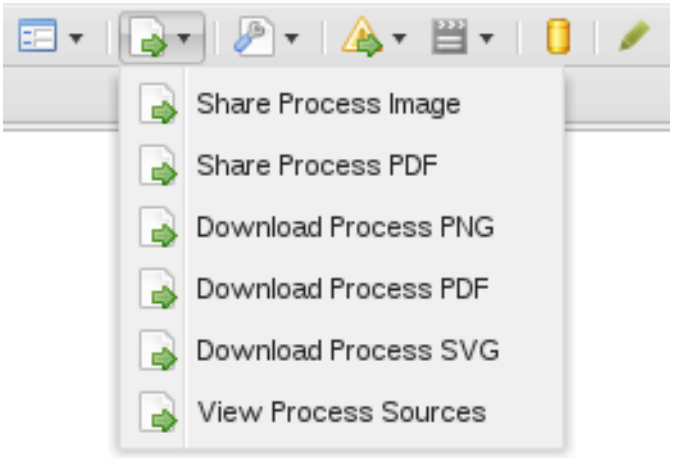
4.4. EXPORTING PROCESS

To export your process definition into one of the supported formats (PNG, PDF, BPMN2, JSON, SVG, or ERDF), do the following:

1. In Business Central, go to **Authoring** → **Project Authoring**.

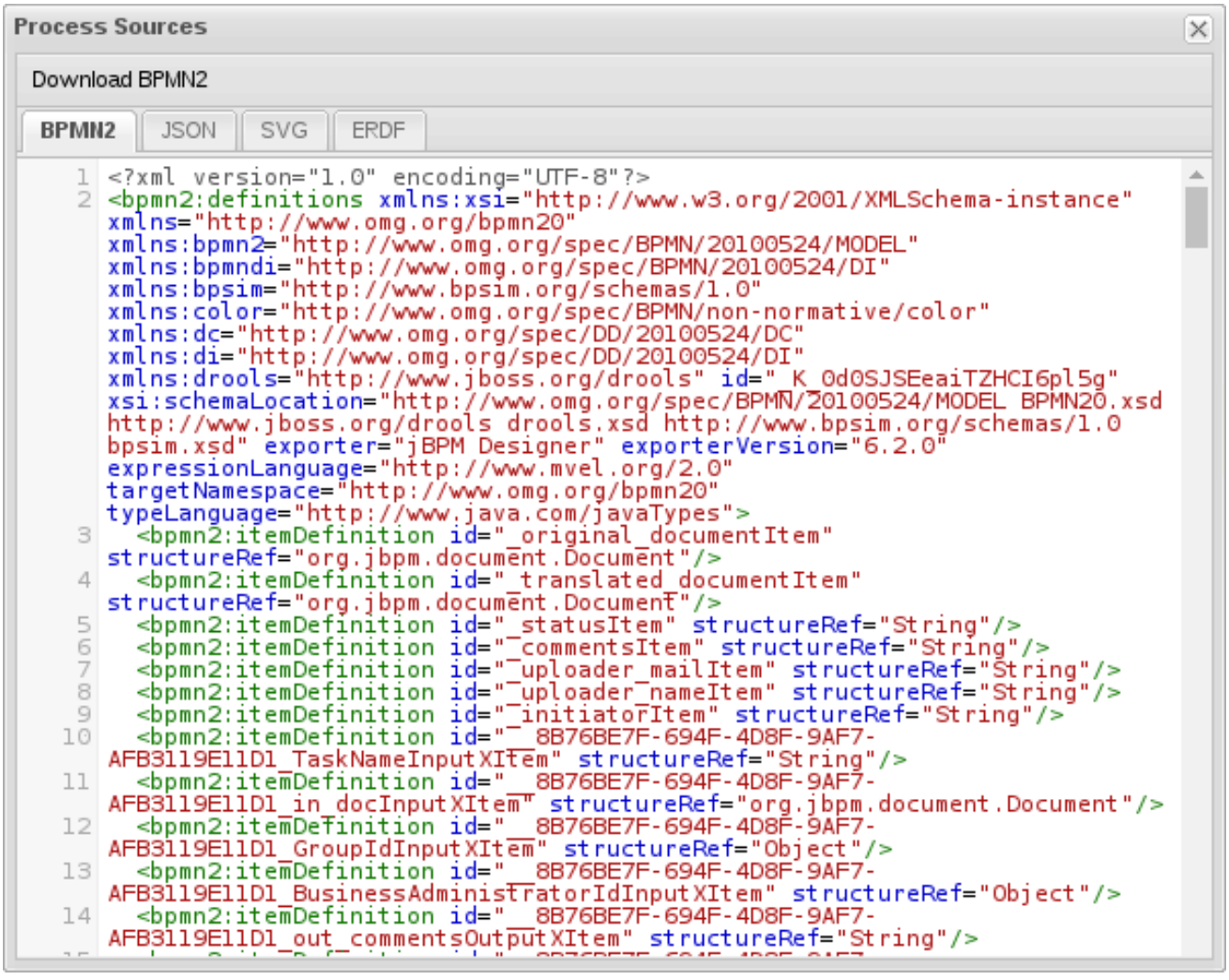
2. Open your process in **Process Designer**.

3. Click the  button and choose one of the following options:



- **Share Process Image**: generates a PNG file into the repository and provides the ability to insert it in an HTML page using generated HTML tag.

- **Share Process PDF**: generates a PDF file into the repository and provides the ability to insert it in an HTML page using generated HTML tag.
  Note that Internet Explorer 11 does not support PDF objects in HTML.
- **Download Process PNG**: generates a PNG file into the repository and the browser starts downloading the file.
- **Download Process PDF**: generates a PDF file into the repository and the browser starts downloading the file.
- **Download Process SVG**: generates an SVG file into the repository and the browser starts downloading the file.
- **View Process Sources**: opens the **Process Sources** dialog box that contains the BPMN2, JSON, SVG, and ERDF source codes. You can download BPMN2 files by clicking **Download BPMN2** at the top. Pressing CTRL+A allows you to select the source code in a particular format, while pressing CTRL+F enables the find tool (use **/re/*SYNTAX*** for a regexp search).



4.5. PROCESS ELEMENTS

4.5.1. Generic Properties of Visualized Process Elements

All Process elements have the following visualization properties, which can be defined in their **Properties** tab:

Background

The background color of the element in the diagram

Border color

The border color of the element in the diagram

Font color

The color of the font in the element name

Font size

The size of the font in the element name

Name

The element name displayed on the BPMN diagram

4.5.2. Defining Process Element Properties

All process elements, including the process, contain a set of properties that define the following:

- *Core* properties, which include properties such as the name, data set, scripts, and others.
- *Extra* properties, which include the properties necessary for element execution (see Section A.6, “Process Elements”), data mapping (variable mapping) and local variable definitions (see Section 4.9.1, “Global Variables”), and properties that represent an extension of the jBPM engine, such as **`onExitAction`**, documentation, and similar.
- *Graphical* properties, which include graphical representation of elements (such as colors, or text settings).
- *Simulation* properties, which are used by the simulation engine.

In element properties of the String type, use **`#{expression}`** to embed a value. The value will be retrieved on element instantiation, and the substitution expression will be replaced with the result of calling the **`toString()`** method on the variable defined in the expression.

Note that the expression can be the name of a variable, in which case it resolves to the value of the variable, but more advanced MVEL expressions are possible as well, for example **`#{person.name.firstname}`**.

To define element properties, do the following:

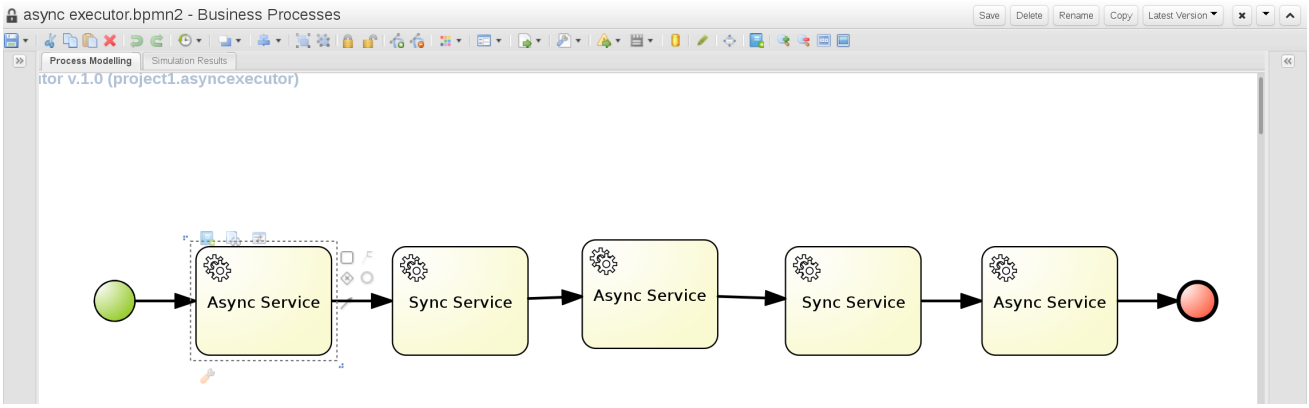
1. Open the process definition in the Process Designer.
2. On the canvas, select an element.
3. Click  in the upper right corner of the Process Designer to display the **Properties** view.
4. In the displayed Properties view, click the property value fields to edit them. Note that where applicable, you can click the drop-down arrow and the relevant value editor appears in a new dialog box.
5. Click **Save** in the upper right corner and fill out the**Save this item** dialogue to save your changes.

4.6. CREATING BUSINESS PROCESS SAVE POINT

To ensure the engine will save the state of the process, a save point is created when a node is executed asynchronously. Asynchronous continuation allows process designers to decide what activities should be executed asynchronously without any additional work.

Using the Process Design tool, you can decide which nodes should be executed in the background. The following example shows two types of tasks:

- Synchronous (Sync Service)
- Asynchronous (Async Service)

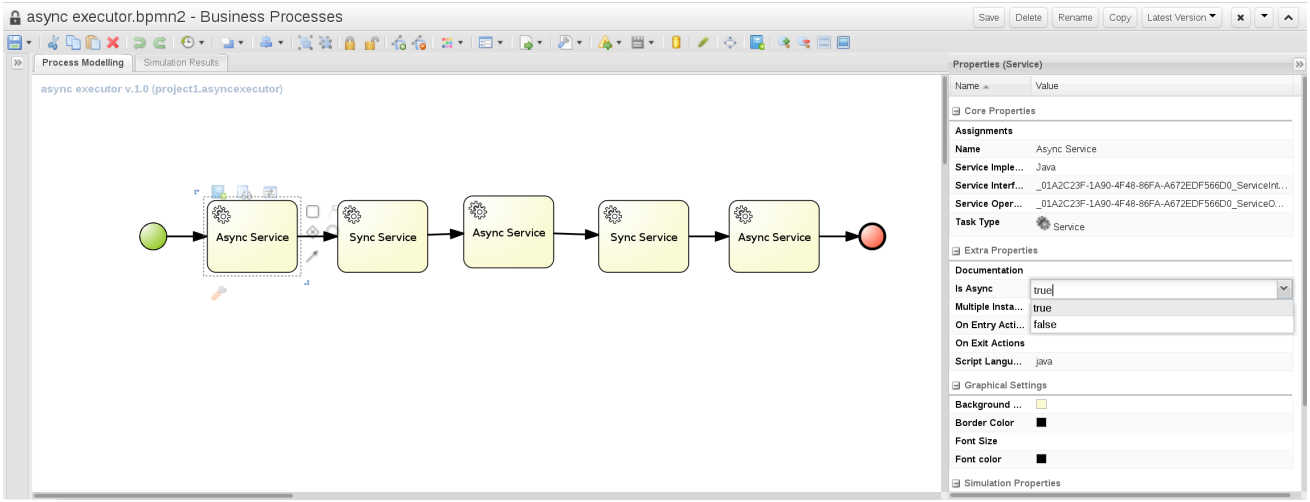


In the provided example, the Async Service will be executed in background while Sync Service will be executed on the same thread as its preceding node - so if the preceding node is asynchronous, the synchronous node will directly follow it within same thread.

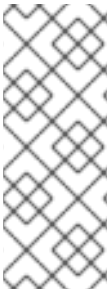
Use the following procedure to make a service task asynchronous.

Procedure: Define a Service Task as Asynchronous

1. Open the **Properties** menu on the right side of the Business Process screen.
2. Select Service Task you want to make asynchronous in the Process Modelling window.
3. Under the **Extra Properties** menu, set the **Is Async** option to **true**.



This feature is available for all task types (service, send, receive, business rule, script, user task), subprocesses (embedded and reusable), and multi-instance task and subprocesses.



NOTE

This feature relies on the ExecutorService, which is the backbone of asynchronous processing in JBoss BPM Suite. The ExecutorService must be configured and running in order to use this feature. This is already fully equipped in Business Central, however if using JBoss BPM Suite in embedded mode, additional steps will be required depending on how you utilize the JBoss BPM Suite API.

4.6.1. Enabling Asynchronous Execution in Embedded Mode

To enable save points by allowing nodes to execute asynchronously in embedded mode, the following steps will be required depending on your API.

KIE API (KieBase and KieSession)
When using the KIE API, configure ExecutorService and add it to kieSession environment under "ExecutorService" key. Once this has been added, it will process the nodes asynchronously.

RuntimeManager API
Configure ExecutorService and add it as one of environment entries when setting up RuntimeEnvironment.

jBPM Services API
Add ExecutorService as an attribute of DeploymentService. If you use CDI or EJB it will be injected automatically (assuming all dependencies are available to the container).

4.7. FORMS

A *form* is a layout definition for a page (defined as HTML) that is displayed as a dialog window to the user on a

- **`process instantiation`** or a
- **`task instantiation`**.

The form is then respectively referred to as a *Process form* or a *Task form*. It serves for acquiring data for the Element instance execution, be it a Process or Task, from a human user: a Process form can take as its input and output Process variables; a Task form can take as its input DataInputSet variables with assignment defined, and as its output DataOutputSet variables with assignment defined.

For example, you could ask the user to provide the input parameters needed for Process instantiation and display any variable data connected to the Process; or using a Human Task show information and request input for further Process execution.

This data can be mapped to the Task as DataInputSet and used as the Task's local variables and to DataOutputSet to provide the data to the parent Process instance (see Section 4.12, "Assignment").

4.7.1. Defining Process form

A Process form is a form that is displayed at Process instantiation to the user who instantiated the Process.

To create a Process form, do the following:

1. Open your Process definition in the Process Designer.
2. In the editor toolbar, click the **Form**( ) icon and then **Edit Process Form**.
3. Select the editor to use to edit the form. Note that this document deals only with the **Graphical Modeler** option.

Note that the Form is created in the root of your current Project and is available from any other Process definitions in the Projects.

4.7.2. Defining Task form

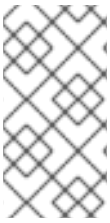
A Task form is a form that is displayed at User Task instantiation, that is, when the execution flow reaches the Task, to the Actor of the User Task.

To create a Task form, do the following:

1. Open your Process definition with the User Task in the Process Designer.
2. Select the Task on the canvas and click the **Edit Process Form** ( ) in the User Task menu.
3. In the displayed Form Editor, define the Task form.

4.7.3. Defining form fields

Once you have created a form definition, you need to define its content: that is its fields and the data they are bound to. You can add either the pre-defined field types to your form, or define your own data origin and use the custom field types in your form definition.



NOTE

Automatic form generation is not recursive, which means that when custom data objects are used, only the top-level form is generated (no subforms). The user is responsible for creating forms that represent the custom data objects and link them to the parent form.

4.8. FORM MODELER

Red Hat JBoss BPM Suite provides a custom editor for defining forms called Form Modeler.

Form Modeler includes the following key features:

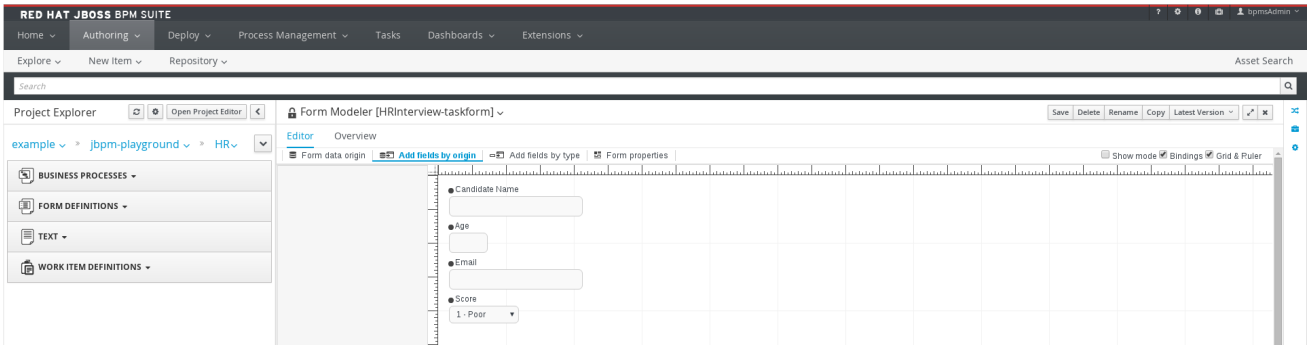
- Form Modeling WYSIWYG UI for forms
- Form autogeneration from data model / Java objects
- Data binding for Java objects

- Formula and expressions
- Customized forms layouts
- Forms embedding

Form Modeler comes with predefined field types, such as **`Short Text`**, **`Long Text`**, or **`Integer`**, which you place onto the canvas to create a form. In addition to that, Form Modeler also allows you to create custom types based on data modeler classes, Java classes (must be on the classpath), or primitive Java data types. For this purpose, the **Form data origin** tab contains three options: **From Data Model**, **From Java Class**, and **From basic type**.

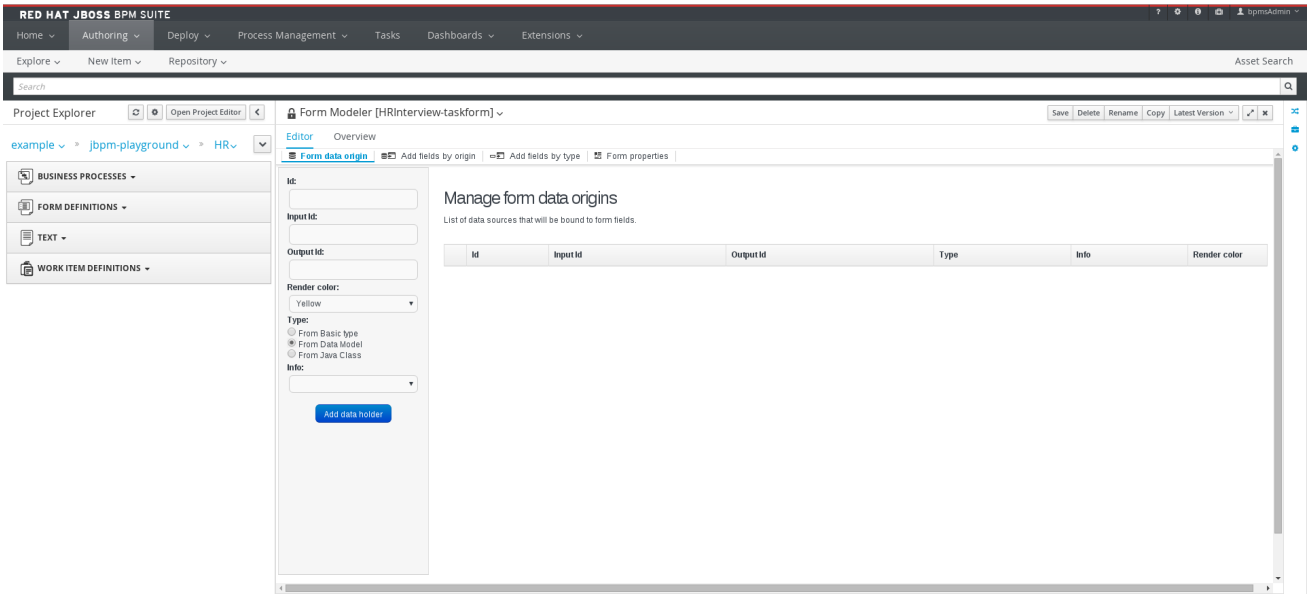
Use the **Add fields by origin** tab visible in the following figure to select fields based on their source.

Figure 4.6. Adding fields by origin



To view and add Java classes created in Data Modeler in Form Modeler, go to section **Form data origin** and select the **From Data Model** option shown in the following figure.

Figure 4.7. Adding classes from data model



You can adjust the form layout using the **Form Properties** tab that contains a **Predefined** layout selected by default, as well as a **Custom** option.

When a task or process calls a form, it sends the form a map of objects, which include local variables of the process or task. Also, when the form is completed, a map is sent back to the process or task with the data acquired in the form. The form assigns this output data to

the local variables of the task or process, and the output data can therefore be further processed.

4.8.1. Creating a Form in Form Modeler

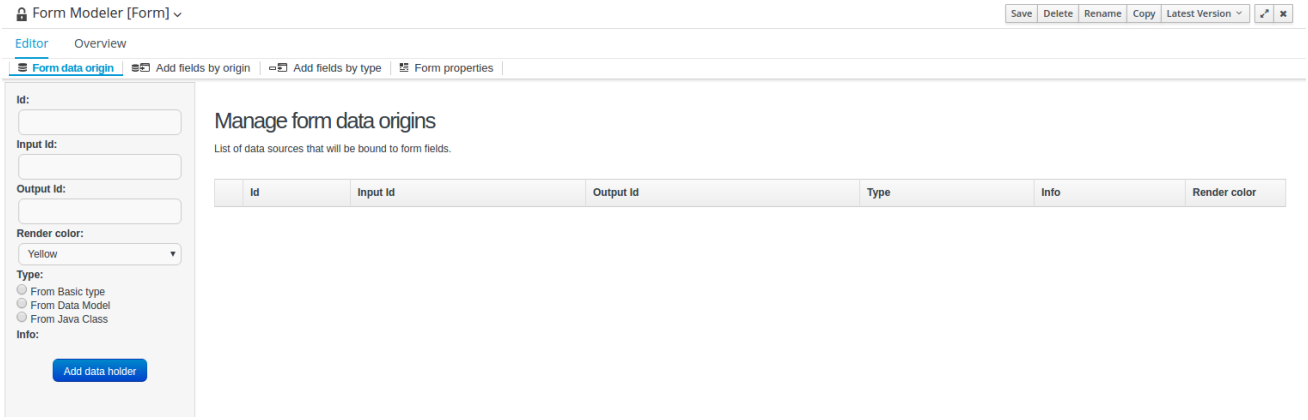
To create a new form in Form Modeler, do the following:

1. In Business Central, go to **Authoring** → **Project Authoring**.
2. On the perspective menu, select **New Item** → **Form**.
3. In the **Create New Form** dialog that opens, fill out the name of your form in **Resource Name** and click **OK**.

The newly created form will open up. You can add various fields to it when you select the **Add fields by type** option on the Form Modeler tab. Use the button to place the field types onto the canvas, where you can modify them.To modify the field types, use the icons that display when you place the cursor over a field: **First**, **Move field**, **Last**, **Group with previous**, **Edit**, or **Clear**. The icons enable you to change the order of the fields in the form, group the fields, or clear and edit their content.

The following figure shows a new form created in Form Modeler.

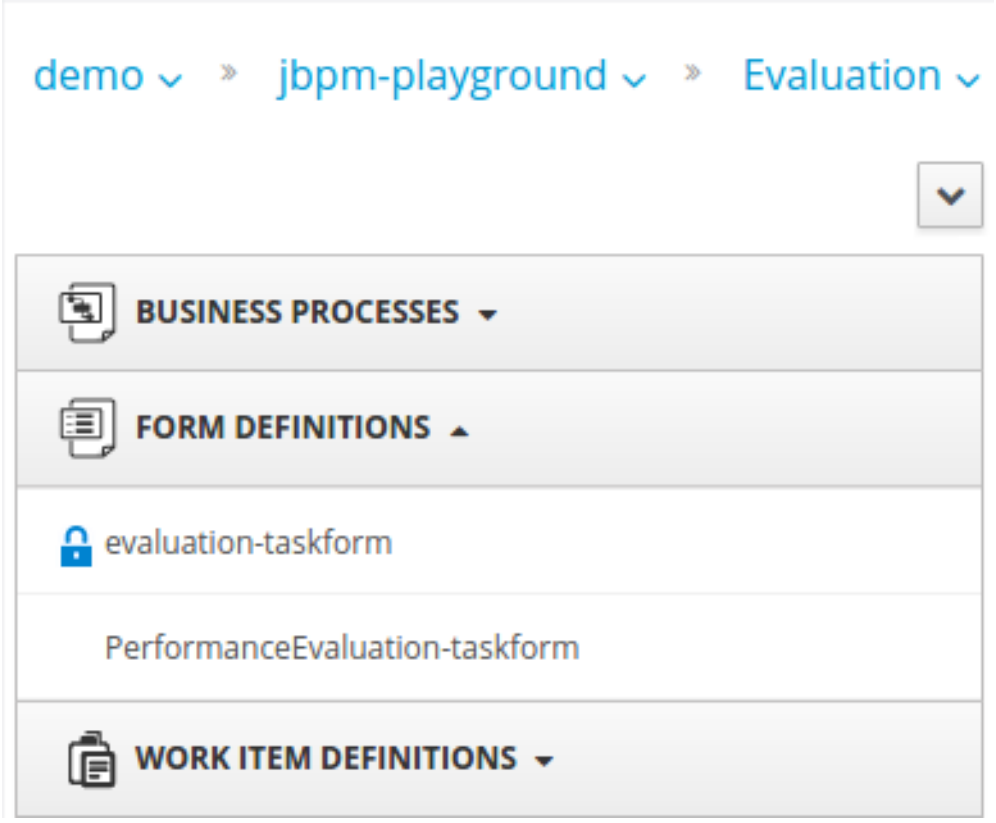
Figure 4.8. New form



4.8.2. Opening an Existing Form in Form Modeler

To open an existing form in a project that already has a form defined, go to **Form Definitions** in Project Explorer and select the form you want to work with from the displayed list.

Figure 4.9. Opening an Existing Form



4.8.3. Setting Properties of a Form Field in Form Modeler

To set the properties of a form field, do the following:

1. In Form Modeler, select the **Add fields by type** tab and click the arrow button to the right of a field type. The field type is added to the canvas.

2. On the canvas, place the cursor on the field and click the edit icon.

3. In the **Properties** dialog that opens on the right, set the form field properties and click **Apply** at the bottom of the dialog for HTML Labels. For other form field properties, the properties change once you have removed focus from the property that you are modifying.

4.8.4. Configuring a Process in Form Modeler

You can generate forms automatically from process variables and task definitions and later modify the forms using the form editor. In runtime, forms receive data from process variables, display it to the user, capture user input, and update the process variables with the new values. To configure a process in Form Modeler, do the following:

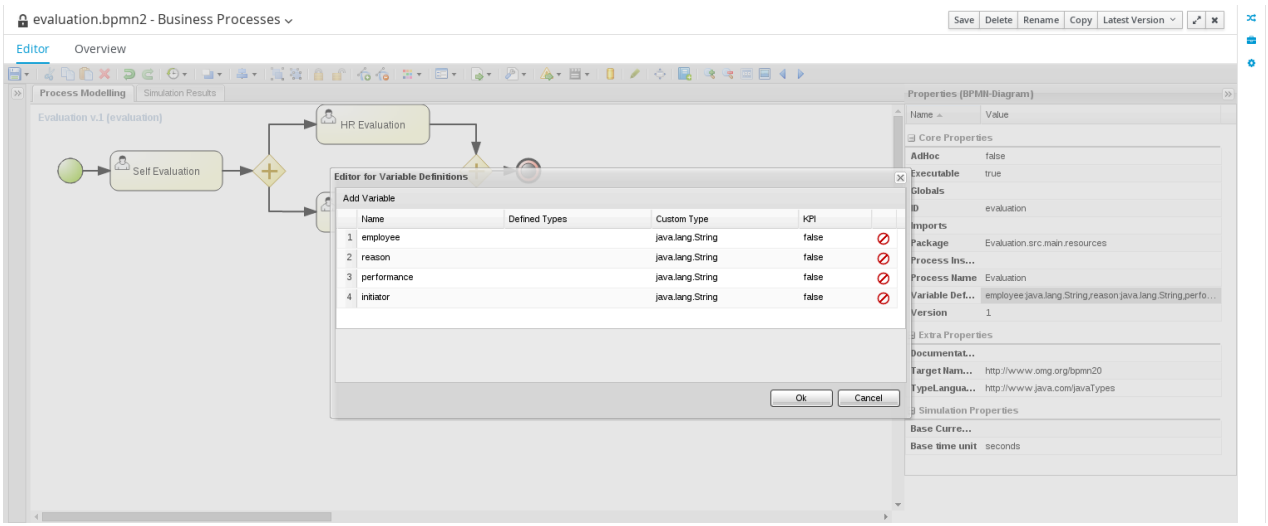
1. Create process variables to hold values entered into forms. Variables can be simple (e.g. 'string') or complex. You can define complex variables using Data Modeler, or create them in any Java integrated development environment (Java IDE) as regular plain Java objects.

2. Declare the process variables in the 'variables definition' property.

3. Determine which variables you want to set as input parameters for the task, which shall receive response from the form, and establish mappings by setting the

'DataInputSet', 'DataOutputSet', and 'Assignments' properties for any human task. To do so, use the Editor for Data Input, Editor for Data Output, and Editor for Data Assignment.

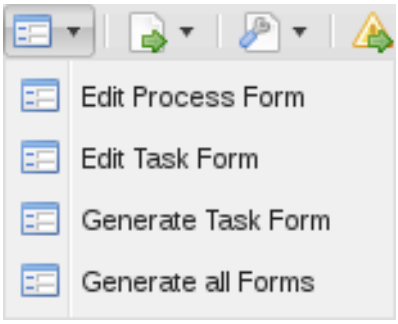
Example 4.1. Defining a Variable using Data Modeler



4.8.5. Generating Forms from Task Definitions

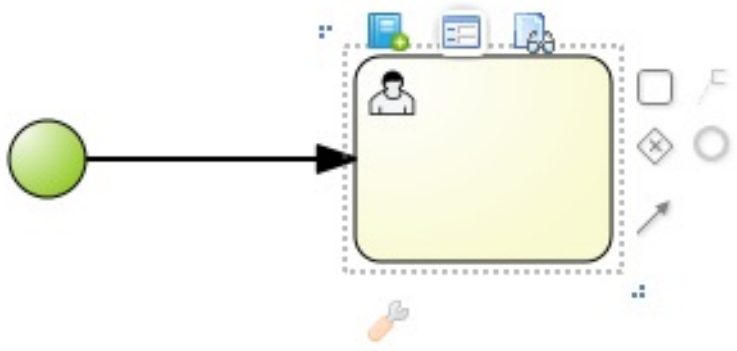
In the Process Designer module, you can generate forms automatically from task and variable definitions, and easily open concrete forms from Form Modeler by using the following menu option:

Figure 4.10. Generating Forms Automatically



To open and edit a form directly, click the Edit Task Form icon ( ) located above a user task.

Figure 4.11. Editing the Task Form



Forms follow a naming convention that relates them to tasks. If you define a form named **`formName-taskform`** in the same package as the process, the human task engine will use the form to display and capture information entered by the user. If you create a form named **`ProcessId-task`**, the application will use it as the initial form when starting the process.

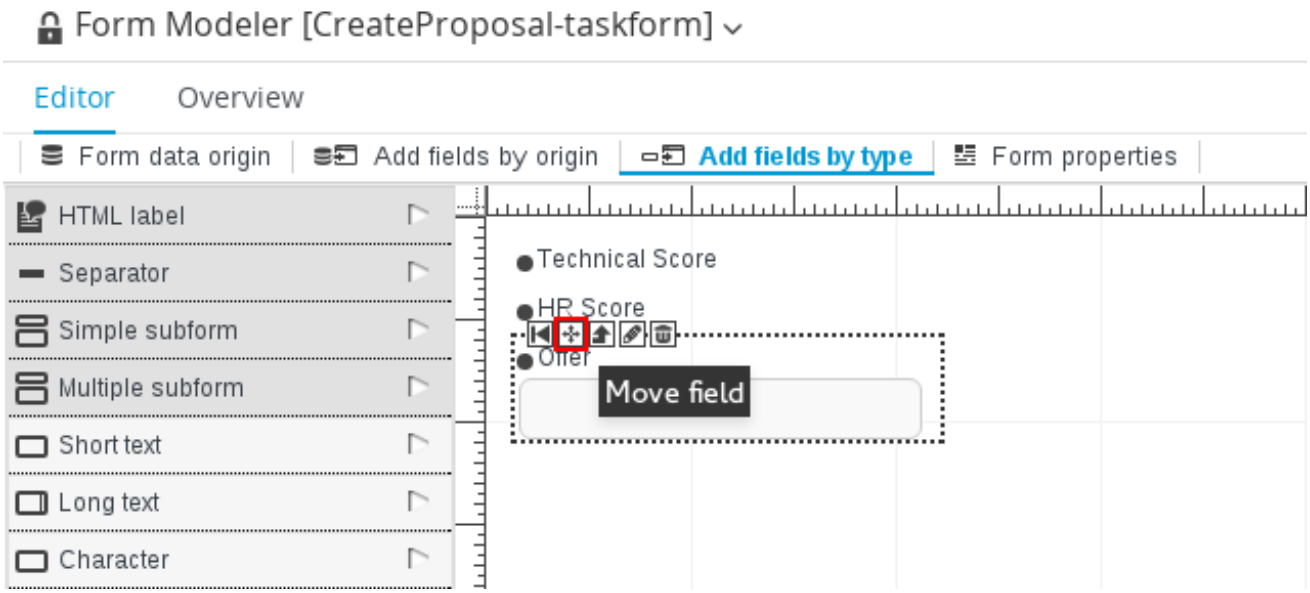
4.8.6. Editing Forms

After you generate a form, you can start editing it. If the form has been generated automatically, the **Form data origin** tab contains the process variables as the origin of the data, which allows you to bind form fields with them and create data bindings. Data bindings determine the way task input is mapped to form variables, and when the form is validated and submitted, the way values update output of the task. You can have as many data origins as required, and use different colors to differentiate them in the **Render color** drop down menu. If the form has been generated automatically, the application creates a data origin for each process variable. For each data origin bindable item, there is a field in the form, and these automatically generated fields also have defined bindings. When you display the fields in the editor, the color of the data origin is displayed over the field to give you quick information on correct binding and implied data origin. To customize a form, you can for example **`move fields`**, **`add new fields`**, **`configure fields`**, or **`set values for object properties`**.

4.8.7. Moving a Field in Form Modeler

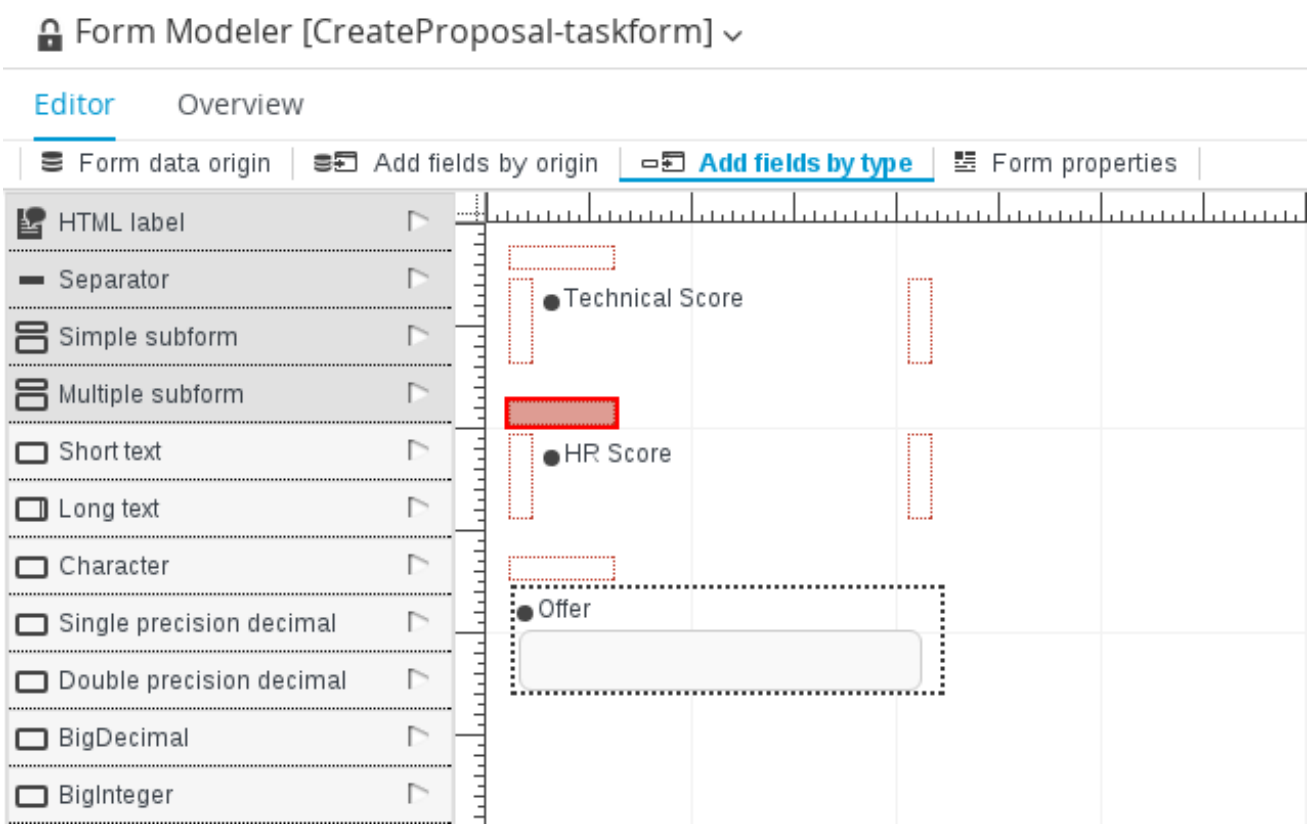
You can place fields in different areas of the form. To move a field, access the field's contextual menu and select the **Move field** option shown on the following screenshot. This option displays the different regions of the form where you can place the field.

Figure 4.12. Moving a Form Field in Form Modeler



After you click the **Move field** option, a set of rectangular contextual icons appears. To move a field, select one of them according to the desired new position of the field.

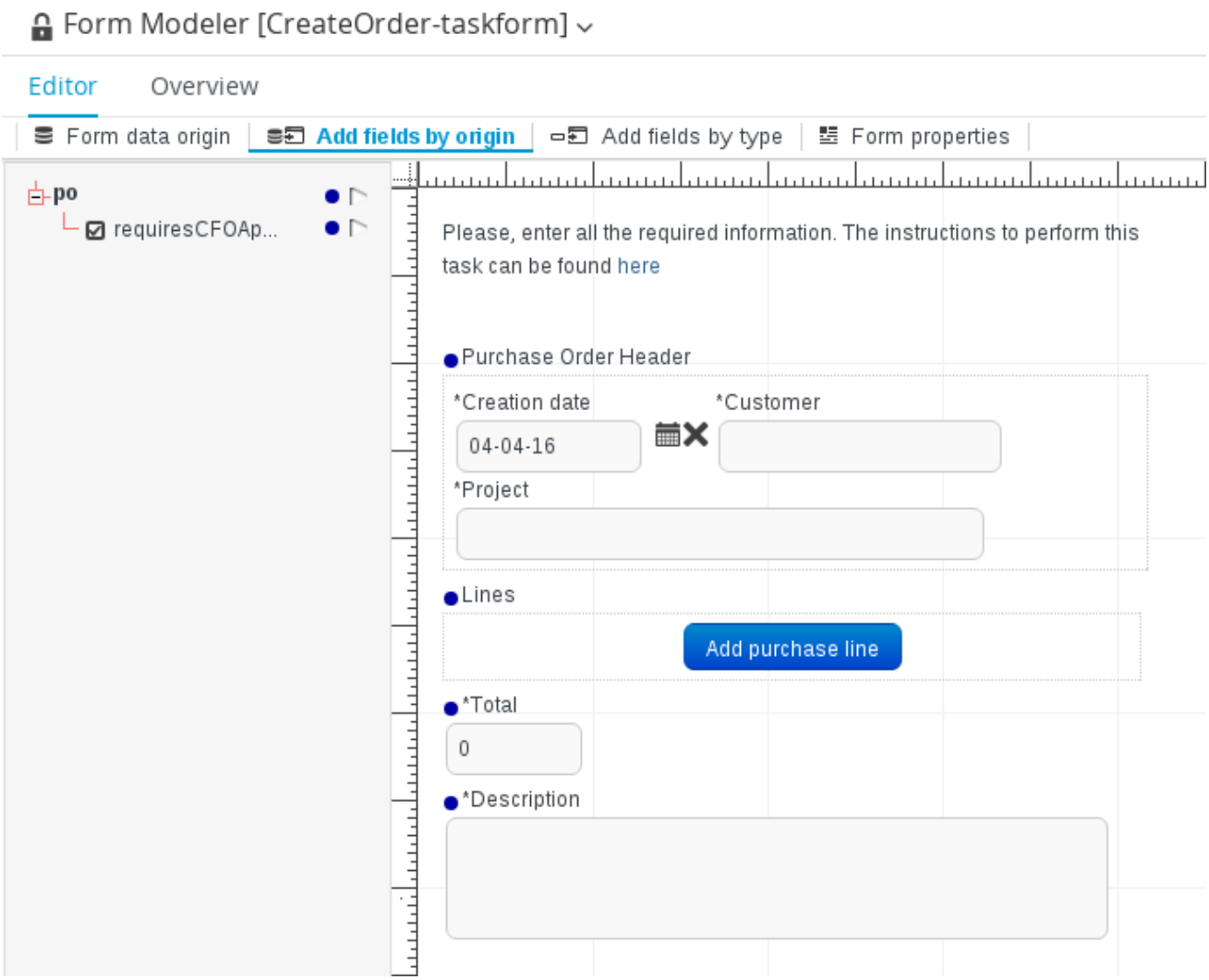
Figure 4.13. Destination Areas to Move a Field



4.8.8. Adding New Fields to a Form

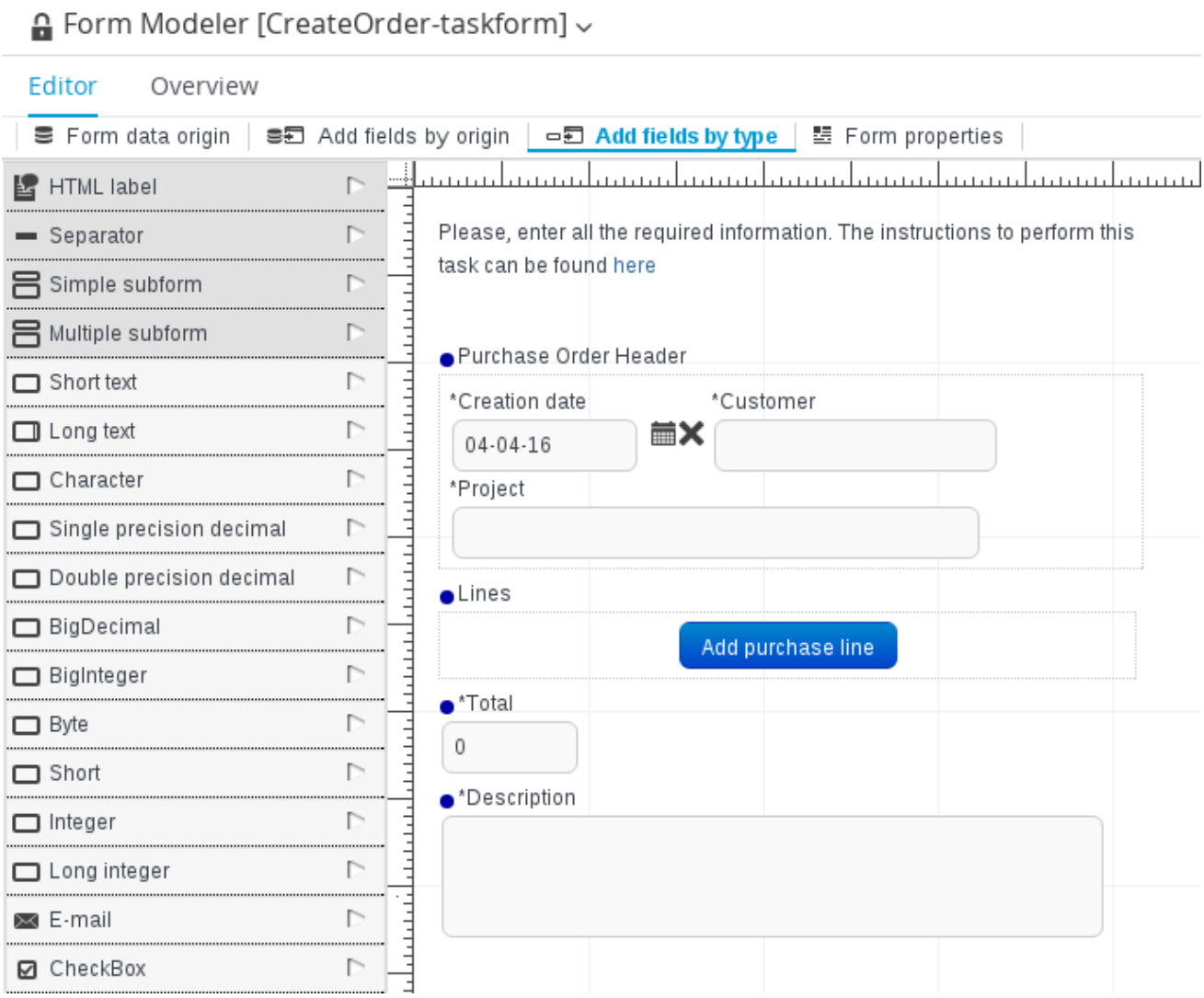
You can add fields to a form by their origin or by selecting the type of the form field. The **Add fields by origin** tab allows you to add fields to the form based on defined data origins.

Figure 4.14. Adding Fields by Origin



The fields then have correct configuration of the **`Input binding expression`** and **`Output binding expression`** properties, so when the form is submitted, the values in the fields are stored in the corresponding data origin. The **Add fields by type** tab allows you to add fields to the form from the fields type palette of the Form Modeler. The fields do not store their value for any data origin until they have correct configuration of the **`Input binding expression`** and **`Output binding expression`** properties.

Figure 4.15. Adding Fields by Type



There are three kinds of field types you can use to model your form: simple types, complex types, and decorators. The **`simple types`** are used to represent simple properties like texts, numeric values, or dates. The following table presents a complete list of supported simple field types:

Table 4.1. Simple Field Types

Name	Description	Java Type	Default on generated forms
Short Text	Simple input to enter short texts.	java.lang.String	yes
Long Text	Text area to enter long text.	java.lang.String	no
Rich Text	HTML Editor to enter formatted texts.	java.lang.Srowing	no
Email	Simple input to enter short text with email pattern.	java.lang.String	no
Float	Input to enter short decimals.	java.lang.Float	yes

Name	Description	Java Type	Default on generated forms
Decimal	Input to enter number with decimals.	java.lang.Double	yes
BigDecimal	Input to enter big decimal numbers.	java.math.BigDecimal	yes
BigInteger	Input to enter big integers.	java.math.BigInteger	yes
Short	Input to enter short integers.	java.lang.Short	yes
Integer	Input to enter integers.	java.lang.Integer	yes
Long Integer	Input to enter long integers.	java.lang.Long	yes
Checkbox	Checkbox to enter true/false values.	java.lang.Boolean	yes
Timestamp	Input to enter date and time values.	java.util.Date	yes
Short Date	Input to enter date values.	java.util.Date	no
Document	Allows the user to upload documents to the form.	org.jbpm.document.Document	No

`Complex field types` are designed for work with properties that are not basic types but Java objects. To use these field types, it is necessary to create extra forms in order to display and write values to the specified Java objects.

Table 4.2. Complex Field Types

Name	Description	Java Type	Default on generated forms
Simple subform	Renders the form; it is used to deal with 1:1 relationships.	java.lang.Object	yes
Multiple subform	This field type is used for 1:N relationships. It allows the user to create, edit, and delete a set child Objects.Text area to enter long text.	java.util.List	yes

`Decorators` are a kind of field types that does not store data in the object displayed in the form. You can use them for decorative purposes.

Table 4.3. Decorators

Name	Description
HTML label	Allows the user to create HTML code that will be rendered in the form.
Separator	Renders an HTML separator.

4.8.9. Configuring Fields of a Form

Each field can be configured to enhance performance of the form. There is a group of common properties called generic field properties and a group of specific properties that differs by field type.

Generic field properties:

- **`Field Type`** - can change the field type to other compatible field types.
- **`Field Name`** - is used as an identifier in calculating of formulas.
- **`Label`** - the text that is displayed as a field label.
- **`Error Message`** - a message displayed when there is a problem with a field, for example in validation.
- **`Label CCS Class`** - allows you to enter a class css to apply in label visualization.
- **`Label CCS Style`** - allows you to directly enter the style to be applied to the label.
- **`Help Text`** - introduced text displayed as an alternative attribute to help the user in data introduction.
- **`Style Class`** - allows you to enter a class CSS to be applied in field visualization.
- **`CSS Style`** - allows you to directly enter the style to be applied to the label.
- **`Read Only`** - a field with this property allows reading only, no write access.
- **`Input Binding Expression`** - defines the link between the field and the process task input variable. In runtime, it is used to set the field value to the task input variable data.
- **`Output Binding Expression`** - defines the link between the field and the process task output variable. In runtime, it is used to set the task output variable.

4.8.10. Creating Subforms with Simple and Complex Field Types

Complex Field types is a category of fields in a form. You can use the complex field types to model form properties that are Java Objects. Simple subform and Multiple subform are the two types of complex field types. A simple subform represents a single object and a multiple subform represents an object array inside a parent form. Once you add one of these fields into a form, you must configure the form with information on how it must display these objects during execution. For example, if your form has fields representing an object array, you can define a tabular display of these fields in the form. You cannot represent them as simple inputs such as text box, checkbox, text area, and date selector.

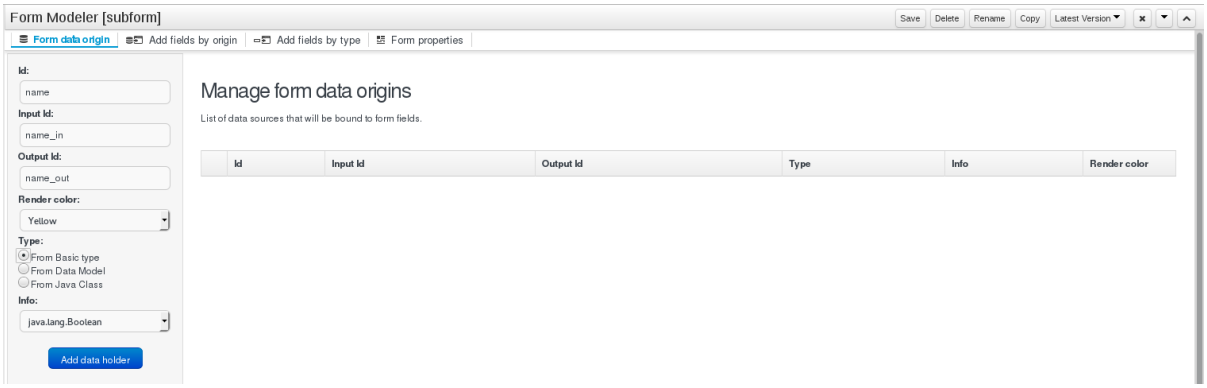
Procedure: To create and insert a subform containing a single object inside a parent form:

1. In Business Central, go to **Authoring** → **Project Authoring**.

2. On the perspective menu, select **New Item** → **Form**.
   A new form opens in the Form Modeler. You must now configure the new form with information of the object it must contain.

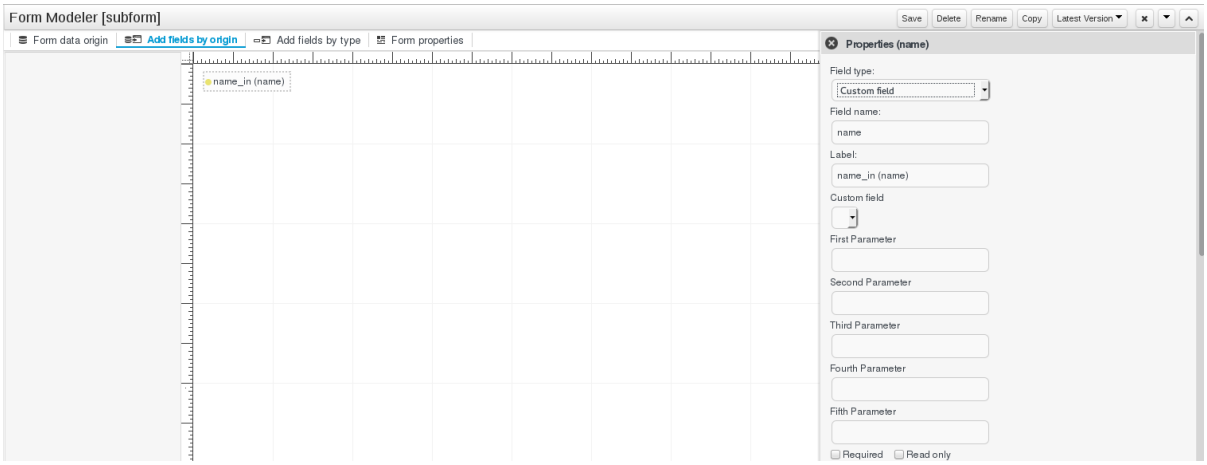
3. Enter the values for the required fields in the **Form data origin** tab and click **Add data holder**.

   Figure 4.16. Create Subform



4. Click **Add fields by origin** tab and add the listed fields to the form.

   Figure 4.17. Add fields by origin



5. Click the Edit icon on the field in the form to open the **Properties** tab.

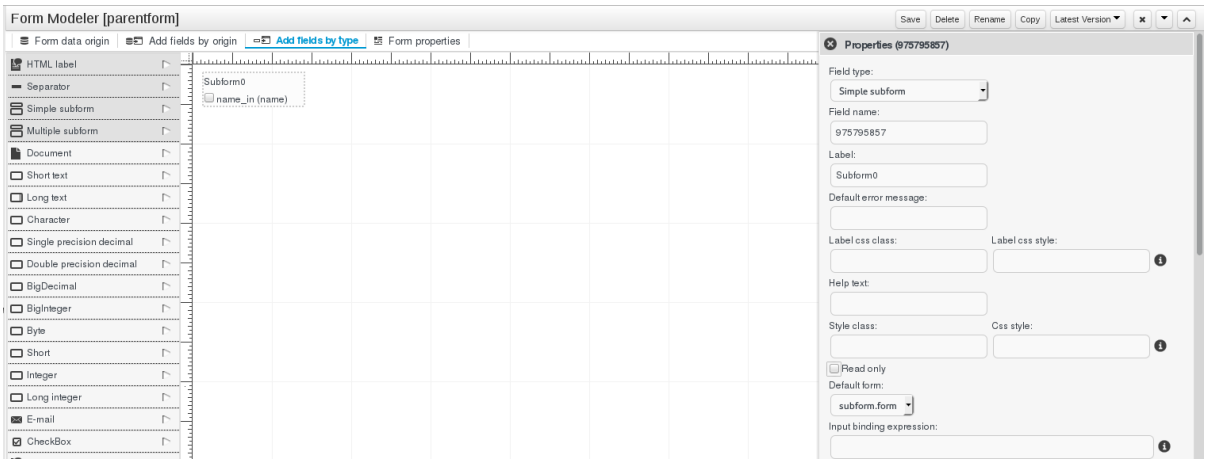
6. In the **Properties** tab, configure the form by providing required values to the fields and click **Save** to save the subform.

7. Open the parent form to configure the properties of the object.

8. In the parent form, click the **Add fields by type** tab. Select the object on the form and configure it in the **Properties** tab.

9. In the **Properties** tab, select **Simple subform** for the **Field type** property. Then select the newly created subform for the **Default form** field property.

Figure 4.18. Configure the Parent Form

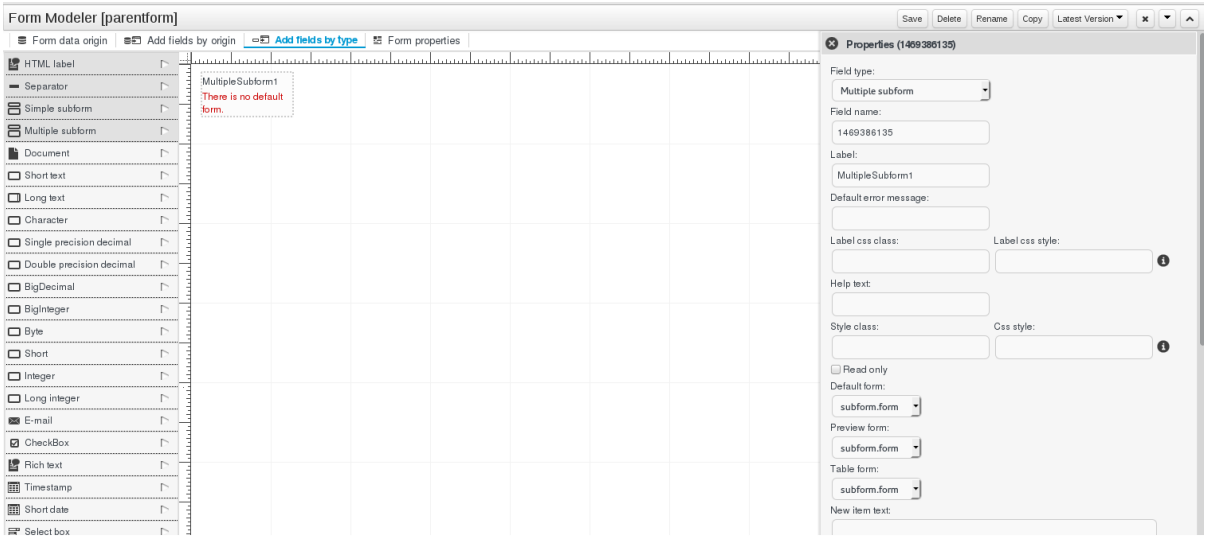


10. Click **Save** to save the parent form.
    This inserts your subform containing a single Java object inside the parent form.

Procedure: To insert a subform with multiple objects inside a parent form:

1. In Business Central, go to **Authoring** → **Project Authoring**.
2. On the perspective menu, select **New Item** → **Form**.
   A new form opens in the Form Modeler. You must now configure the new form with information on the object array it must contain.
3. Enter the values for the required fields in the **Form data origin** tab and click **Add data holder**.
4. Click **Add fields by origin** tab and add the listed fields to the form.
5. Click the Edit icon on the field in the form to open the **Properties** tab.
6. In the **Properties** tab, configure the form by providing required values to the fields. You can use the Formula Engine to automatically calculate field values.
7. Click **Save** to save the subform.
8. Open the parent form to configure the properties of each of the objects.
9. In the parent form, click the **Add fields by type** tab. Select each object on the form one by one and configure them in the **Properties** tab.
10. In the **Properties** tab, select **Multiple subform** for the **Field type** property. Then select the newly created subform for the **Default form** field property.

Figure 4.19. Configure the Parent Form



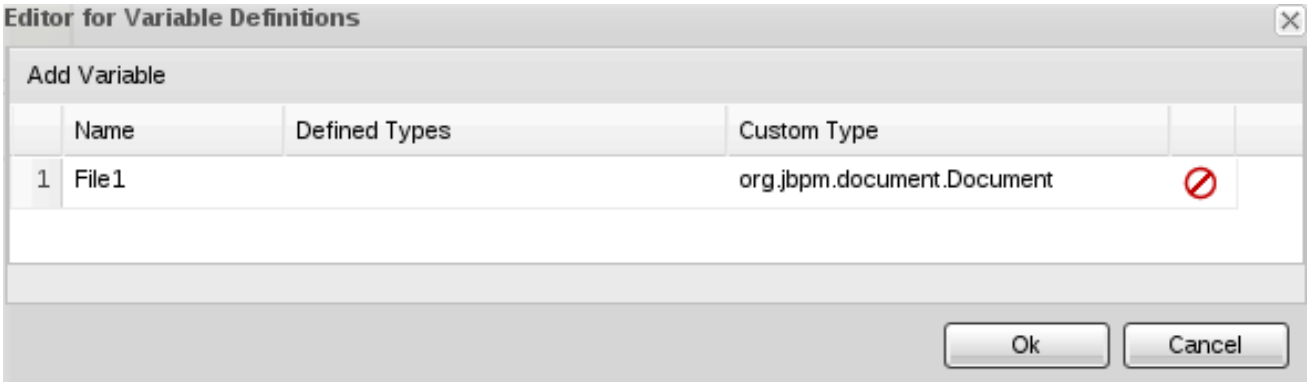
11. Click **Save** to save the parent form.
    This inserts your subform containing an array of Java objects inside the parent form.

4.8.11. Attaching Documents to a Form

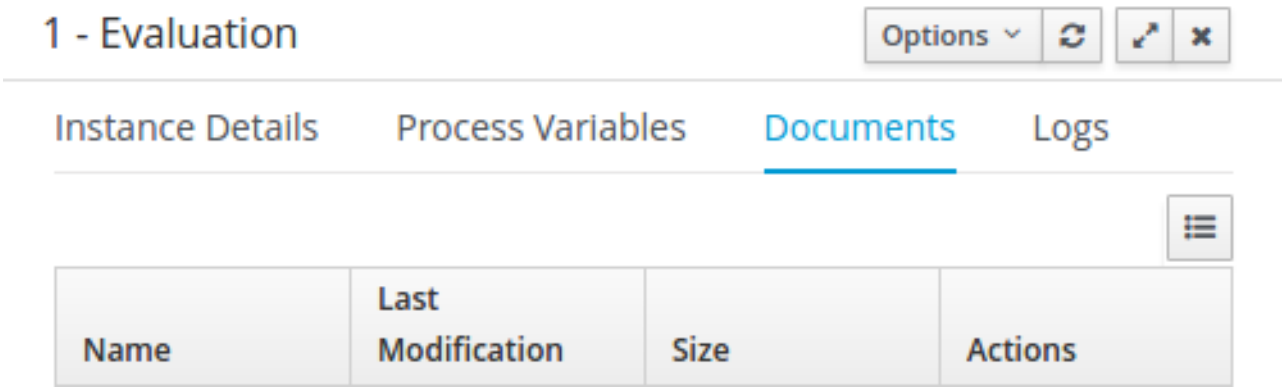
The 6.1 release of Red Hat JBoss BPM Suite has introduced the concept of attaching documents to a form by using a **`Document`** form field. This field can be attached to any form, process or task based.

To attach a **`Document`** field to a form, click on it while creating or editing an existing form (in the **Add fields by type** section).

Alternatively, you can let the system automatically generate the **`Document`** form fields based on the presence of a **`org.jbpm.document.Document`** variable type in your process.



When the end user uploads a document using this form, the uploaded document is available in a list in the task list (**task details**) and process instance details (**Document** tab). The uploaded documents are shown as links and the users can click to view them.



The Process Engine generates an instance of **`org.jbpm.document.Document`** to be stored in a process variable, which you can pass through your persistence strategy to decide where and how to store that document.

Pluggable Variable Persistence
New with this release is also the ability for you to store your document in a location of your choice. This is defined as *Pluggable Variable Persistence* and this allows you to store these documents automatically in a centralized content management system (CMS) of choice, behind the scenes.

To implement your custom persistence strategy, start by defining your *Marshaling Strategy*. This strategy is declared to the Process Engine by the use of deployment descriptors (see *Red Hat JBoss BPM Suite Administration and Configuration Guide*) using the **`<marshalling-strategy>`** element. This element should name a type that provides an implementation of the **`org.kie.api.marshalling`** interface.

The following methods in this interface help you create your strategy.

- **`public boolean accept(Object object)`**: Determines if the given object can be marshalled by the strategy.
- **`byte[] marshal(Context context, ObjectOutputStream os, Object object)`**: Marshals the given object and returns the marshalled object as byte[].
- **`Object unmarshal(Context context, ObjectInputStream is, byte[] object, ClassLoader classloader)`**: Reads the object received as byte[] and returns the unmarshalled object
- **`void write(ObjectOutputStream os, Object object)`**: same as **`marshal`** method, provided for backwards compatibility.
- **`Object read(ObjectInputStream os)`**: same as **`unmarshal`**, provided for backwards compatibility.

For example, if you create a custom strategy that stores your uploaded documents in Google Drive, your implementation class should not only implement the methods of the **`org.kie.api.marshalling`** package, but this implementation should also be made available to the Process Engine, by putting the classes in its classpath (and declaring the type in the deployment descriptors).

There is a default marshalling strategy that simply saves the uploaded documents in the file system under a folder called **`docs`**. This default implementation is defined by the **`DocumentStorageService`** class and is implemented through the **`DocumentStorageServiceImpl`** class.

4.8.12. Rendering Forms for External Use

Forms generated by the Form Builder can be reused in other client applications with the help of the REST API and a JavaScript library. The REST API defines the end points for the external client applications to call and the JavaScript library makes it easy to interact with these endpoints and to render these forms.

To use this API you will need to integrate the Forms REST JavaScript library in your client application. The details of the library and the methods that it provides are given in the following section, along with a simple example. Details of the REST API are present in the *Red Hat JBoss BPM Suite Developers Guide*, although you should probably only use the REST API via the JavaScript library described here.

4.8.12.1. JavaScript Library for Form Reuse

The JavaScript API for Form Reuse makes it easy to use forms created in one Business Central application to be used in remote applications and allows loading of these forms from different Business Central instances, submitting them, launching processes or task instances, and executing callback functions when the actions are completed.

Blueprint for using the JavaScript Library
A simple example of using this API would involve the following steps:

1. Integrate the JavaScript library in the codebase for the external client application so that its functions are available.

2. Create a new instance of the **`jBPMFormsAPI`** class in your own JavaScript code. This is the starting point for all interactions with this library.

   ```
   var jbpmRestAPI = new jBPMFormsAPI();
   ```

3. Call your desired methods on this instance. For example, if you want to show a form, you would use the following method:

   ```
   jbpmRestAPI.showStartProcessForm(hostUrl, deploymentId, processId,
   divId, onsuccess, onerror);
   ```

   and provide the relevant details (hostUrl, deploymentId, processId and so on. A full list of the methods and parameters follows after this section).

4. Do post processing with the optional **`onsuccess`** and **`onerror`** methods.

5. Work with the form, starting processes (**`startProcess()`**), claiming tasks (**`claimTask()`**) starting tasks (**`startTask()`**) or completing tasks (**`completeTask`**). Full list of available methods follows after this section.

6. Once you're finished with the form, clear the container that displayed it using **`clearContainer()`** method.

Full list of available methods in the JavaScript Library
The JavaScript library is pretty comprehensive and provides several methods to render and process forms.

1. **`showStartProcessForm(hostUrl, deploymentId, processId, divId, onsuccessCallback, onerrorCallback):`** Makes a call to the REST endpoint to obtain the form URL. If it receives a valid response, it embeds the process start form

in the stated div. You need these parameters:

- **hostURL**: The URL of the Business Central instance that holds the deployments.
- **deploymentId**: The deployment identifier that contains the process to run.
- **processId**: The identifier of the process to run.
- **divId**: The identifier of the div that has to contain the form.
- **onsuccessCallback** (optional): A JavaScript function executed if the form is going to be rendered. This function will receive the server response as a parameter.
- **onerrorCallback** (optional): A JavaScript function executed if any error occurs and it is impossible to render the form. This function will receive the server response as a parameter.

2. **startProcess(divId, onsuccessCallback, onerrorCallback):** Submits the form loaded on the stated div and starts the process. You need these parameters:
   - **divId**: The identifier of the div that contains the form.
   - **onsuccessCallback**(optional): A JavaScript function executed after the process is started. This function receives the server response as a parameter.
   - **onerrorCallback** (optional): A JavaScript function executed if any error occurs and it is impossible to start the process. This function receives the server response as a parameter.
3. **showTaskForm(hostUrl, taskId, divId, onsuccessCallback, onerrorCallback):** Makes a call to the REST endpoint to obtain the form URL. If it receives a valid response, it embeds the task form in the stated div. You need these parameters:
   - **hostURL**: The URL of the Business Central instance that holds the deployments.
   - **taskId**: The identifier of the task to show the form.
   - **divId**: The identifier of the div that has to contain the form.
   - **onsuccessCallback** (optional): A JavaScript function executed if the form is going to be rendered. This function receives the server response as a parameter.
   - **onerrorCallback** (optional): A JavaScript function executed if any error occurs and it is impossible to render the form. This function receives the server response as a parameter.
4. **claimTask(divId, onsuccessCallback, onerrorCallback):** Claims the task whose form is being rendered. You need these parameters:
   - **divId**: The identifier of the div that contains the form.
   - **onsuccessCallback** (optional): A JavaScript function executed after the task is claimed. This function receives the server response as a parameter.

- **onerrorCallback** (optional): A JavaScript function executed if any error occurs and it is impossible to claim the task. This function receives the server response as a parameter.

5. **startTask(divId, onsuccessCallback, onerrorCallback):** Starts the task whose form is being rendered. You need these parameters:
   - **divId**: The identifier of the div that contains the form.
   - **onsuccessCallback** (optional): A JavaScript function executed after the task is claimed. This function receives the server response as a parameter.
   - **onerrorCallback** (optional): A JavaScript function executed if any error occurs and it is impossible to claim the task. This function receives the server response as a parameter.
6. **releaseTask(divId, onsuccessCallback, onerrorCallback):** Releases the task whose form is being rendered. You need these parameters:
   - **divId**: The identifier of the div that contains the form.
   - **onsuccessCallback** (optional): A JavaScript function executed after the task is claimed. This function receives the server response as a parameter.
   - **onerrorCallback** (optional): A JavaScript function executed if any error occurs and it is impossible to claim the task. This function receives the server response as a parameter.
7. **saveTask(divId, onsuccessCallback, onerrorCallback):** Submits the form and saves the state of the task whose form is being rendered. You need these parameters:
   - **divId**: The identifier of the div that contains the form.
   - **onsuccessCallback** (optional): A JavaScript function executed after the task is claimed. This function receives the server response as a parameter.
   - **onerrorCallback** (optional): A JavaScript function executed if any error occurs and it is impossible to claim the task. This function receives the server response as a parameter.
8. **completeTask(divId, onsuccessCallback, onerrorCallback):** Submits the form and completes task whose form is being rendered. You need these parameters:
   - **divId**: The identifier of the div that contains the form.
   - **onsuccessCallback** (optional): A JavaScript function executed after the task is claimed. This function receives the server response as a parameter.
   - **onerrorCallback** (optional): A JavaScript function executed if any error occurs and it is impossible to claim the task. This function receives the server response as a parameter.
9. **clearContainer(divId):** Cleans the div content and the related data stored on the component. You need these parameters:
   - **divId**: The identifier of the div that contains the form.

4.9. VARIABLES

Variables are elements that serve for storing a particular type of data during runtime. The type of data a variable contains is defined by its data type.

Just like any context data, every variable has its scope that defines its visibility. An element, such as a process, sub-process, or task can only access variables in its own and parent contexts: variables defined in the element's child elements cannot be accessed. Therefore, when an elements requires access to a variable on runtime, its own context is searched first. If the variable cannot be found directly in the element's context, the immediate parent context is searched. The search continues to "level up" until the process context is reached; in case of global variables, the search is performed directly on the session container. If the variable cannot be found, a read access request returns **`null`** and a write access produces an error message, and the process continues its execution. Variables are searched for based on their ID.

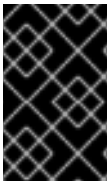
In Red Hat JBoss BPM Suite, variables can live in the following contexts:

- Session context: *Global variables* are visible to all process instances and assets in the given session and are intended to be used primarily by business rules and by constraints. The are created dynamically by the rules or constraints.
- Process context: *Process variables* are defined as properties in the BPMN2 definition file and are visible within the process instance. They are initialized at process creation and destroyed on process finish.
- Element context: *Local variables* are available within their process element, such as an activity. They are initialized when the element context is initialized, that is, when the execution workflow enters the node and execution of the **`onEntry`** action finished if applicable. They are destroyed when the element context is destroyed, that is, when the execution workflow leaves the element.
  Values of local variables can be mapped to global or process variables using the assignment mechanism (for more information, see Section 4.12, "Assignment"). This allows you to maintain relative independence of the parent element that accommodates the local variable. Such isolation may help prevent technical exceptions.

4.9.1. Global Variables

Global variables (also known as globals) exist in a knowledge session and can be accessed and are shared by all assets in that session. Global variables belong to the particular session of the Knowledge Base and they are used to pass information to the engine.

Every global variable defines its ID and item subject reference. The ID serves as the variable name and must be unique within the process definition. The item subject reference defines the data type the variable stores.



IMPORTANT

The rules are evaluated at the moment the fact is inserted. Therefore, if you are using a global variable to constrain a fact pattern and the global is not set, the system returns a **`NullPointerException`**.

4.9.1.1. Creating Global Variables

Global variables are initialized either when the process with the variable definition is added

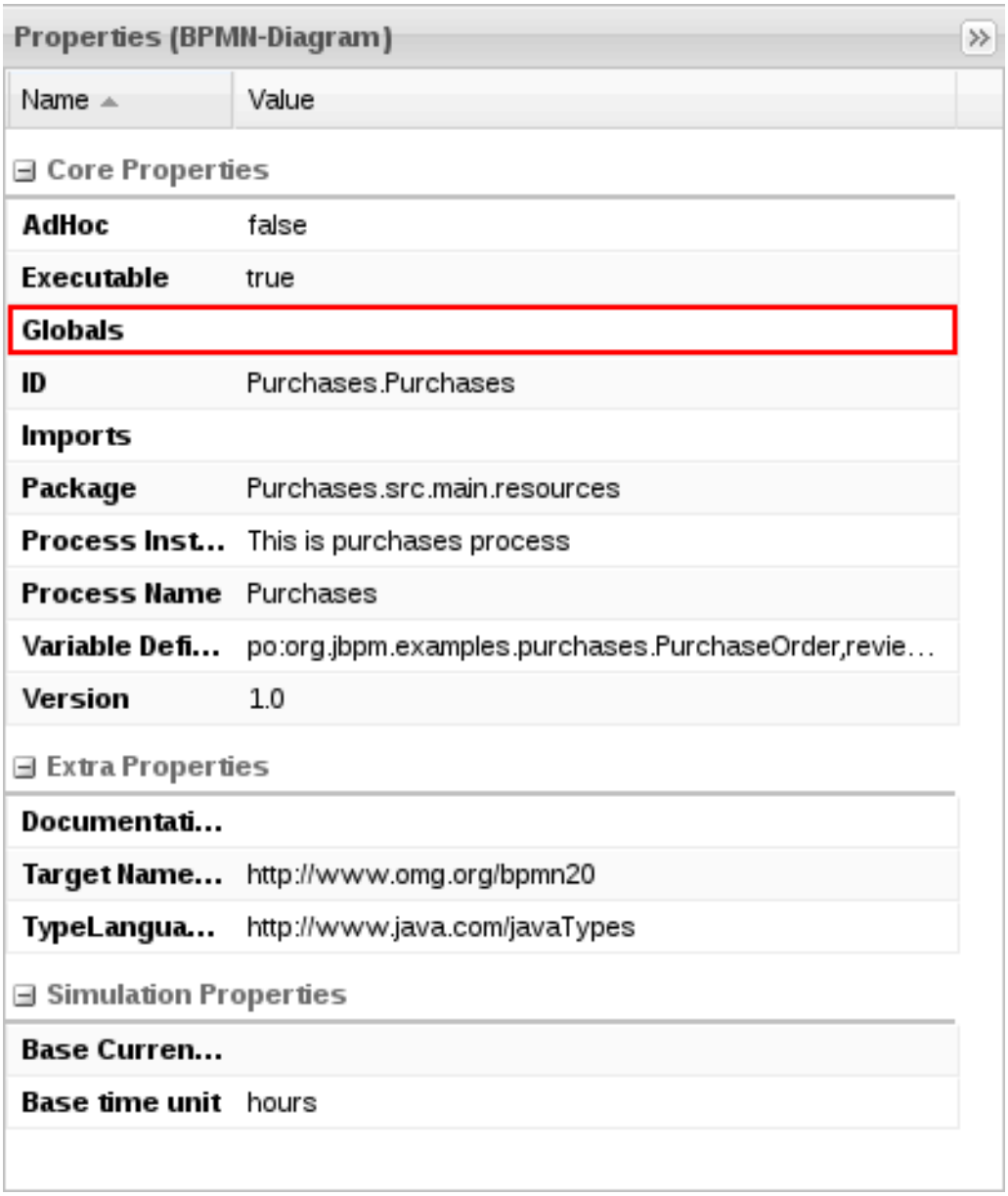
to the session or when the session is initialized with globals as its parameters. Values of global variables can be changed typically during the assignment, which is a mapping between a process variable and an activity variable. The global variable is then associated with the local activity context, local activity variable, or by a direct call to the variable from a child context.

Procedure: Defining Globals in Process Designer

To define a global variable, do the following:

1. In Business Central, go to **Authoring → Project Authoring**.
2. Open the respective process in **Process Designer**.
3. Click « in the right hand corner of the **Process Designer** and in the **Properties (BPMN-Diagram)** panel that opens, locate the **Globals** property.

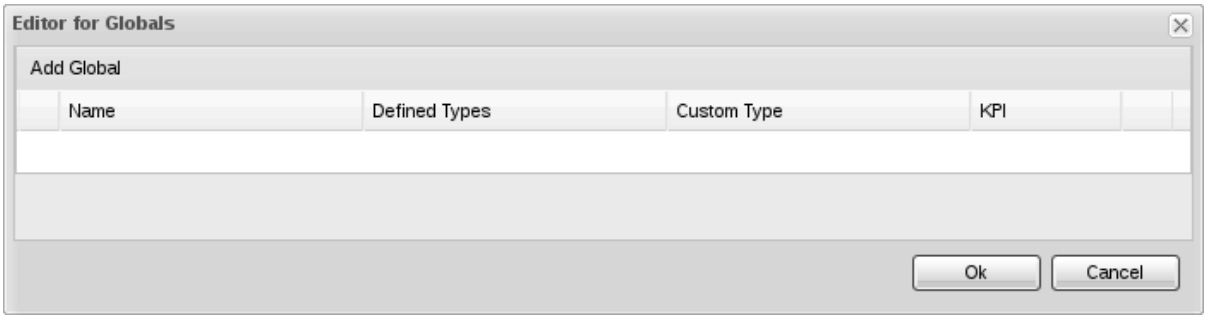
   Figure 4.20. Globals property in the Properties (BPMN-Diagram) panel



4. Click the empty value cell and expand the **Editor for Globals** window by clicking the arrow on the right side.

5. In the **Editor for Globals** window, click **Add Global** at the top and define the variable details.

Figure 4.21. Editor for Globals window



6. Click **Ok** to add the global variable.

Procedure: Defining and Initializing Global Variables Using API

To define and initialize global variables at a process instantiation using the API, do the following:

1. Define the variables as a **`Map`** of **`String`** and **`Object`** values.
2. Provide the map as a parameter to the **`startProcess()`** method.

Example 4.2. Instantiating Process With Global Variable

```
Map<String, Object> params = new HashMap<String, Object>();
params.put("var", "variable value");
ksession.startProcess("Process Definition Name", params);
```

4.9.1.2. Accessing Globals

```
processInstance.getContextInstance().getVariable("globalStatus")
```

4.9.1.3. Process variables

A Process variable is a variable that exists in a Process context and can be accessed by its Process or its child elements: Process variables belong to the particular Process instance and cannot be accessed by other Process instances. Every Process variable defines its ID and item subject reference: the ID serves as the variable name and must be unique within the Process definition. The item subject reference defines the data type the variable stores.

Process variables are initialized when the Process instance is CREATED. Their value can be changed by the Process Activities using the Assignment, when the global variable is associated with the local Activity context, local Activity variable, or by a direct call to the variable from a child context.

4.9.2. Local Variables

A local variable is a variable that exists in a child element context of a Process and can be accessed only from within this context: local variables belong to the particular element of a Process.

For Tasks, with the exception of the Script Task, the user can define local variable in the **DataInputSet** and **DataOutputSet** parameters: DataInputSet define variables that enter the Task and therefore provide the entry data needed for the Task execution, while the DataOutputSet variables can refer to the context of the Task after execution to acquire output data.

User Tasks typically present data related to the User Task to the actor that is executing the User Task and usually also request the actor to provide result data related to the execution. To request and provide such data, you can use Task forms and map the acquired data into the DataInputSet parameter to serve as input data of the User Task and into the DataOutputSet parameter from the User Task namespace back to the parent namespace to serve as the User Task output data (see Section 4.12, “Assignment”).

INITIALIZATION OF LOCAL VARIABLES

Local variables are initialized when the Process element instance is CREATED. Their value can be changed by their parent Activity by a direct call to the variable.

4.9.2.1. Accessing Local Variables

To set a variable value, call the respective setter on the variable field from the Script Activity; for example, **person.setAge(10)** sets the **Age** field of the **person** global variable to **10**.

4.10. ACTION SCRIPTS

Action scripts are pieces of code that define the **Script** property or an Element’s interceptor action. They have access to globals, the Process variables, and the predefined variable **kcontext**. Accordingly, **kcontext** is an instance of ProcessContext class and the interface content can be found at the following location: Interface ProcessContext.

Currently, dialects Java and MVEL are supported for action script definitions. Note that MVEL accepts any valid Java code and additionally provides support for nested access of parameters, for example, the MVEL equivalent of Java call **person.getName()** is **person.name**. It also provides other improvements over Java and MVEL expressions are generally more convenient for the business user.

Example 4.3. Action script that prints out the name of the person

```
// Java dialect
System.out.println( person.getName() );

//  MVEL dialect
System.out.println( person.name );
```

4.11. INTERCEPTOR ACTIONS

For every activity, you can define the following actions:

- **On Entry Actions**, which are executed before the activity execution starts, after the activity receives the token.

- **`On Exit Actions`**, which are executed after the activity execution, before the outgoing flow is taken.

You can define both types of actions in the **Properties** tab of the activity. You can define them either in Java or MVEL, and set the language in the **Script Language** property.

4.12. ASSIGNMENT

The assignment mechanism allows you to pass data into, and retrieve data out of, Activities in Business Processes. Assignments that pass data into Activities are executed before the Activity itself is executed. Assignments map from Business Process variables to local data items in activities, known as DataInputs. Assignments that retrieve data from Activities are executed after the Activity has executed. They map from local data items in activities, known as DataOutputs, to Business Process variables.

4.12.1. Data I/O Editor

The Data I/O Editor is the dialog used to define Activity DataInputs and DataOutputs, as well as the mappings between them and process variables.

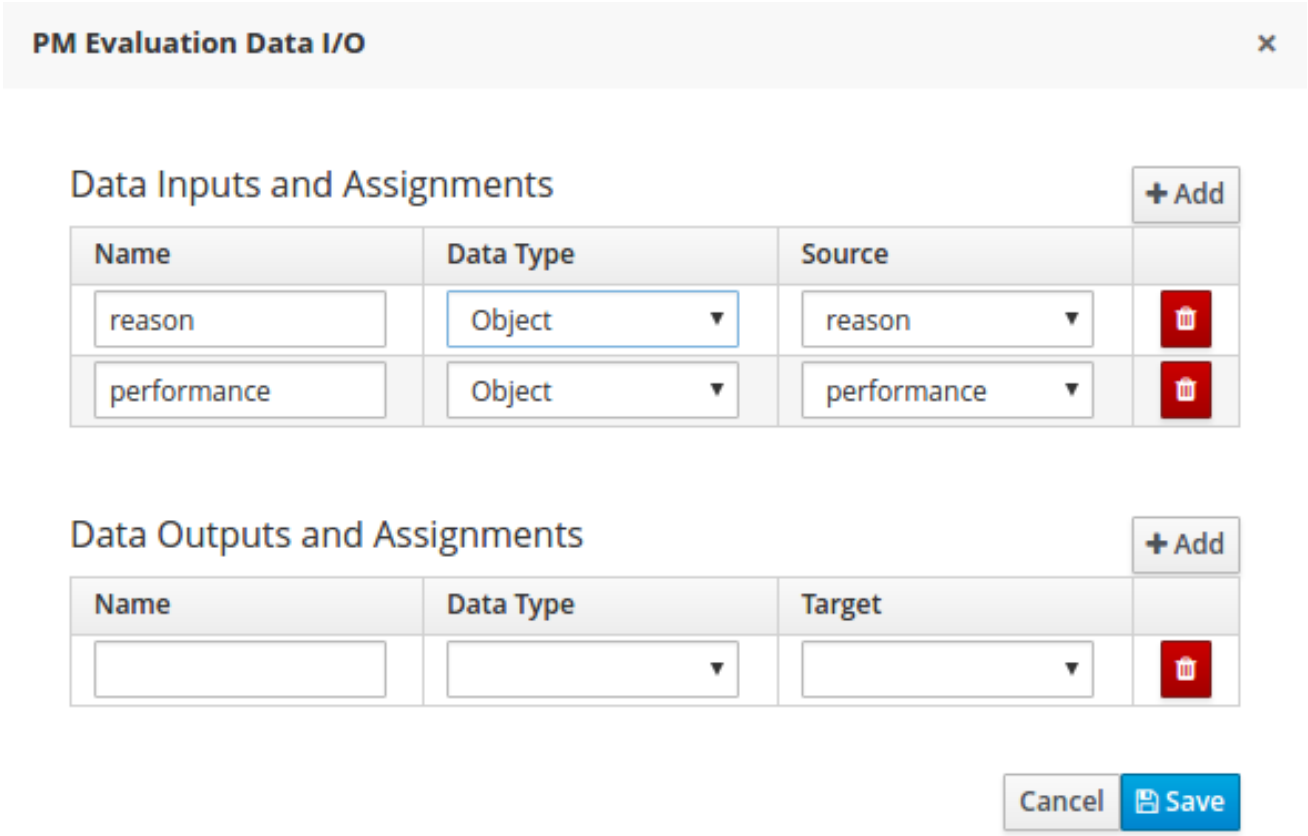
Like Process Variables, DataInputs and DataOutputs have a name and data-type, such as Integer, String, or a subclass of Java Object, such as a user-defined Data Object created within JBoss BPM Suite. The data-types of DataInputs and DataOutputs should match the data-types of the Process Variables which they are mapped to or from. Their names may be the same as the corresponding Process Variables, but this is not a requirement.

Process Variables are defined in the **Variable Definitions** property of the Business Process. Element DataInputs and DataOutputs are defined in one of three properties of Activities, depending on the element type:

- Elements such as **`User Tasks`** and **`Call Activities`**, which have both DataInputs and DataOutputs, use a property called **`Assignments`**.
- Elements such as **`Start Events`** and **`Intermediate Catch Events`**, which have DataOutputs but do not have DataInputs, use a property called **`DataOutputAssociations`**.
- Elements such as **`End Events`** and **`Intermediate Throw Events`**, which have DataInputs but do not have DataOutputs, use a property called **`DataInputAssociations`**.

The **`Assignments`**, **`DataOutputAssociations`**, and **`DataInputAssociations`** properties are all edited in the Data I/O Editor. DataInputs can have values assigned to them either by mapping from process variables or by assigning constant values to them. DataOutputs are mapped to Process Variables.

To define the DataInputs, DataOutputs and Assignments for an Element, select the Element in the Business Process and click the  button to open the Data I/O Editor. Data Input Assignments and Data Output Assignments can be added by clicking the **Add** button.



You can also open the Data I/O Editor to edit the Data Inputs and/or Outputs by editing the appropriate property for the activity: **`Assignments`**, **`DataOutputAssociations`**, or **`DataInputAssociations`**.

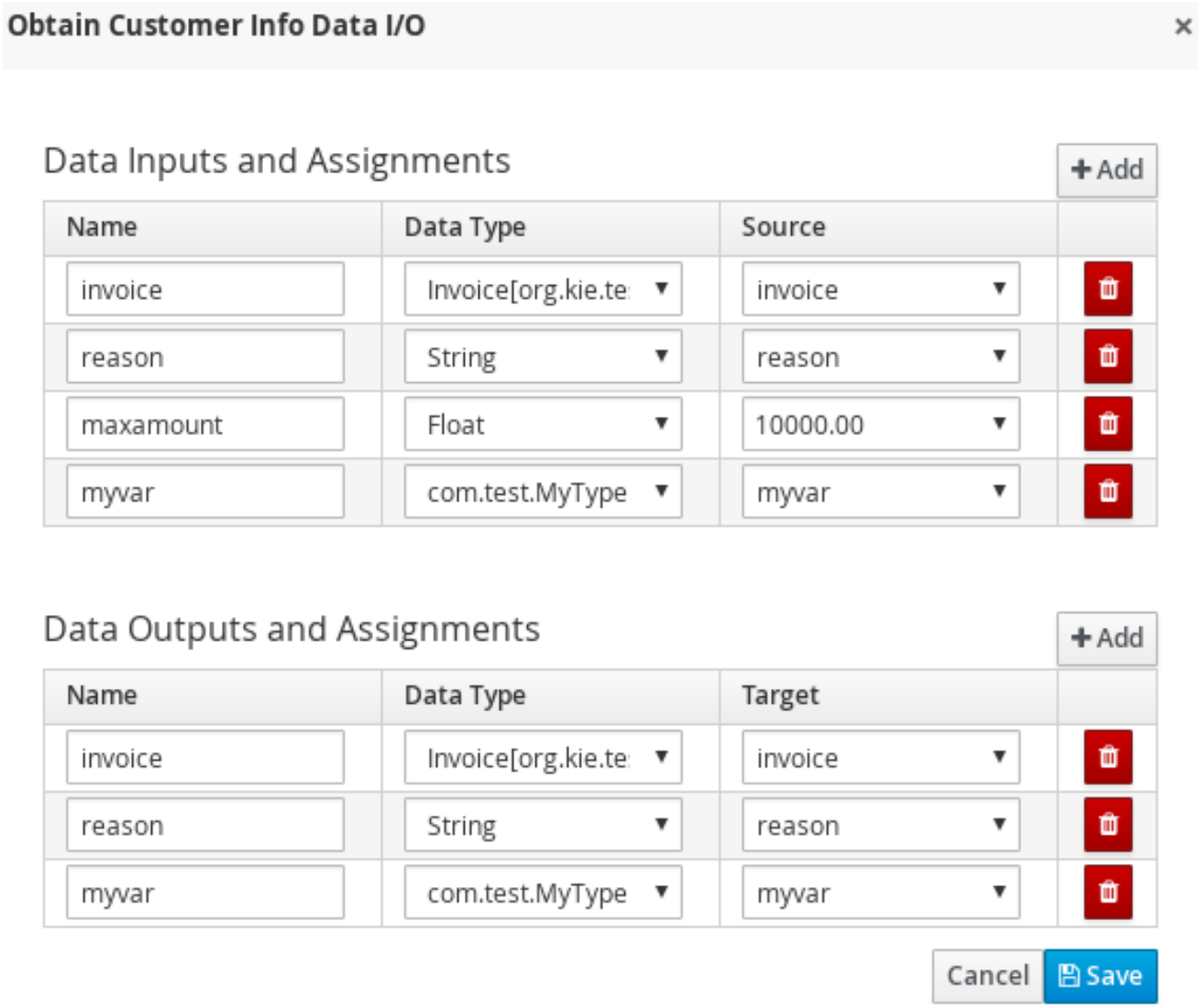
NOTE

The Data I/O Editor tool is available in Red Hat JBoss BPM Suite 6.2 or better.

4.12.2. Data I/O Editor Example

In the following example, the Data I/O Editor has been used to create some Data Inputs and Data Outputs for the user activity **`Check Invoice`**. The example makes use of two process variables that have been defined in the process:

- **`invoice`** with the type **`org.kie.test.Invoice;`**
- **`reason`** with the type **`String`**



The following Data Inputs have been added:

- **`invoice`**
- **`reason`**
- **`maxamount`**
- **`myvar`**

The Data Inputs and Data Outputs are linked to the corresponding process variables by setting the **Source** and **Target** fields in the dialog.

The Data I/O Editor allows you to create and assign a constant to a Data Input when setting the **Source** column for a Data Input. This is demonstrated by the **`maxamount`** Data Input, that has the constant **`1000.00`**, which will be assigned to it at runtime.

The **`myvar`** Data Input and Data Output demonstrates a custom **Data Type** **`com.test.MyType`**, which is entered in the dialog by the user.

4.13. CONSTRAINTS

A constraint is a boolean expression that is evaluated when the element with the constraint is executed. The workflow depends on the result of the evaluation, that is **`true`** or **`false`**.

There are two types of constraints:

- **`Code constraints`**, which are defined in Java or MVEL, and have access to the data in the working memory, including the global and process variables.

  Example 4.4. Java Code Constraint

  ```
  return person.getAge() > 20;
  ```

  Example 4.5. MVEL Code Constraint

  ```
  return person.age > 20;
  ```

- **`Rule constraints`**, which are defined in the form of DRL rule conditions. They have access to data in the working memory, including the global variables. However, they cannot access the variables in the process directly, but through the process instance. To retrieve the reference of the parent process instance, use the **`processInstance`** variable of the type **`WorkflowProcessInstance`**. Note that you need to insert the process instance into the session and update it if necessary, for example, using Java code or an on-entry, on-exit, or explicit action in your process.

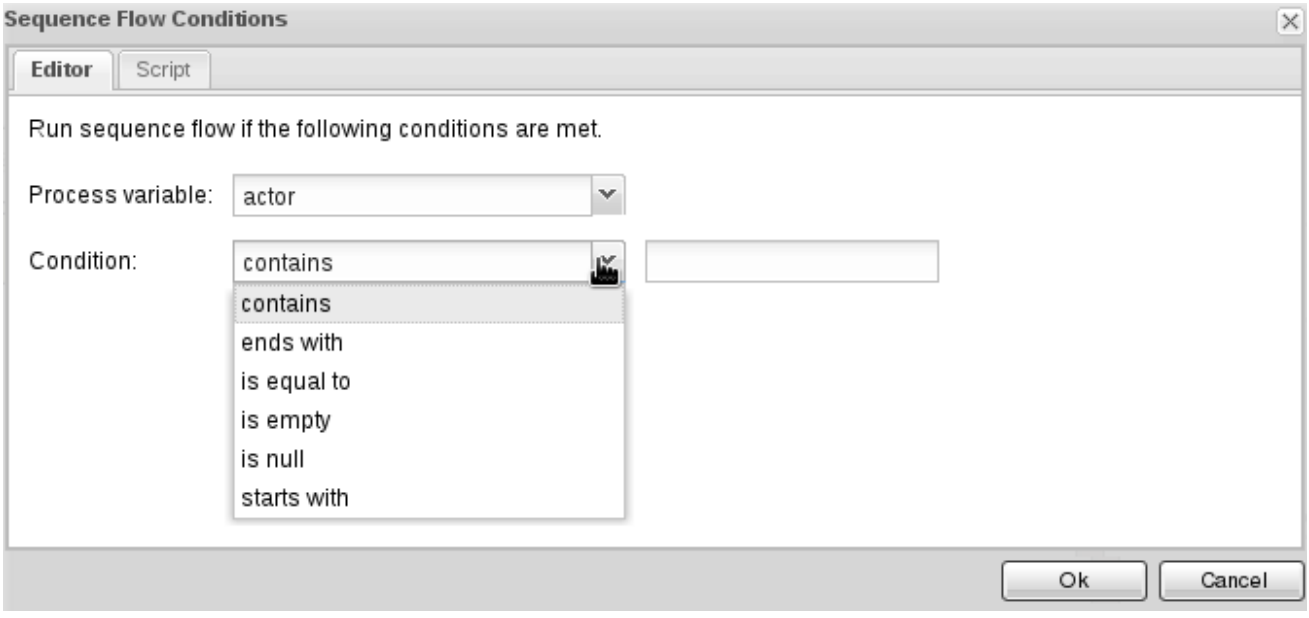
  Example 4.6. Rule Constraint with Process Variable Assignment

  ```
  import org.kie.api.runtime.process.ProcessInstance;
  import org.kie.api.runtime.process.WorkflowProcessInstance;
  ...
  processInstance : WorkflowProcessInstance()
  Person( name == ( processInstance.getVariable("name") ) )
  ```

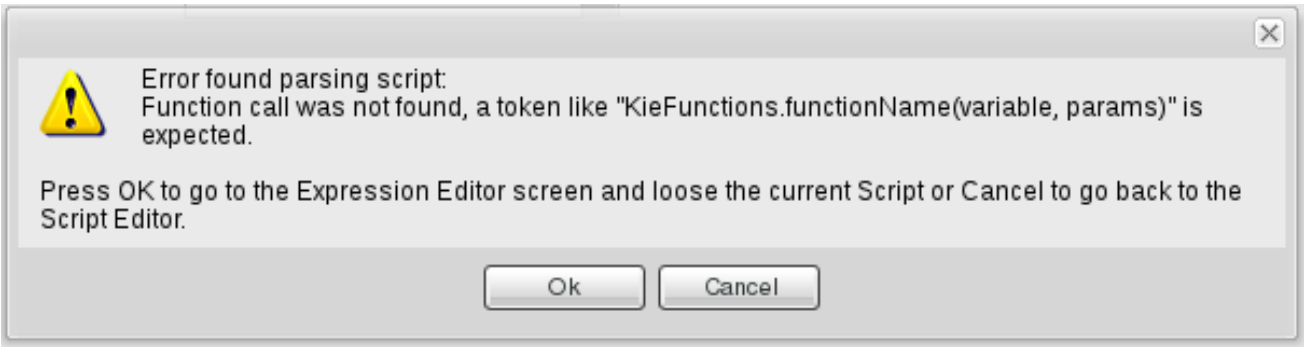
  This rule constraint retrieves the process variable **`name`**.

Red Hat JBoss BPM Suite includes a script editor for Java expressions. The constrain condition allows code constraints for scripts in Java as demonstrated by the editor below.

Figure 4.22. Script Editor



When a Java script cannot be represented by the editor, the following alert appears:



4.14. DATA MODELS

Data models are models of data objects. A data object is a custom complex data type (for example, a Person object with data fields Name, Address, and Date of Birth).

Data models are saved in data models definitions stored in your Project. Red Hat JBoss BPM Suite provides the Data modeler, a custom graphical editor, for defining data objects.

4.14.1. Data Modeler

The Data Modeler is the built-in editor for creating data objects as part of a Project data model from Business Central. Data objects are custom data types implemented as POJOs. These custom data types can then be used in any resource (such as a Process) after importing them.

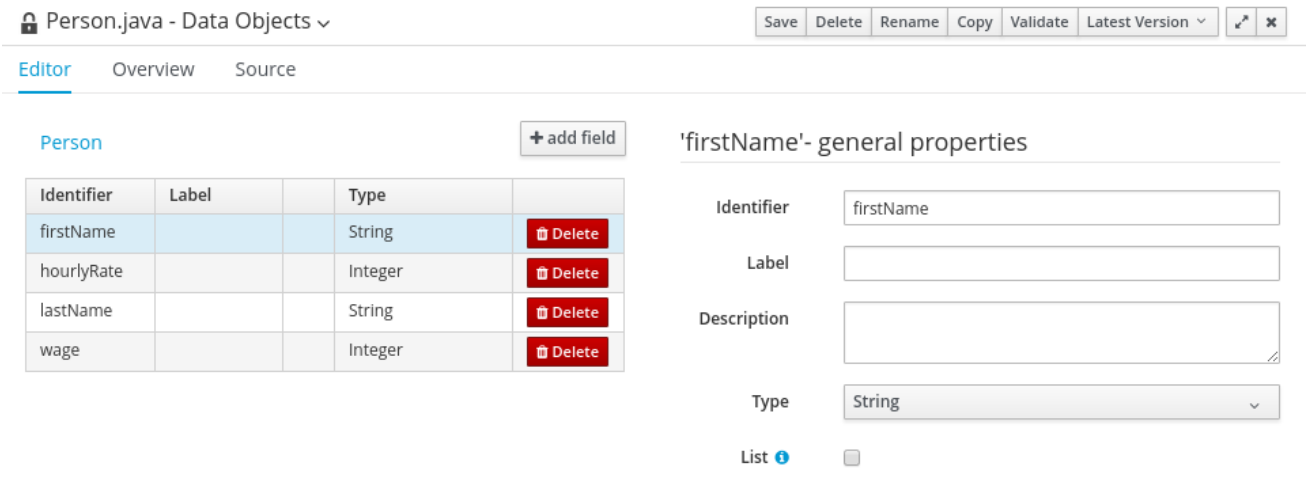
To open the editor, open the Project Authoring perspective, click **New Item** → **Data Object** on the perspective menu. If you want to edit an existing model, these files are located under **Data Objects** in **Project Explorer**.

You will be prompted to enter the name of this model object when creating a new model, and asked to select a location for it (in terms of the package). On successful addition, it will bring up the editor where you can create fields for your model object.

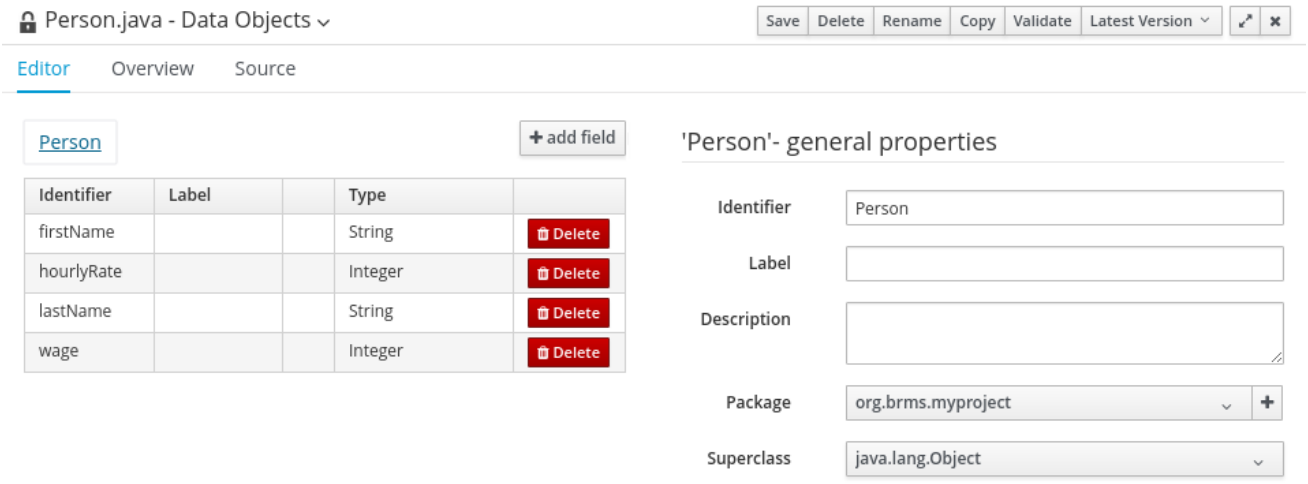
The Data Modeler supports roundtrips between the **`Editor`** and **`Source`** tabs, along with source code preservation. This allows you to make changes to your model in external tools, like JBDS, and the Data Modeler updates the necessary code blocks automatically.

In the main editor window the user can

- Add/delete fields
- Select a given field. When a field is selected then the field information will be loaded in all the domain editors.



- Select the data object class. For example, by clicking on the data object name (on the main window) instead of loading the field properties, the domain editors will load the class properties.



4.14.2. Annotations in Data Modeler

Red Hat JBoss BPM Suite supports all Drools annotations by default, and can be customized using the **Drools & jBPM** domain screen. For further information about available domain screens, see Section 4.14.5, “Data Object Domain Screens”.

If you want to add/edit custom or pre-defined annotations, you can switch to the source tab and modify the source code directly (in Red Hat JBoss Developer Studio or Business Central). You can also use the **Advanced** screen to manage arbitrary annotations.

When creating or adding fields to a persistable data object, the JPA annotations that are added by default will generate a model that can be used by Red Hat JBoss BPM Suite at runtime. In general, modifying the default configurations where the model will be used by processes is not recommended.

Red Hat JBoss BPM Suite 6.2 onwards supports the use of JPA specific annotations, with Hibernate available as the default JPA implementation. Other JPA annotations are also supported where the JPA provider is loaded in the classpath.

NOTE

When adding an annotation in the Data Modeler, the annotation class should be on the workbench classpath, or a project dependency can be added to a `.jar` file that has the annotation. The Data Modeler will run a validation check to confirm that the annotation is on the classpath, and the project will not build if the annotation is not present.

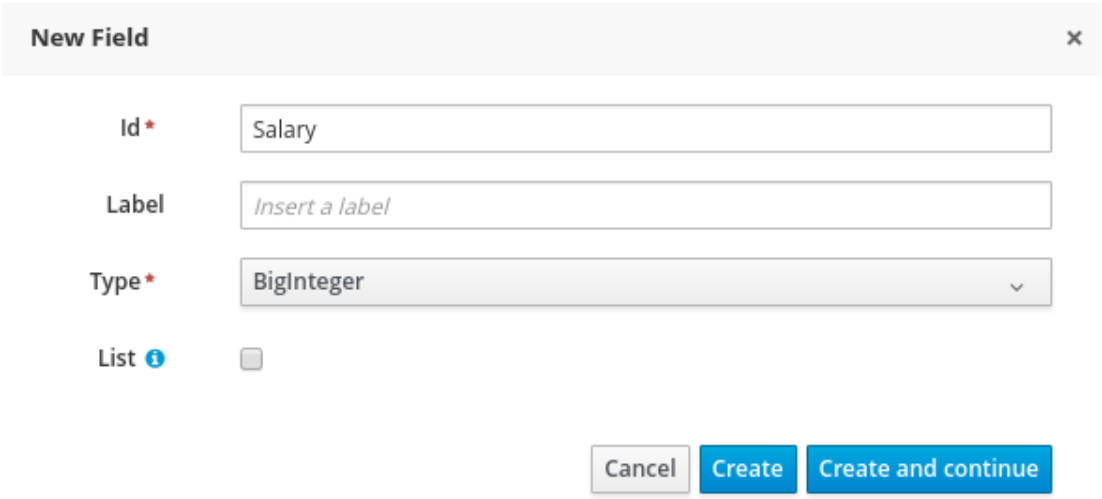
4.14.3. Creating Data Object (Not Persistable)

1. In the Project Authoring perspective, click **New Item** → **Data Object** on the perspective menu. Enter the name and the location (the name has to be unique across the package, but it is possible to have two data object with the same name in two different packages). Uncheck the **Persistable** box to prevent the Data Object from being made persistable. Press the **Ok** button.

2. Create fields of the data object:

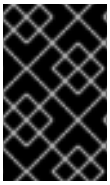
   a. By clicking **add field** button in the main editor window, you can add a field to the object with the attributes **Id**, **Label** and **Type**. Required attributes are marked with *.

   - **Id**: field ID unique within the data object
   - **Label**: label to be used in the**Fields** panel (optional)
   - **Type**: data type of the field



   b. Click the **Create** button (the new field is created and the**New field** window closes) or the **Create and continue** button (the new field is created and the **New field** window remains open, so it is possible to continue adding more fields).

Attributes can be edited selecting the attribute and editing the fields using the general properties screen to the right.



USING A DATA OBJECT

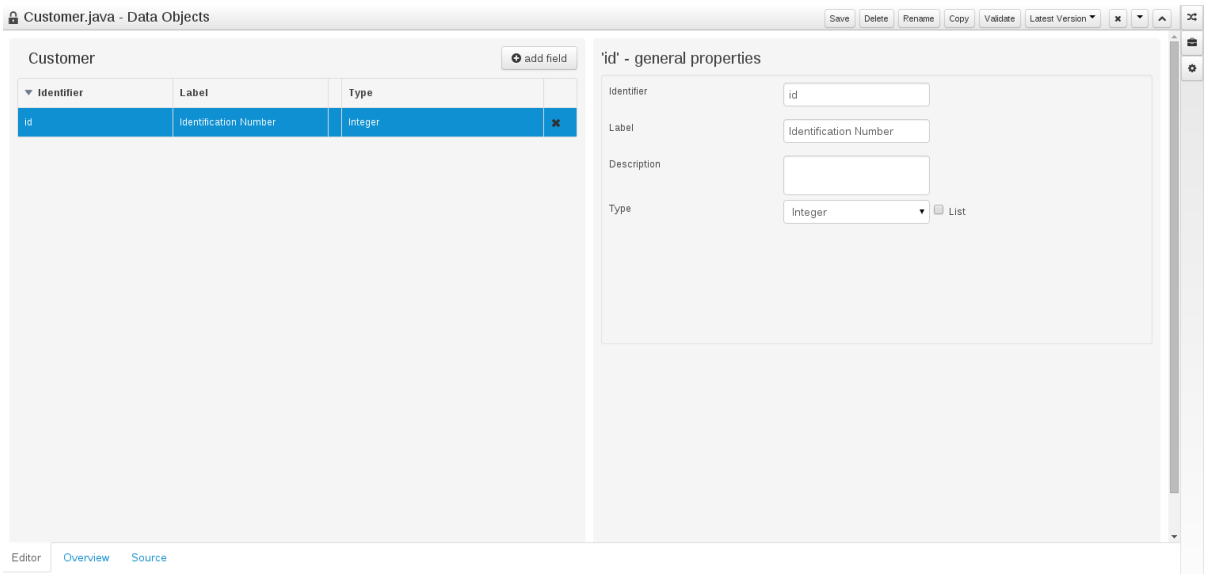
To use a data object, make sure you import the data model into your resource. This is necessary even if the data model lives in the same Project as your resource (Business Process).

4.14.4. Persistable Data Objects

From Red Hat JBoss BPM Suite 6.2 onwards, the Data Modeler is extended to support the generation of persistable Data Objects by checking the **Persistable** check box on the perspective menu when you create a new Data Object. Persistable Data Objects are based on the JPA specification. When the **Persistable** check box is selected when the object is created, the platform will set some configurations required for persistence by default.The object can also be made persistable after the object has already been createdThis can then be customized as required.

Procedure: Creating a Persistable Data Object

1. When creating a persistable Data Object, the Identifier field **`id`** will be generated automatically, along with the **`@Entity`** annotation.

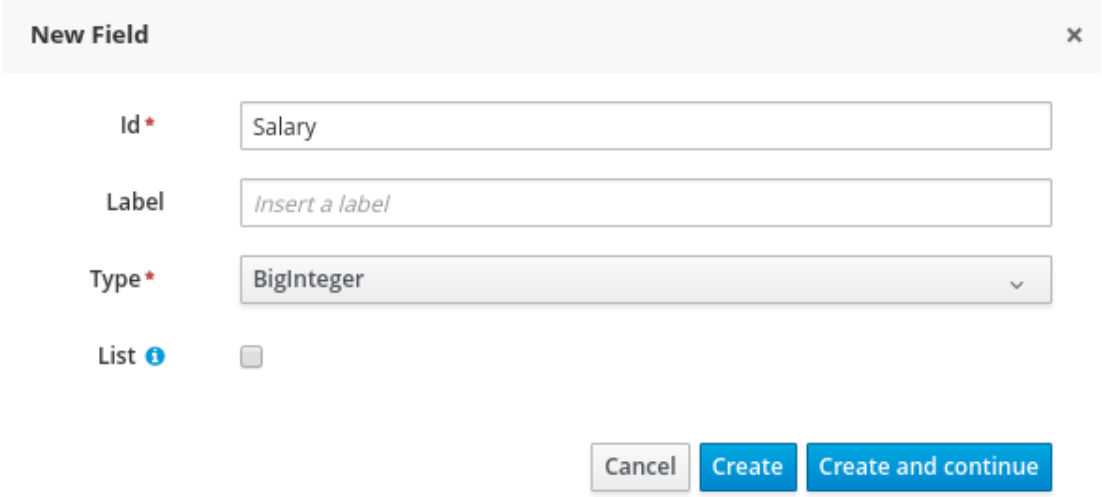


   Selecting the **List** option changes a property to have multiple values. For example, it can enable an attribute **`Name`**, to have a list of strings with multiple names.

2. Create fields of the data object:

   a. By clicking **add field** button in the main editor window, you can add a field to the object with the attributes **Id**, **Label** and **Type**. Required attributes are marked with *.

      - **Id**: field ID unique within the data object
      - **Label**: label to be used in the**Fields** panel (optional)
      - **Type**: data type of the field



b. Click the **Create** button (the new field is created and the**New field** window closes) or the **Create and continue** button (the new field is created and the **New field** window remains open, so it is possible to continue adding more fields).

Attributes can be edited selecting the attribute and editing the fields using the general properties screen to the right.

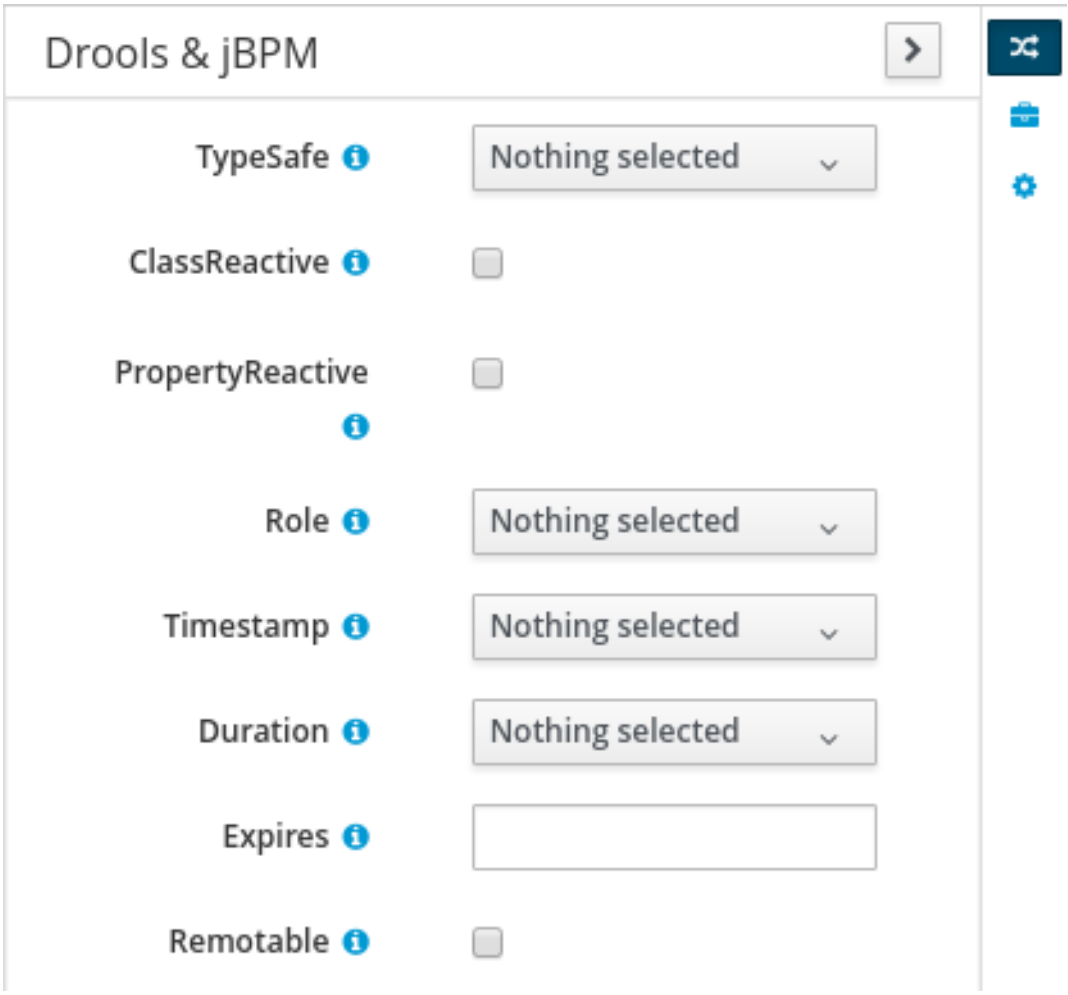
4.14.5. Data Object Domain Screens

The following domain screen tabs can be selected from the right side of the Data Object editor screen.

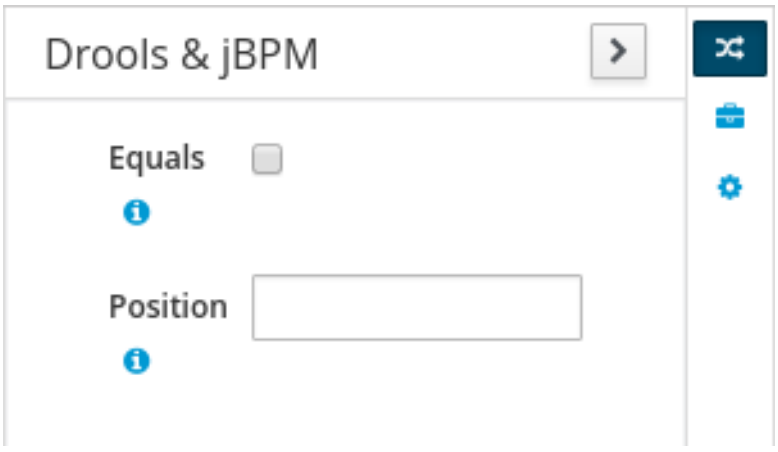
Drools & jBPM
The **Drools & jBPM** screen allows configuration of Drools-specific attributes.

The Data Modeler in Business Central supports the editing of pre-defined annotations of fact model classes and attributes. The following Drools annotations are supported, and can be customized using the **Drools & jBPM** interface:

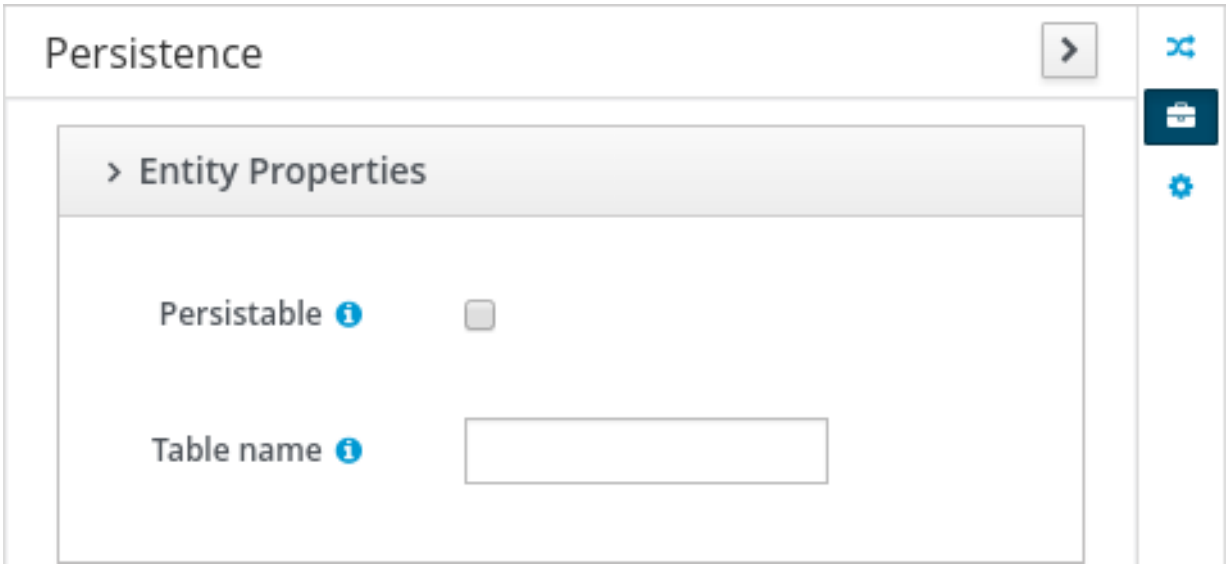
- **`TypeSafe`**
- **`Role`**
- **`Timestamp`**
- **`Duration`**
- **`Expires`**



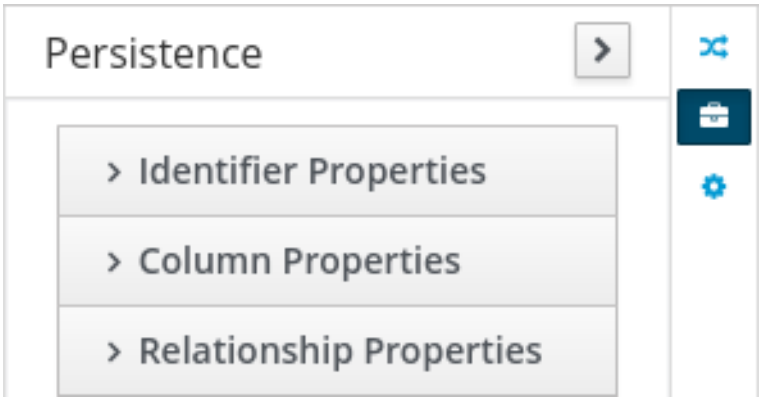
For the fields within the fact model, the **`position`** annotation is supported. The**Drools & jBPM** screen when a specific field is selected looks as follows:



Persistence
The **Persistence** screen can be used to configure attributes on basic JPA annotations for persistence. Fine tuning of annotations, or to add specific annotations, use the **Advanced** screen.



The **Persistence** screen when a specific field is selected looks as follows:



The following annotations can be managed via the **Persistence** screen.

Table 4.4. Type Annotations

Annotation	Automatically Generated when Data Object is Persistable
javax.persistence.Entity	Yes
javax.persistence.Table	No

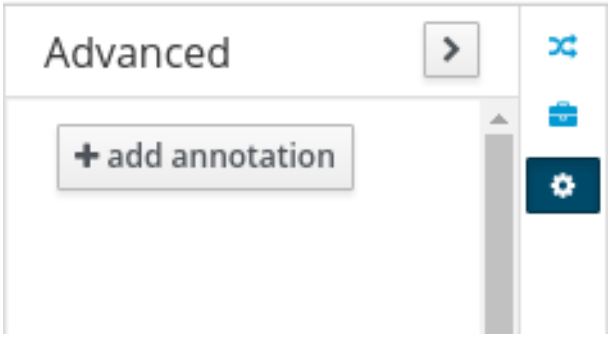
Table 4.5. Field Annotations

Annotation	Automatically Generated when Data Object is Persistable
javax.persistence.Id	Yes
javax.persistence.GeneratedValue	Yes

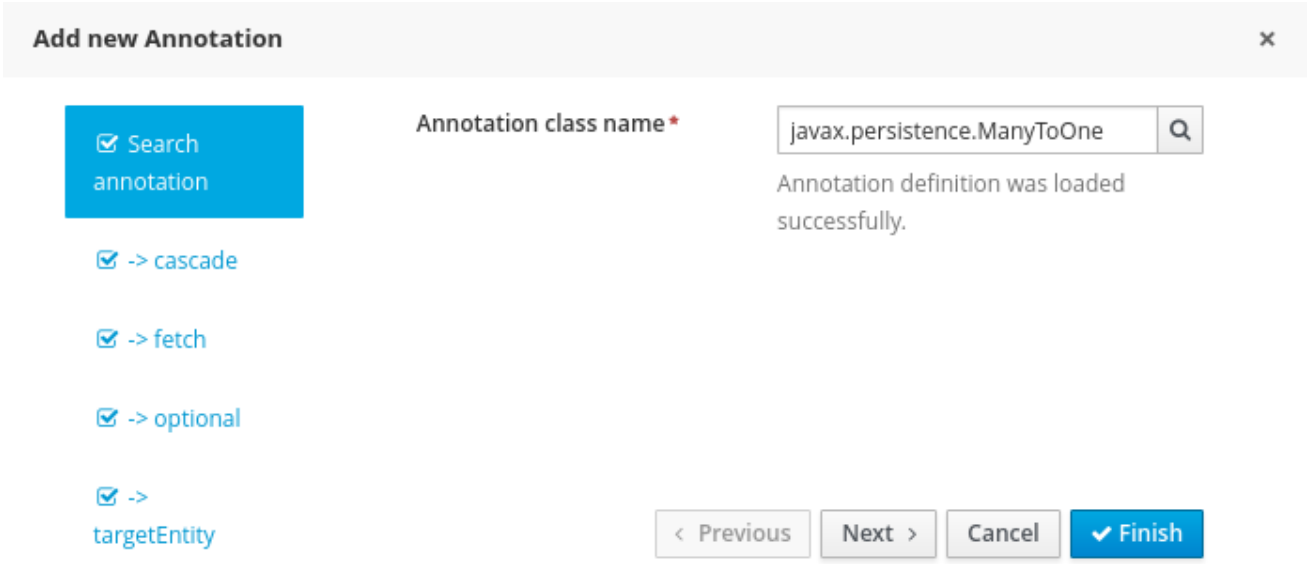
Annotation	Automatically Generated when Data Object is Persistable
javax.persistence.SequenceGenerator	Yes
javax.persistence.Column	No
javax.persistence.OneToOne	No
javax.persistence.OneToMany	Yes - when a field has one or multiple values
javax.persistence.ManyToOne	Yes - when a field has multiple values
javax.persistence.ManyToMany	No
javax.persistence.ElementCollection	Yes - generated by the UI when a new field has one or multiple of a base java type, such as Integer, Boolean, String. This annotation cannot be edited with the **Persistence** screen tool (use the **Advanced** screen tool instead).

All other JPA annotations can be added using the **Advanced** screen.

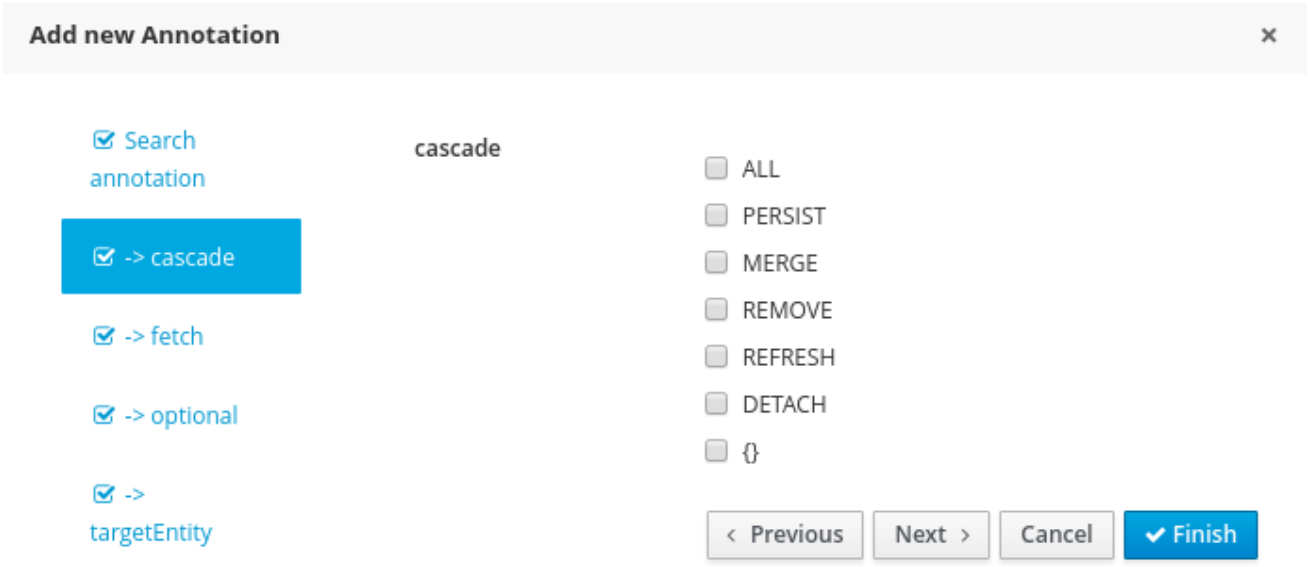
Advanced
The **Advanced** screen is used for fine-tuning of annotations. Annotations can be configured, added and removed using the Advanced screen. These can be any annotation that is on the classpath.



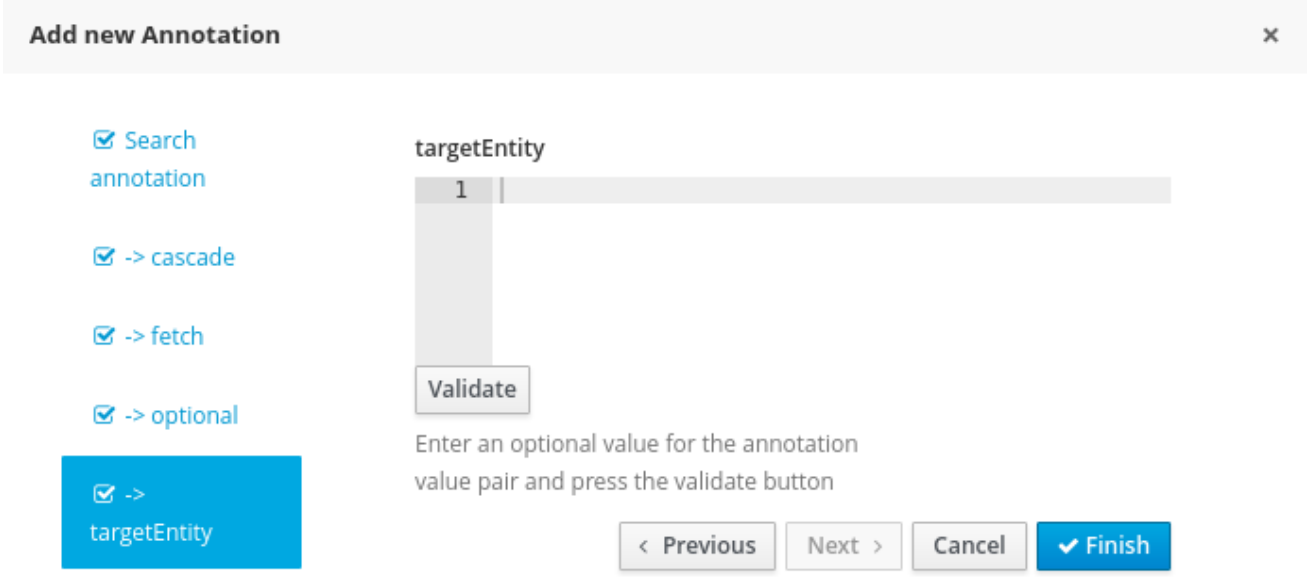
After you click on the **add annotation** option, the **Add new Annotation** window is displayed. It is required to enter a fully qualified class name of an annotation and by pressing the search icon, the annotation definition is loaded into the wizard. Then it is possible to set different annotation parameters (required parameters are marked with *).



If possible, the wizard will provide a suitable editor for the given parameters.



If it is not possible to provide a customized editor, the wizard will provide a generic parameter editor.



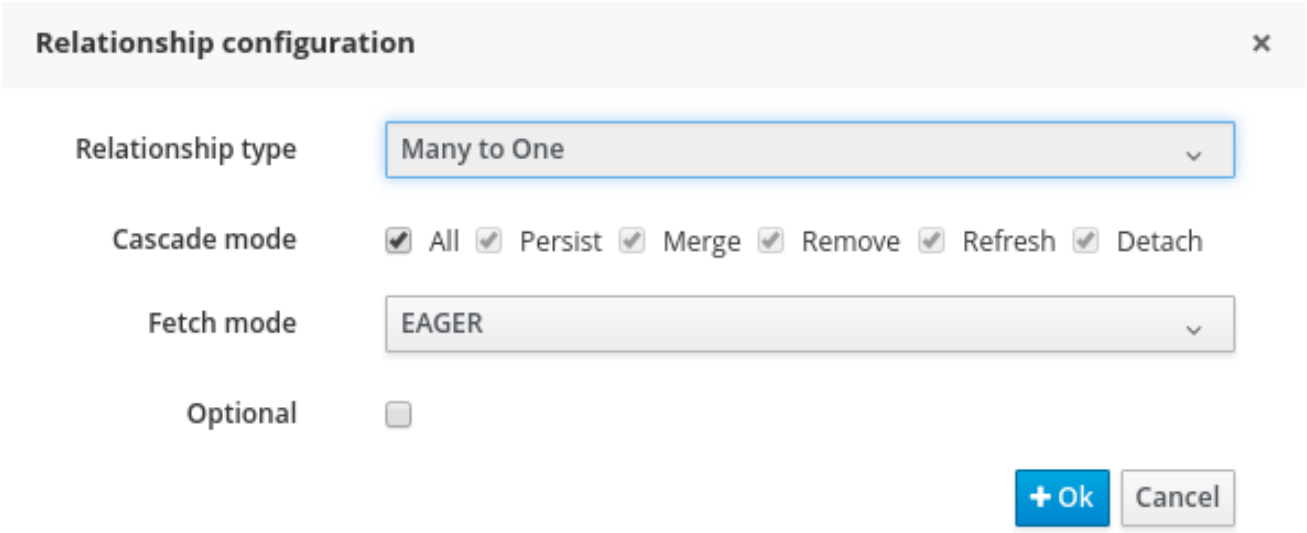
After you enter all the required parameters, the **Finish** button is enabled and the annotation can be added to the given field or data object.

4.14.6. Configuring Relationships Between Data Objects

When an attribute type is defined as another Data Object, the relationship is identified and defined by the symbol in the object attribute list. You can jump to the Data Object definition to view and edit by clicking on the icon.

Relationship customization is only relevant where the data object is persistable.

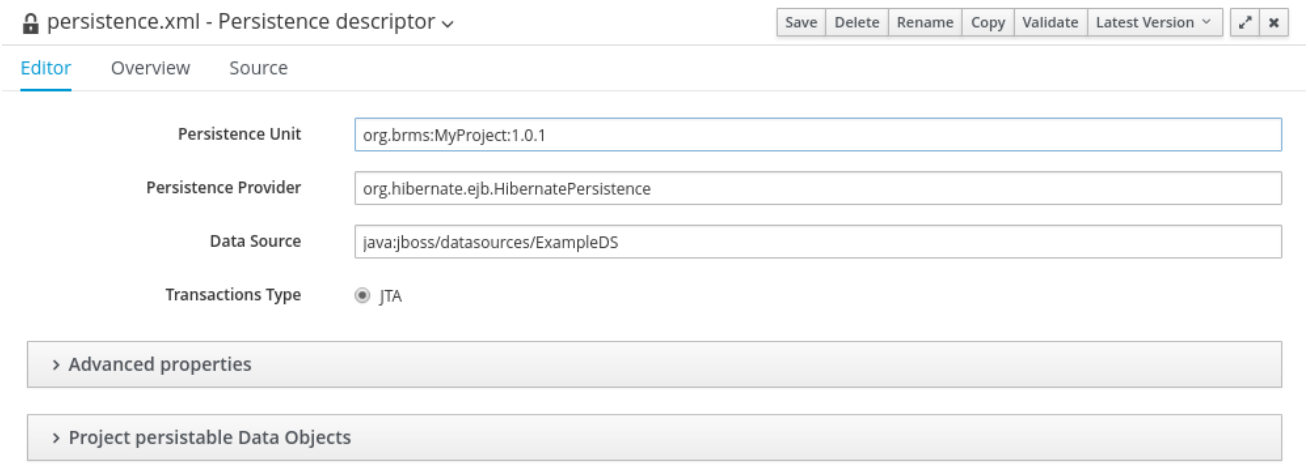
Relationships can be configured by selecting an attribute with a relationship and choosing the **Persistence** button on the right. Under**Relationship Properties**, click the **Relationship Type** property editing option.



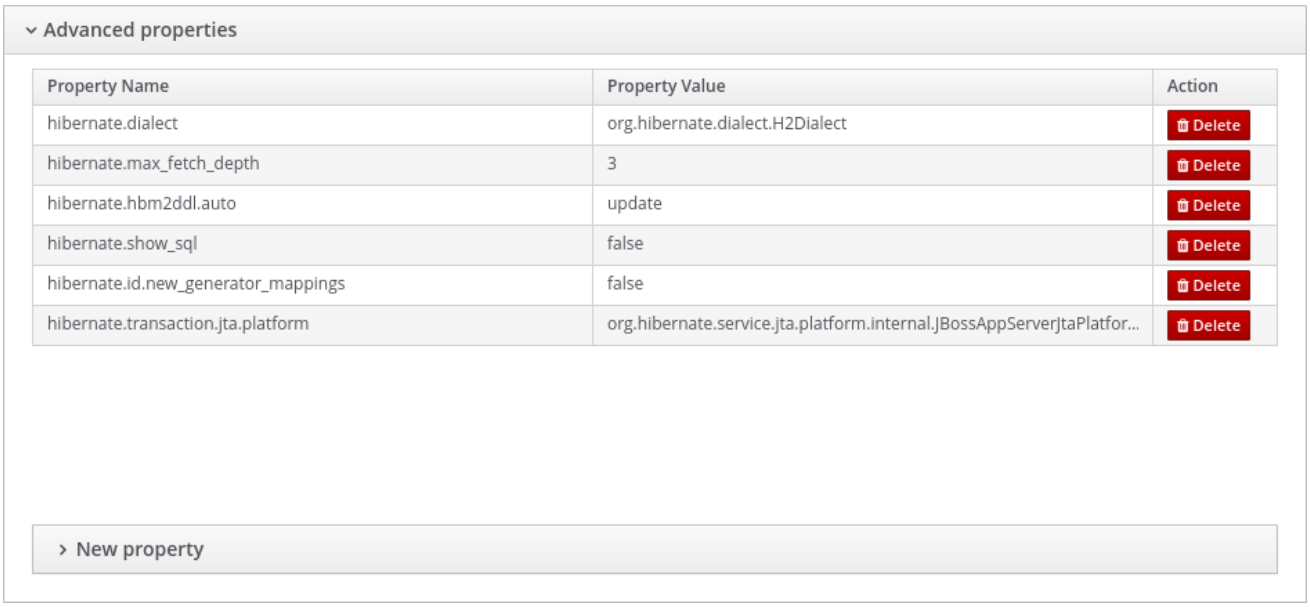
Attempting to delete a Data Object that has a relationship with another Data Object will show the **Usage Detected** screen. It is still possible to delete the object from here, however this will stop your project from building successfully until the resulting errors are resolved.

4.14.7. Persistence Descriptor

When creating persistent data objects, the **`persistence.xml`** file is created by default when the user opens the project editor and selects **Persistence Descriptor**. It can then be configured via the Persistence Descriptor. The Persistence Descriptor is accessible by opening the Project Editor, and clicking **Project Settings: Project General Settings → Persistence descriptor**.

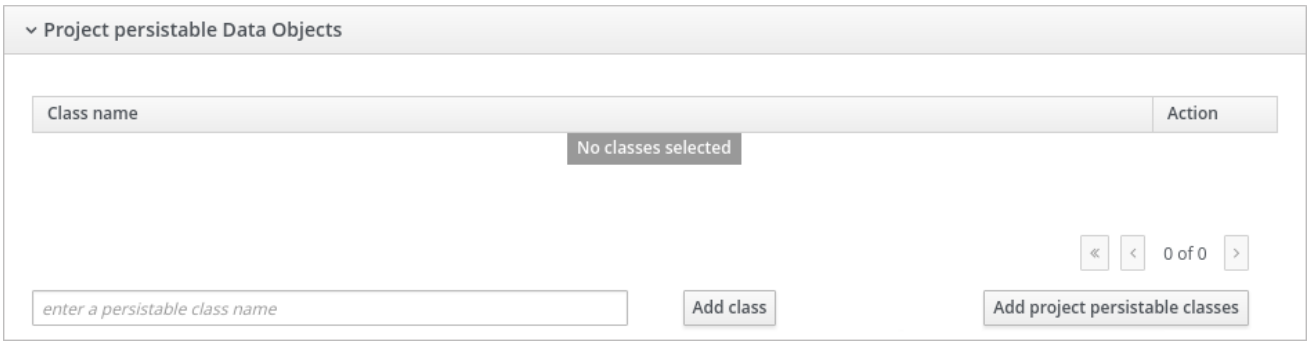


In the **Advanced properties** section, it is possible to change or delete all the properties generated by default and add new ones as well.



If you open the **Project persisteable Data Objects** section in the Persistence Descriptor, you will see that there are two available buttons:

- **Add class** enables the user to add arbitrary classes to the**`persistence.xml`** file to be declared as entities.
- **Add project persistable classes** will automatically load all the persisteable data objects in the current project.



4.14.8. Deployment Descriptor

Deployment Descriptor editor can also be accessed via the Project Editor menu, and allows configuration of the **`kie-deployment-descriptor.xml`** file for deployment in the jBPM runtime. Automatic configuration of the JPA Marshalling Strategies is only available in JBoss BPM Suite.

4.14.9. Data Sets

The data set functionality in Business Central defines how to access and parse data. Data sets serve as a source of data that can be displayed by the Dashbuilder displayer. You can add the Dashbuilder displayers to a custom perspective in the Plugin Management perspective. Note that the data set perspective is visible only to users of the Administrator group.

4.14.9.1. Managing Data Sets

To add a data set definition:

1. Log into Business Central and click **Extensions → Data Sets**.
2. Click **New Data Set**.
3. Select the provider type and click **Next**. Currently, the following provider types are supported:
   - Java Class – generate a data set from a Java class.
   - SQL – generate a data set from an ANSI-SQL compliant database.
   - CSV – generate a data set from a remote or local CSV file.
   - Elastic Search – generate a data set from Elastic Search nodes.
4. Complete the **Data Set Creation Wizard** and click **Test**.
5. Depending on what provider you chose, the configuration steps will differ. Once you complete the steps, click **Save** to create a data set definition.

To edit a data set:

1. Log into Business Central and click **Extensions** → **Data Sets**.
2. In **Data Set Explorer**, click on an existing data set and click **Edit**.
3. **Data Set Editor** opens. You can edit your data set in three tabs. Note that some of the tabs differ based on the provider type you chose. The following applies to the *CSV* data provider.
   - **CSV Configuration** – allows you to change the name of your data set definition, the source file, the separator, and other properties.
   - **Preview** – after you click **Test** in the **CSV Configuration** tab, the system executes the data set lookup call and if the data is available, you will see a preview. Notice two subtabs:
     - **Data columns** – allows you to customize what columns are part of your data set definition.
     - **Filter** – allows you to add a new filter.
   - **Advanced** – allows you to manage:
     - Caching – see Section 4.14.9.2, “Caching” for more information.
     - Cache life-cycle – see Section 4.14.9.3, “Data Refresh” for more information.

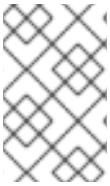
4.14.9.2. Caching

Red Hat JBoss BPM Suite data set functionality provides two cache levels:

- Client level
- Back end level

Client Cache
When turned on, the data set is cached in a web browser during the look-up operation. Consequently, further look-up operations do not perform any request to the back end.

Backend Cache
When turned on, the data set is cached by the Red Hat JBoss BPM Suite engine. This reduces the number of requests to the remote storage system.



NOTE

The Java and CSV data providers rely on back-end caching. As a result, back-end cache settings are not always visible in the **Advanced** tab of the **Data Set Explorer**.

4.14.9.3. Data Refresh

The refresh features allow you to invalidate cached data set data after a specified interval of time. The **Refresh on stale data** feature invalidates cached data when the back-end data changes.

4.15. DOMAIN-SPECIFIC TASKS

A domain-specific task is a task with custom properties and handling for a given domain or company. You can use it repeatedly in different business processes and accommodate interactions with other technical system.

In Red Hat JBoss BPM Suite, domain-specific task nodes are referred to as **`custom work items`** or **`custom service nodes`**.

When creating custom work items, define the following:

Work Item Handler

A work item handler is a Java class that defines how to execute a custom task. Tasks are executed in the Execution Engine, which contains a work item handler class, that defines how to handle the particular work item. For the Execution Engine to execute your custom work item, you need to:

- Create a work item handler class for the custom work item.
- Register the work item handler with the Execution Engine.

Work Item Definition

A work item definition defines how the custom task is presented (its name, icon, parameters, and similar attributes).

4.15.1. Work Item Definition

You can define a work item definition in:

- Red Hat JBoss Developer Studio Process Designer
- Web Process Designer

A work item has the following properties:

- **`name`** – A unique identifier in the given work item set.
- **`description`** – A description of your work item set.
- **`version`** – A version number.
- **`parameters`** – A set of work item parameters used as properties.
- **`displayName`** – A name displayed in the palette.
- **`icon`** – A path to the icon file.
- **`category`** – A name of the node in the palette into which the work item is added. If the defined category does not exit, a new category is created.
- **`defaultHandler`** – A **`WorkItemHandler`** class that is used to execute the work item.
- **`dependencies`** – The dependencies for the**`defaultHandler`** class.

4.15.2. Creating Custom Work Item Definition

JBoss Developer Studio Process Designer
To create a custom work item definition (WID) in JBoss Developer Studio Process Designer, follow these steps:

1. Create **WID_NAME.wid** in **META-INF**. For example, **$PROJECT_HOME/src/main/resources/META-INF/WID_NAME.wid**.
   This file is identical to a work item definition file created in Business Central.

2. Copy all the icons you want to use into **$PROJECT_HOME/src/main/resources/icons**.

Web Process Designer
To create a custom work item definition (WID) in the Web Process Designer, follow these steps:

1. Log into Business Central.

2. Click **Authoring → Project Authoring**.

3. Choose the organizational unit and repository of your project to view the assets in your project.

4. Click **WORK ITEM DEFINITIONS → WorkDefinitions**.
   The **WorkDefinitions** asset is created by default and contains a number of pre-set work item definitions.

5. The **Work Item Definitions** editor opens. Add your WID at the end, for example:

```
[
"name" : "Google Calendar",
"description" : "Create a meeting in Google Calendar",
"version" : "1.0",
"parameters" : [
 "FilePath" : new StringDataType(),
 "User" : new StringDataType(),
 "Password" : new StringDataType(),
 "Body" : new StringDataType()
],
"displayName" : "Google Calendar",
"icon" : "calendar.gif",
]
```

   Add the imports required by your WID. For example:

```
import
org.drools.core.process.core.datatype.impl.type.StringDataType;
import
org.drools.core.process.core.datatype.impl.type.ObjectDataType;
```

NOTE

You have to separate the previous definition with a comma "**,**". Otherwise, the validation will fail.

6. Click **Validate** to make sure your definition is correct.

7. Click **Save**.

To upload a custom icon for your work item definition, follow these steps:

1. Click **New Item** → **Uploaded file**.
2. In the **Create new Uploaded file** dialogue box, define the resource name, including file extension.
3. Click **Choose File** and upload the file (`png` or `gif`, 16x16 pixels).
4. Click **Ok**.

You can now refer to your icon in your WID. Your WID is in the Process Designer, in the **Service Tasks** section by default.

4.15.3. Work Item Handler

A work item handler is a Java class used to execute or abort (during asynchronous execution) work items. The class defines the business logic of the work item, for example how to contact another system and request information, which is then parsed into the custom task parameters. Every work item handler must implement `org.kie.api.runtime.process.WorkItemHandler`, which is a part of the KIE API.

For more information about work item handlers, see Appendix B. Service Tasks from Red Hat JBoss BPM Suite User Guide.

DIFFERENT WORK ITEM HANDLER FOR EVERY SYSTEM

You can customize the behavior of your work item by registering different work item handlers on different systems.

Red Hat JBoss BPM Suite comes with multiple work item handlers in the following modules:

- The `jbpm-bpm2` module in the `org.jbpm.bpmn2.handler` package contains the following work item handlers:
- ReceiveTaskHandler (for the BPMN <receiveTask> element)
- SendTaskHandler (for the BPMN <sendTask> element)
- ServiceTaskHandler (for the BPMN <serviceTask> element)
- The `jbpm-workitems` module in packages within `org.jbpm.process.workitem` contains, for example:
- ArchiveWorkItemHandler
- WebServiceWorkItemHandler
- TransformWorkItemHandler
- RSSWorkItemHandler
- RESTWorkItemHandler

- JavaInvocationWorkItemHandler
- JabberWorkItemHandler
- JavaHandlerWorkItemHandler
- FTPUploadWorkItemHandler
- ExecWorkItemHandler
- EmailWorkItemHandler

The work item handlers must define the **executeWorkItem()** and **abortWorkItem()** methods as defined by the **WorkItemHandler** interface. These are called during runtime on work item execution.

When a work item is executed, the following is performed:

1. Information about the task is extracted from the WorkItem instance.
2. The work item business logic is performed.
3. The Process instance is informed that the work item execution finished (as completed or aborted) using the respective method of the WorkItemManager:

```
public class GoogleCalendarHandler implements WorkItemHandler {

 @Override
 public void executeWorkItem(WorkItem workItem, WorkItemManager
manager) {

 Map<String, Object> results = new HashMap<String, Object>();
 // obtain parameters
 String filePath = (String) workItem.getParameter("FilePath");
 String user = (String) workItem.getParameter("User");

 // execute the custom logic here

 // pass results to next processing, for example

 Object result;
 results.put("Result", result);
 manager.completeWorkItem(workItem.getId(), results)
 }

 @Override
 public void abortWorkItem(WorkItem workItem, WorkItemManager
manager) {

   manager.abortWorkItem(workItem.getId());
 }

}
```

If you use the work item in a maven project, you need to declare the following dependency:

```
<dependency>
 <groupId>org.jbpm</groupId>
 <artifactId>jbpm-workitems</artifactId>
 <version>6.4.0.Final-redhat-3</version>
</dependency>
```

If a work item cannot be completed immediately and it is required that the Process execution continues while the work item completes the execution, the Process execution can continue asynchronously and the work item manager can be notified about the work item completion later.

To abort the work item, use the **`WorkItemHandler.abortWorkItem()`** before it is completed. For more information about asynchronous execution, see Section 5.1.4.1, “Asynchronous execution” .

4.15.4. Registering Work Item handler in Business Central

To register a work item handler in Business Central, follow these steps:

Procedure: Uploading JAR File

1. Log into Business Central.
2. Click **Authoring** → **Artifact repository**.
3. Click **Upload** and select the JAR file of your work item handler.
4. Click **Upload**.

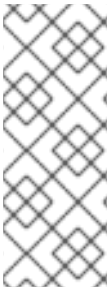
Procedure: Adding Dependencies

1. Click **Authoring** → **Project Authoring**.
2. Click **Open Project Editor**.
3. Click **Project Settings: Project General Settings** and select **Dependencies list** from the menu.
4. Click **Add from repository** and select the file you have uploaded.

Procedure: Registering Work Item Handler

1. Click **Authoring** → **Project Authoring**.
2. Click **Open Project Editor**.
3. Click **Project Settings: Project General Settings** and select **Deployment descriptor** from the menu.
4. Navigate to **Work Item handlers** and click **Add**.
5. Enter the name of your custom work item definition into the first **`Value`** field with no white spaces. For example, *GoogleCalendar*.
6. Instantiate your work item handler in the second field. For example, if your work item is in the **`com.sample`** package, **`new com.sample.GoogleCalendarHandler()`**.

7. Click **Save**.



NOTE

If you want your work item handler to be available for all your projects, place the JAR file in **DEPLOY_DIR/business-central.war/WEB-INF/lib/**.

If you want to register your work item handler for all your projects, you can do so in **{SERVER_HOME}/business-central.war/WEB-INF/classes/META-INF/kie-wb-deployment-descriptor.xml**.

4.15.5. Registering Work Item Handler Outside of Business Central

If you use **RuntimeManager** directly, use the following example:

```
RuntimeEnvironment environment =
RuntimeEnvironmentBuilder.Factory.get().newDefaultBuilder()
 .userGroupCallback(userGroupCallback)
 .addAsset(ResourceFactory.newClassPathResource("BPMN2-ScriptTask.bpmn2"),
ResourceType.BPMN2)
 .registerableItemsFactory(new DefaultRegisterableItemsFactory() {

 @Override
 public Map<String, WorkItemHandler> getWorkItemHandlers(RuntimeEngine
runtime) {
  Map<String, WorkItemHandler> handlers =
super.getWorkItemHandlers(runtime);
  handlers.put("async", new AsyncWorkItemHandler(executorService,
"org.jbpm.executor.commands.PrintOutCommand"));
  return handlers;
  }
 })
 .get();

manager =
RuntimeManagerFactory.Factory.get().newSingletonRuntimeManager(environment
);
```

- **userGroupCallback** is in the **org.jbpm.services.task.identity** package.
- **executorService** is in the **org.jbpm.executor.ExecutorServiceFactory** package.

If you want to include custom **WorkItemHandler**, you can implement the **RegisterableItemsFactory** interface. Alternatively, you can extend the following existing implementation and add your handlers:

- **org.jbpm.runtime.manager.impl.SimpleRegisterableItemsFactory**
- **org.jbpm.runtime.manager.impl.DefaultRegisterableItemsFactory**
- **org.jbpm.runtime.manager.impl.KModuleRegisterableItemsFactory**
- **org.jbpm.runtime.manager.impl.cdi.InjectableRegisterableItemsFactory**

For further information about the implementation, see the **org.jbpm.runtime.manager.impl.*** package.

NOTE

The recommended practice is to use the Service API and register your work item handlers in KJAR in **`kie-deployment-descriptor.xml`**.

4.16. SERVICE REPOSITORY

The service repository feature allows you to import an already existing work item from a repository directly into your project. It allows multiple users to reuse generic work items, such as work items allowing integration with Twitter, performing file system operations, and similar. Imported work items are automatically added to your palette and ready to use.

PUBLIC SERVICE REPOSITORY

A public service repository with various predefined work items is available at http://docs.jboss.org/jbpm/v6.4/repository/.

NOTE

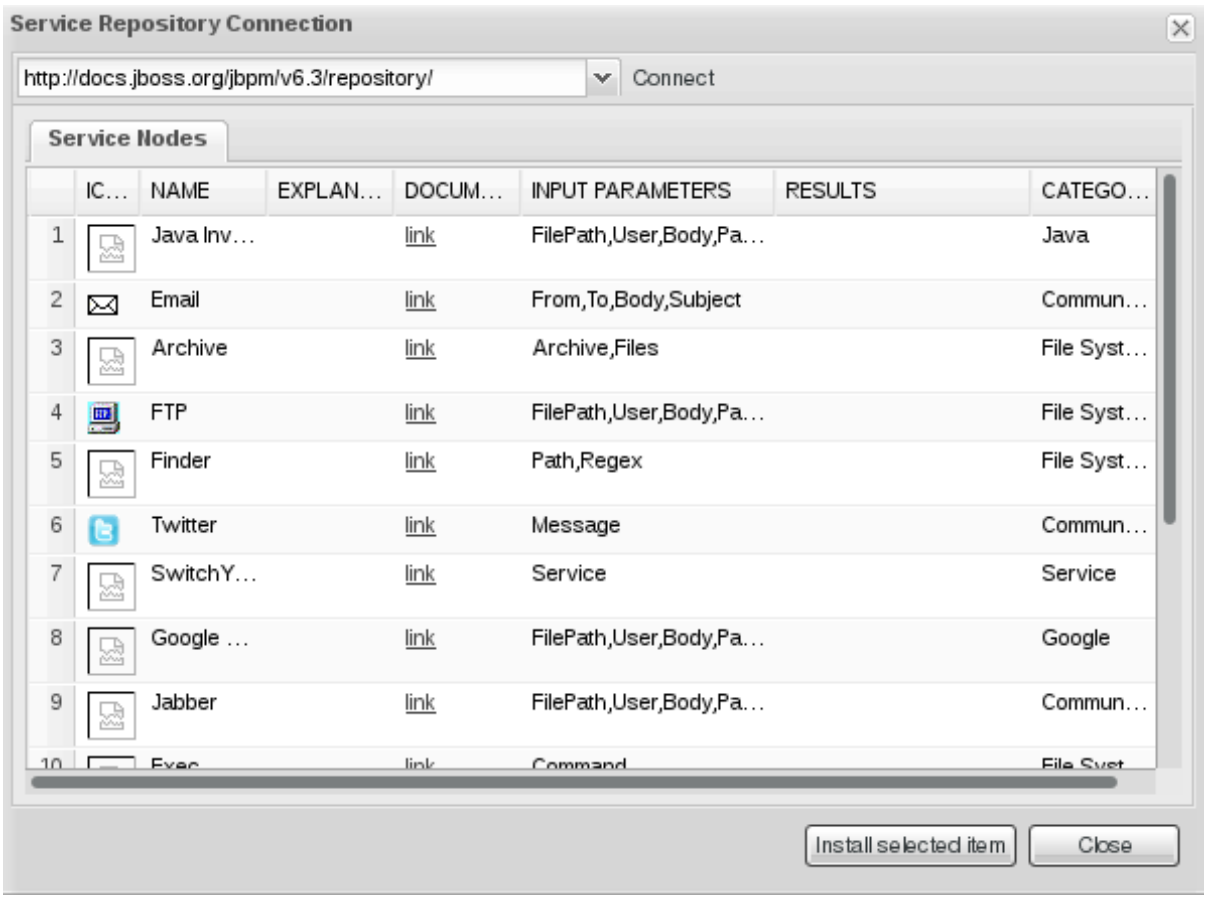
Although you can import any of these work items, only the following work items are available by default (and supported) in Red Hat JBoss BPM Suite: Log, Email, Rest, and WS. You can still import the other work items, but they are *not* supported by Red Hat.

4.16.1. Importing from Service Repository

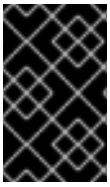
To import a work item from a service repository, do the following:

1. Open your process in the Process Designer.
2. In the editor menu, click the **Connect to a Service Repository** () button.
3. In the **Service Repository Connection** window, define the location of the repository on the location input line and click **Connect**.

Figure 4.23. Establishing Connection to Service Repository



4. Click the asset you want to import and then **Install selected item**.



WORK ITEMS MAY NOT APPEAR IN YOUR PALETTE

Every work item must be registered in the ***DEPLOY_DIRECTORY*/business-central.war/WEB-INF/classes/META-INF/CustomWorkItemHandler.conf** file. If a work item is not registered in the file, it will not be available for use.

4.16.2. Setting up Service Repository

A service repository can be any repository, local or remote, with the **index.conf** file in its root directory.

Repository Configuration File
The **index.conf** file must be located in the root directory of the service repository. It contains a list of any directory within the repository that are to be considered directories of the service repository.

Example 4.7. index.conf

```
Email
FileSystem
ESB
FTP
Google
Java
Jabber
Rest
```

```
RSS
Transform
Twitter
```

Each directory contains either another **index.conf** file so as to serve as a directory or work item resources. Note that the hierarchical structure of the repository is not shown when browsing the repository using the import wizard, as the category property in the configuration file is used for that.

Work Items and Their Resources
Directories with work items must contain:

- A *work item configuration file* is a file with the same name as the parent directory (for example **Twitter.conf**) that contains details about the work item resources in the service repository. The file is an extension of the work item definition file (see Section 4.15.1, “Work Item Definition”). Note that the configuration file must contain references to any dependencies the work item handler requires. Optionally, it can define the documentation property with a path to documentation and category which defines the category the custom work item is placed under in the repository.

  Example 4.8. Work Item Configuration File

  ```
  import
  org.drools.core.process.core.datatype.impl.type.StringDataType;
  [
    [
    "name" : "Twitter",
    "description" : "Send a twitter message",
    "parameters" : [
    "Message" : new StringDataType()
    ],
    "displayName" : "Twitter",
    "eclipse:customEditor" :
  "org.drools.eclipse.flow.common.editor.editpart.work.SampleCustomE
  ditor",
    "icon" : "twitter.gif",
    "category" : "Communication",
    "defaultHandler" :
  "org.jbpm.process.workitem.twitter.TwitterHandler",
    "documentation" : "index.html",
    //Every work item definition should specify dependencies even if
  it doesn't have one.
    "dependencies" : []
      [
     "file:./lib/jbpm-twitter.jar",
     "file:./lib/twitter4j-core-2.2.2.jar"
      ]
    ]
  ]
  ```

- All resources referenced in the work item configuration file: icon, documentation, and dependencies.

4.17. USER TASK CALLS

The User Task service exposes a Java API for managing the life cycle of its User Tasks via the **TaskClient** class. The API is intended for developers to allow direct managing of the lifecycle of User Tasks. End users are advised to use the Business Central web application for User Task management.

To manage user tasks via a public API use the methods of the **org.kie.api.task.TaskService** class. The methods of this interface take the following arguments:

- **taskId**: ID of the target Task instance usually extracted from the currently selected User Task in the user task list in the user interface
- **userId**: ID of the user that is executing the action called by the method; usually the ID of the user that is logged in

The following is a subset of methods provided by the **org.kie.api.task.TaskService** class:

```
void start(long taskId, String userId);
void stop(long taskId, String userId);
void release(long taskId, String userId);
void suspend(long taskId, String userId);
void resume(long taskId, String userId);
void skip(long taskId, String userId);
void delegate(long taskId, String userId, String targetUserId);
void complete(long taskId, String userId, Map<String, Object> results);
```

Example 4.9. Starting and completing a simple user task

```
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.manager.RuntimeEngine;
import org.kie.api.runtime.manager.RuntimeManager;
import org.kie.api.runtime.process.ProcessInstance;
import org.kie.api.task.TaskService;
import org.kie.api.task.model.TaskSummary;

....
KieSession ksession = runtimeEngine.getKieSession();
TaskService taskService = runtimeEngine.getTaskService();
ProcessInstance processInstance =
ksession.startProcess("com.sample.bpmn.hello");

// John is assigned a task and he completes it
List<TaskSummary> list =
taskService.getTasksAssignedAsPotentialOwner("john", "en-UK");
TaskSummary task = list.get(0);

logger.info("John is executing task {}", task.getName());

taskService.start(task.getId(), "john");
taskService.complete(task.getId(), "john", null);

...
```

4.18. ACTOR ASSIGNMENT CALLS

User Tasks must define either the **ActorID** or the **GroupID** parameter, which define the users who can or should execute the User Tasks. It is in the Task List of these users the Task appears.

If the User Task element defines exactly one user, the User Task appears only in the Task List of that particular user. If a User Task is assigned to more than one user, that is, to multiple actors or to a group, it appears in the Task List of all the users and any of the users can claim and execute the User Task. End users define these properties in the Process Designer.

PREDEFINED ADMINISTRATOR USER

The Administrator can manipulate the life cycle of all Tasks, even if not being their potential owner. By default, a special user with userId **Administrator** is the administrator of each Task. It is therefore recommended to always define at least user Administrator when registering the list of valid users with the User Task service.

4.18.1. Connecting to custom directory information services

It is often necessary to establish connection and transfer data from existing systems and services, such as LDAP, to acquire data on actors and groups for User Tasks. This can be done by implementing the **UserGroupInfoProducer** interface which allows you to create your own implementation for user and group management, and then configuring it over CDI for Business Central. These are the steps required to implement and make this interface active:

1. Create an implementation of the **UserGroupInfoProducer** interface and provide your own custom callback (see Section 4.19.1, “Connecting to LDAP”) and user info implementations according to the needs from the producer.
   This implementation must be annotated with the **@Selectable** qualifier for it to be found by Business Central. The listing below shows an example LDAP implementation:

```
import javax.enterprise.context.ApplicationScoped;
import javax.enterprise.inject.Alternative;
import javax.enterprise.inject.Produces;

import org.jbpm.services.task.identity.LDAPUserGroupCallbackImpl;
import org.jbpm.services.task.identity.LDAPUserInfoImpl;
import org.jbpm.shared.services.cdi.Selectable;
import org.kie.api.task.UserGroupCallback;
import org.kie.internal.task.api.UserInfo;

@ApplicationScoped
@Alternative
@Selectable
public class LDAPUserGroupInfoProducer implements
UserGroupInfoProducer {

  private UserGroupCallback callback = new
```

```
LDAPUserGroupCallbackImpl(true);
  private UserInfo userInfo = new LDAPUserInfoImpl(true);

  @Override
  @Produces
  public UserGroupCallback produceCallback() {
    return callback;
  }

  @Override
  @Produces
  public UserInfo produceUserInfo() {
    return userInfo;
  }

}
```

2. Package your custom implementations (the **LDAPUserGroupInfoProducer**, the **LDAPUserGroupCallbackImpl** and the **LDAPUserInfoImpl** classes from the example above) into a bean archive (jar with META-INF/beans.xml so it can be found by CDI container). Add this jar file to **business-central.war/WEB-INF/lib**.

3. Modify **business-central.war/WEB-INF/beans.xml** and add the implementation (**LDAPUserGroupInfoProducer** from the example above) as an alternative to be used by Business Central.

```
<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>com.test.services.producer.LDAPUserGroupInfoProducer</class>
    </alternatives>
</beans>
```

WARNING

The use of a custom **UserGroupInfoProducer** requires internal APIs, which may be broken in future releases. Using a custom **UserGroupInfoProducer** is not recommended or supported by Red Hat.

4. Restart your server and your custom callback implementation should now be used by Business Central.

4.19. LDAP CONNECTION

A dedicated **UserGroupCallback** implementation for LDAP servers is provided with the product to allow the User Task service to retrieve information on users, and groups and roles directly from an LDAP service.

The LDAP UserGroupCallback implementation takes the following properties:

- **ldap.bind.user**: username used to connect to the LDAP server (optional if LDAP server accepts anonymous access)
- **ldap.bind.pwd**: password used to connect to the LDAP server (optional if LDAP server accepts anonymous access)
- **ldap.user.ctx**: context in LDAP with user information (mandatory)
- **ldap.role.ctx**: context in LDAP with group and role information (mandatory)
- **ldap.user.roles.ctx**: context in LDAP with user group and role membership information (optional; if not specified, ldap.role.ctx is used)
- **ldap.user.filter**: filter used to search for user information; usually contains substitution keys {0}, which are replaced with parameters (mandatory)
- **ldap.role.filter**: filter used to search for group and role information, usually contains substitution keys {0}, which are replaced with parameters (mandatory)
- **ldap.user.roles.filter**: filter used to search for user group and role membership information, usually contains substitution keys {0}, which are replaced with parameters (mandatory)
- **ldap.user.attr.id**: attribute name of the user ID in LDAP (optional; if not specified, **uid** is used)
- **ldap.roles.attr.id**: attribute name of the group and role ID in LDAP (optional; if not specified **cn** is used)
- **ldap.user.id.dn**: user ID in a DN, instructs the callback to query for user DN before searching for roles (optional, by default **false**)
- **java.naming.factory.initial**: initial context factory class name (by default **com.sun.jndi.ldap.LdapCtxFactory**)
- **java.naming.security.authentication**: authentication type (possible values are **none**, **simple**, **strong**; by default **simple**)
- **java.naming.security.protocol**: security protocol to be used; for instance**ssl**
- **java.naming.provider.url**: LDAP url (by default **ldap://localhost:389**; if the protocol is set to **ssl** then **ldap://localhost:636**)

4.19.1. Connecting to LDAP

To be able to use the LDAP UserGroupCallback implementation configure the respective LDAP properties (see Section 4.19, “LDAP connection”) in one of the following ways:

- *programatically*: build a **Properties** object with the respective LDAPUserGroupCallbackImpl properties and create **LDAPUserGroupCallbackImpl** with the **Properties** object as its parameter.

```
import org.kie.api.PropertiesConfiguration;
import org.kie.api.task.UserGroupCallback;
...
Properties properties = new Properties();
properties.setProperty(LDAPUserGroupCallbackImpl.USER_CTX,
"ou=People,dc=my-domain,dc=com");
properties.setProperty(LDAPUserGroupCallbackImpl.ROLE_CTX,
"ou=Roles,dc=my-domain,dc=com");
properties.setProperty(LDAPUserGroupCallbackImpl.USER_ROLES_CTX,
"ou=Roles,dc=my-domain,dc=com");
properties.setProperty(LDAPUserGroupCallbackImpl.USER_FILTER, "(uid=
{0})");
properties.setProperty(LDAPUserGroupCallbackImpl.ROLE_FILTER, "(cn=
{0})");
properties.setProperty(LDAPUserGroupCallbackImpl.USER_ROLES_FILTER,
"(member={0})");

UserGroupCallback ldapUserGroupCallback = new
LDAPUserGroupCallbackImpl(properties);

UserGroupCallbackManager.getInstance().setCallback(ldapUserGroupCall
back);
```

- *declaratively*: create the **jbpm.usergroup.callback.properties** file in the root of your application or specify the file location as a system property: **-Djbpm.usergroup.callback.properties=FILE_LOCATION_ON_CLASSPATH** Make sure to register the LDAP callback when starting the User Task server.

```
#ldap.bind.user=
#ldap.bind.pwd=
ldap.user.ctx=ou\=People,dc\=my-domain,dc\=com
ldap.role.ctx=ou\=Roles,dc\=my-domain,dc\=com
ldap.user.roles.ctx=ou\=Roles,dc\=my-domain,dc\=com
ldap.user.filter=(uid\={0})
ldap.role.filter=(cn\={0})
ldap.user.roles.filter=(member\={0})
#ldap.user.attr.id=
#ldap.roles.attr.id=
```

4.20. EXCEPTION MANAGEMENT

When an unexpected event, that deviates from the normative behavior, occurs in a Process instance, it is referred to as an exception. There are two types of exceptions: business exceptions and technical exceptions.

Business exceptions

Business exceptions relate to the possible incorrect scenarios of the particular Process, for example, trying to debit an empty bank account. Handling of such exceptions is designed directly in the Process model using BPMN Process elements.

When modeling business exception management, the following mechanisms are to be used:

Errors

An Error is a signal that an unexpected situation occurred (see Section A.4.1, “Errors”). The mechanism can be used immediately when the problem arises and does not allow for any compensation.

Compensation

Compensation is equivalent to the Error mechanism; however, it can be used only on Sub-Processes when it is required that the execution flow continues after the compensation using the "regular" outgoing Flow (execution continues after the compensation as if no compensation occurred).

Canceling

Canceling is equivalent to the Error mechanism; however, it can be used only on Sub-Processes and it is required that the Sub-Process takes the flow leaving the respective Cancel Intermediate Event so that the "normal" execution flow is never taken as opposed to compensation.

Technical exceptions

Technical exceptions happen when a technical component of a business process acts in an unexpected way. When using Java-based systems, this often results in a Java Exception being thrown by the system. Technical components used in a Process fail in a way that can not be described using BPMN (for further information, see Section 5.1.5, “Technical exceptions”).

CHAPTER 5. ADVANCED PROCESS MODELING

5.1. PROCESS MODELING OPTIONS

You can create processes in multiple ways:

Using one of the graphical editors

You can use two delivered graphical editors. Process Designer is available through Business Central and Eclipse Process Designer. See *Red Hat JBoss BRMS User Guide*for more information on how to use the editors.

Using an XML editor

You can use any XML or text editor to create a process specification using the BPMN2 XML schema.

Using the Process API

You can use the Red Hat JBoss BPM Suite **core** API directly. The most important process model elements are defined in the packages org.jbpm.workflow.core and org.jbpm.workflow.core.node.

5.1.1. Process modeling using XML

The BPMN2 file must meet the BPMN2 schema. The file content comprises the following parts:

XML prolog

The XML prolog consists of the XML declaration and DTD declaration.

The process element

The **process** element defines process attributes and contains definitions of the process elements (nodes and connections).

BPMN diagram definition

The **BPMNDiagram** element contains definitions for visualization of the Process elements in the Process Diagram.

Example 5.1. BPMN2 example file

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions id="Definition"
             targetNamespace="http://www.jboss.org/drools"
             typeLanguage="http://www.java.com/javaTypes"
             expressionLanguage="http://www.mvel.org/2.0"
             xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"Rule
Task
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"

xsi:schemaLocation="http://www.omg.org/spec/BPMN/20100524/MODEL
BPMN20.xsd"
             xmlns:g="http://www.jboss.org/drools/flow/gpd"
             xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
             xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
             xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
             xmlns:tns="http://www.jboss.org/drools">
  <process processType="Private" isExecutable="true"
```

```
id="com.sample.hello" name="Hello Process" >

    <!-- nodes -->
    <startEvent id="_1" name="Start" />
    <scriptTask id="_2" name="Hello" >
      <script>System.out.println("Hello World");</script>
    </scriptTask>
    <endEvent id="_3" name="End" >
        <terminateEventDefinition/>
    </endEvent>

    <!-- connections -->
    <sequenceFlow id="_1-_2" sourceRef="_1" targetRef="_2" />
    <sequenceFlow id="_2-_3" sourceRef="_2" targetRef="_3" />
  </process>

  <bpmndi:BPMNDiagram>
    <bpmndi:BPMNPlane bpmnElement="com.sample.hello" >
      <bpmndi:BPMNShape bpmnElement="_1" >
        <dc:Bounds x="16" y="16" width="48" height="48" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape bpmnElement="_2" >
        <dc:Bounds x="96" y="16" width="80" height="48" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape bpmnElement="_3" >
        <dc:Bounds x="208" y="16" width="48" height="48" />
      </bpmndi:BPMNShape>
      <bpmndi:BPMNEdge bpmnElement="_1-_2" >
        <di:waypoint x="40" y="40" />
        <di:waypoint x="136" y="40" />
      </bpmndi:BPMNEdge>
      <bpmndi:BPMNEdge bpmnElement="_2-_3" >
        <di:waypoint x="136" y="40" />
        <di:waypoint x="232" y="40" />
      </bpmndi:BPMNEdge>
    </bpmndi:BPMNPlane>
  </bpmndi:BPMNDiagram>
</definitions>
```

5.1.2. Process modeling using API

To be able to execute processes from within your application, you need to perform the following in your code:

1. Create a Runtime Manager in your Execution Server.
    - Singleton: allows sequential execution of multiple instances in the one session
    - PerProcessInstance: allows you to create multiple process instances; every instance is created within its own session.
    - PerRequestSession: every external interaction with a process instances causes that the process session finishes and the process instance is re-created in a new session.

2. Get a runtime context and create a session in it.
3. Start a Process from the underlying Knowledge Base.
4. Close the Runtime Manager.

Example 5.2. Process instantiation in a session of Per Process Instance Runtime Manager

```
import org.kie.api.runtime.manager.RuntimeManager;
import org.kie.api.runtime.manager.RuntimeManagerFactory.Factory;
import org.kie.api.runtime.manager.RuntimeEngine;
import org.kie.api.runtime.KieSession;
...
RuntimeManager manager =
    RuntimeManagerFactory.Factory.get()
        .newPerProcessInstanceRuntimeManager(environment);

RuntimeEngine runtime =
    manager.getRuntimeEngine(
        ProcessInstanceIdContext.get());

KieSession ksession = runtime.getKieSession();
// do something here, e.g.
ksession.startProcess("org.jbpm.hello");

manager.disposeRuntimeEngine(engine);
manager.close();
```

5.1.3. Process update

5.1.3.1. Process update

When updating a Process definition, the new Process definition must define an increased version number and an update policy: the update policy defines how to handle the running Process instances of the older Process definition. You can decide to apply on them one of the following strategies:

- **`Abort`**: any running Process instances are aborted. If necessary, you can have the Process instance restarted using the new Process definition.
- **`Transfer`**: any running Process instances are migrated to the new process definition: once the instance has been migrated successfully, it will continue its execution based on the updated process logic.

Note that the older version of the Process definition remains in the repository as well as in the respective sessions. Therefore, the new process should have a different ID, though the name can remain the same, and you can use the version parameter to show when a Process is updated (the version parameter is just a String and is not validated).

Example 5.3. Process abort update

```
import org.kie.api.KieBase;
```

```
import org.kie.api.KieServices;
import org.kie.api.runtime.KieSessionConfiguration;

// build kbase with the replace-version-1.bpmn process
        KieBase kbase =
KieServices.Factory.get().newKieSessionConfiguration();
        kbase.addKnowledgePackages(getProcessPackages("replace-version-
1.bpmn"));

        KieSession ksession = kbase.newStatefulKnowledgeSession();
        try {
            // start a replace-version-1.bpmn process instance
            ksession.startProcess("com.sample.process", Collections.
<String, Object>singletonMap("name", "process1"));

            // add the replace-version-2.bpmn process and start its
instance
            kbase.addKnowledgePackages(getProcessPackages("replace-
version-2.bpmn"));
            ksession.startProcess("com.sample.process", Collections.
<String, Object>singletonMap("name", "process2"));

            // signal all processes in the session to continue (both
instances finish)
            ksession.signalEvent("continue", null);
        } finally {
            ksession.dispose();
        }
```

5.1.4. Multi-threading

Technical multi-threading is what happens when multiple threads or processes are started on a computer, for example by a Java or C program. Logical multi-threading is what we see in a BPM process after the process reaches a parallel gateway, for example. From a functional standpoint, the original process splits in two processes that are executed in a parallel fashion.

The Process engine supports logical multi-threading; for example, in Processes that include a parallel Gateway. The logical multi-threading is implemented using one technical thread: A Process that includes logical multi-threading is executed in one technical thread. Avoiding technical multi-threading prevents further implementation complexity as multiple technical threads of one Process instance need to communicate their state information to each other. While it might seem that technical multi-threading would bring significant performance benefits, the extra logic needed to make sure the threads can work together well may cancel out these benefits.

In general, the execution engine also executes actions in serial. For example, when the engine encounters a Script Task, it synchronously executes the script and waits for it to complete before continuing execution. Similarly, when a Process encounters a Parallel Gateway, it sequentially triggers each of the outgoing Flows. This is possible since execution is almost always instantaneous. Similarly, action scripts in a Process are also executed synchronously, and the engine waits for them to finish before continuing

execution. That means, that if the execution needs to wait, for example, a **Thread.sleep()** call is being execution, the engine does not continue any execution — it remains blocked during the wait period.

5.1.4.1. Asynchronous execution

If a work item execution of a Task does not execute instantaneously, but needs to wait, for example, to receive a response from an external system, the service handler must handle your service asynchronously. The asynchronous handler only invokes the service and notifies the engine once the results are available. In the mean time, the process engine continues the execution of the process.

A typical example of a service that requires asynchronous invocation is a Human Task. The engine is not to wait until a human actor responds to the request but continue and process the result of the Task when it becomes available. The human task handler creates a new task on the task list of the assigned actor and the engine is then allowed to continue the execution: The handler notifies the engine asynchronously when the user completes the task.

To implement an asynchronous service handler, implement the actual service in a new thread using the **executeWorkItem()** method in the work item handler that allows the Process instance to continue its execution.

Example 5.4. Example of asynchronous service handling in Java

```
import org.kie.api.runtime.process.WorkItem;
import org.kie.api.runtime.process.WorkItemHandler;
import org.kie.api.runtime.process.WorkItemManager;

public class MyServiceTaskHandler implements WorkItemHandler {



  public void executeWorkItem(WorkItem workItem, WorkItemManager
manager) {

    new Thread(new Runnable() {

      public void run() {

        // The main thread with the parent element execution
        }
    }).start();
  }

  public void abortWorkItem(WorkItem workItem, WorkItemManager manager)
{

  }
}
```

It is recommended to have your handler contact a service that executes the business operation, instead of performing the task, as failure of the business operation will not affect your process. This approach also provides greater flexibility when developing and reusing

services.

For example, your human task handler can invoke the human task service to add a task to the service. To implement an asynchronous handler, you usually have to do an asynchronous invocation of the handler. This usually depends on the technology you use to do the communication and might be an asynchronous invocation of a web service, or sending a JMS message to the external service.

The Red Hat JBoss BPM Suite Job Executor
Red Hat JBoss BPM Suite, from 6.1 version, implements a special component, the Job Executor, to accommodate asynchronous execution. The Executor provides an environment independent from the process runtime environment for instantiation of asynchronous executions defined in a **`Command`** object.

In particular, the JBoss BPM Suite Job Executor provides advanced handling of common asynchronous execution operations, like error handling, retry, cancellation and history logging. You can continue to delegate work to a separate thread, as described above, in a custom implementation of **`WorkItemHandler`** and use the JBoss BPM Suite Job Executor to handle these operations for you.

The JBoss BPM Suite Job Executor works on instances of the **`Command`** interface. This interface implements a single method **`execute()`** that accepts the **`CommandContext`** data transfer object with serializable data. The **`Command`** instance should only contain the business logic to be executed and have no reference to the underlying process engine or runtime related data. At the minimum, the **`CommandContext`** should provide the **`businessKey`** - the unique identifier of the caller and **`callback`** - the fully qualified classname (FQCN) of the **`CommandCallback`** instance to be called on command completion. Of course, when executed as part of a process (**`WorkItemHandler`**), you will need to provide additional data like the **`workItem`**, **`processInstanceId`** and **`deploymentId`**.

The result for the execution is returned via an instance of the **`ExecutionResults`** class. As with the input data, the data provided by this class must also be serializable.

Business Central provides a way for you to view the current jobs within the system; to schedule them, cancel them, or view their history. Go to **Deploy → Jobs** to access this screen. You can even create a new job by clicking on the **New Job** button.

The Asynchronous WorkItemHandler
Red Hat JBoss BPM Suite provides an out of the box asynchronous **`WorkItemHandler`** that is backed by the JBoss BPM Suite Job Executor. All the features that the JBoss BPM Suite Executor delivers are available for background execution within a process instance. There are two ways to configure this **`AsyncWorkItemHandler`** class:

1. As a generic handler where you provide the command name as part of the work item parameters. In Business Central while modeling a process, if you need to execute some work item asynchronously: specify **`async`** as the value for the **`TaskName`** property, create a data input called **`CommandClass`** and assign the FQCN of this **`CommandClass`** as the data input.

2. As a specific handler which is created to handle a given type of work item, thus allowing you to register different instances of **`AsyncWorkItemHandler`** for different work items. Commands are most likely to be dedicated to a particular work item, which allows you to specify the **`CommandClass`** at registration time instead of requiring it at design time, as with the first approach. But this means that an additional CDI bean that implements **`WorkItemHandlerProducer`** interface needs

to be provided and placed on the application classpath so that the CDI container can find it. When you are ready to model your process, set the value of the **TaskName** property to the one provided at registration time.

Customizing Retry Interval
The Red Hat JBoss BPM Suite Job Executor allows you to specify the number of retries in case of errors while running a job. In addition to that, it enables you to configure retry delay on each job. You can customize the retry interval depending on the time required for you to correct the problems that caused the job to fail.

To customize the retry interval, you can either set the **retryDelay** parameter in the **CommandContext** or the **RetryDelay** parameter of asynchronous work item.

You can set the retry delay as:

- A single time expression. For example, 5m, 2h.
- A comma separated list of time expressions that must be used for subsequent retries. For example, 10s,10m,1h, and 1d.

If you are providing a comma separated list of time expressions and if the number of retry delays is smaller than number of retries, the executor will use the last available value from the list. In case you provide a single time expression for retry delay, the retries will be equally spaced.

Configuring the JBoss BPM Suite Executor
The JBoss BPM Suite executor can be configured to allow fine tuning of its environment via the following system properties:

1. **org.kie.executor.disabled**: true OR false - enable/disable the executor.
2. **org.kie.executor.pool.size**: Integer. Specify the thread pool size for the executor, which by default is 1.
3. **org.kie.executor.retry.count**: Integer. Specifies the default number of retries in case of an error executing a job. The default value is 3.
4. **org.kie.executor.interval**: Integer. Specifies the time to wait between checking for waiting jobs. The default value is 3 seconds.
5. **org.kie.executor.timeunit**: NANOSECONDS OR MICROSECONDS OR MILLISECONDS OR SECONDS OR MINUTES OR HOURS OR DAYS. Specifies the unit for the interval property. The default is SECONDS.

NOTE

The JBoss BPM Suite Job Executor can also be used as an embedded service in customer application as it has been moved to the public kie-api. See *Red Hat JBoss BPMS Development Guide* for more information on how to use Job Executor in embedded mode.

5.1.4.2. Multiple Sessions and persistence

The simplest way to run multiple Process instances is to run them in one knowledge session. However, it is possible to run multiple Process instances in different knowledge sessions or in different technical threads.

When using multiple knowledge session with multiple processes and adding persistence, use a database that allows row-level as well as table-level locks: There could be a situation when there are 2 or more threads running, each within its own knowledge session instance. On each thread, a Process is being started using the local knowledge session instance. In this use case, a race condition exists in which both thread A and thread B have coincidentally simultaneously finished a Process instance. At this point, both thread A and B are committing changes to the database. If row-level locks are not possible, then the following situation can occur:

- Thread A has a lock on the ProcessInstanceInfo table, having just committed a change to that table.
- Thread A wants a lock on the SessionInfo table in order to commit a change.
- Thread B has the opposite situation: It has a lock on the SessionInfo table, having just committed a change.
- Thread B wants a lock on the ProcessInstanceInfo table, even though Thread A already has a lock on it.

This is a deadlock situation which the database and application are not be able to solve, unless row-level locks are possible and enabled in the database and tables used.

5.1.4.3. Asynchronous Events

In cases where several process instances from different process definitions are waiting for the same signal, they are generally executed sequentially in the same single thread. However, if one of those process instances throws a runtime exception, all the other process instances are affected, usually resulting in a rolled back transaction. To avoid this, Red Hat JBoss BPM Suite supports using asynchronous signals events for:

- Throwing Intermediate Signal Events
- End Events

From the Business Central, set the **Data Input** value of the throw event to async to automatically set the Executor Service on each ksession. This ensures that each process instance is signaled in a different transaction.

5.1.5. Technical exceptions

Technical exceptions occur when a technical component of a Process acts in an unexpected way. When using Java-based systems, this often results in a Java Exception. As these exceptions cannot be handled using BPMN2, it is important to handle them in expected ways.

The following types of code might throw exceptions:

- Code present directly in the process definition
- Code that is not part of the product executed during a Process
- Code that interacts with a technical component outside of the Process Engine

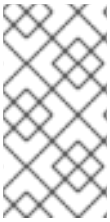
This includes the following:

- Code in Element properties, such as the **Script** property of a *Script Task* element or in the definitions of the interception actions, that is, the **onEntry** and **onExit** properties
- Code in **WorkItemHandlers** associated with **task** and task-type nodes

Code in Element properties
Exceptions thrown by code defined in Element properties can cause the Process instance to fail in an unrecoverable way. Often, it is the code that starts the Process that will end up throwing the exception generated by a Process without returning a reference to the Process instance. Such code includes for example the **onEntry** and **onExit** properties, Script defined for the Script Task, etc.

Therefore, it is important to limit the scope of the code in these Elements so that is operates only over Process variables. Using a **scriptTask** to interact with a different technical component, such as a database or web service has *significant risks* because any exceptions thrown will corrupt or abort the Process instance.

To interact with other systems, use **task** Elements, **serviceTask** Elements and other **task**-type Elements. Do not use the **scriptTask** nodes for these purposes.



NOTE

If the script defined in a **scriptTask** causes the problem, the Process Engine usually throws the **WorkflowRuntimeException** with information on the Process (see Section 5.1.5.1.5, “Extracting information from WorkflowRuntimeException”).

Code in WorkItemHandlers
WorkItemHandlers are used when your Process interacts with other technical systems (for more information on WorkItemHandlers see Section 4.15.1, “Work Item Definition”).

You can either build exception handling into your own WorkItemhandler implementations or wrap your implementation into the **handler decorator** classes (for examples and detailed information see Section 5.1.5.1.2, “Exception handling classes”). These classes include the logic that is executed when an exception is thrown during the execution or abortion of a work item:

SignallingTaskHandlerDecorator

catches the exception and signals it to the Process instance using a configurable event type when the **executeWorkItem()** or **abortWorkItem** methods of the original **WorkItemHandler** instance throw an exception. The exception thrown is passed as part of the event. This functionality can be also used to signal to an Event SubProcess defined in the Process definition.

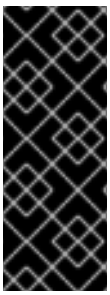
LoggingTaskHandlerDecorator

logs error about any exceptions thrown by the **executeWorkItem()** and **abortWorkItem()** methods. It also saves any exceptions thrown to an internal list so that they can be retrieved later for inspection or further logging. The content and format of the message logged are configurable.

While the classes described above covers most cases involving exception handling as it catches any throwable objects, you might still want to write a custom WorkItemHandler that includes exception handling logic. In such a case, consider the following:

- Does the implementation catch all exceptions the code could return?

- Does the implementation complete or abort the work item after an exception has been caught or uses a mechanisms to retry the process later (in some cases, incomplete process instances might be acceptable)?
- Does the implementation define any other actions that need to be taken when an exception is caught? Would it be beneficial to interact with other technical systems? Should a Sub-Process be triggered to handle the exception?



IMPORTANT

If WorkItemManager to signals that the work item has been completed or aborted, make sure the signal is sent after any signals to the Process instance were sent. Depending on how your Process definition, calling WorkItemManager.completeWorkItem() or WorkItemManager.abortWorkItem() triggers the completion of the Process instance as these methods trigger further execution of the Process execution flow.

5.1.5.1. Technical exception examples

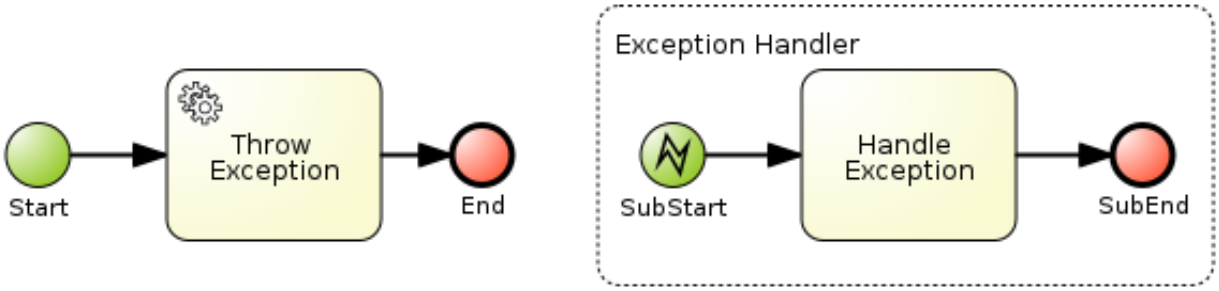
5.1.5.1.1. Service Task handlers

The example involves a Throwing Error Intermediate Event caught by an Error Event Sub-Process.

When the Throwing Error Intermediate Event throws the Error, the Process instance is interrupted:

1. Execution of the Process instance stops: no other parts of the Process are executed.
2. The Process instance finishes as ABORTED.

Figure 5.1. Process with an exception handling Event Sub-Process



Parts of the BPMN2 definition of the example Process relevant for exception handling

```
<itemDefinition id="_stringItem" structureRef="java.lang.String"/> 1
<message id="_message" itemRef="_stringItem"/> 2

<interface id="_serviceInterface"
name="org.jbpm.examples.exceptions.service.ExceptionService">
<operation id="_serviceOperation" name="throwException">
<inMessageRef>_message</inMessageRef> 3
</operation>
```

```
</interface>

<error id="_exception" errorCode="code" structureRef="_exceptionItem"/> 4

<itemDefinition id="_exceptionItem"
structureRef="org.kie.api.runtime.process.WorkItem"/> 5
<message id="_exceptionMessage" itemRef="_exceptionItem"/> 6

<interface id="_handlingServiceInterface"
name="org.jbpm.examples.exceptions.service.ExceptionService">
<operation id="_handlingServiceOperation" name="handleException">
<inMessageRef>_exceptionMessage</inMessageRef> 7
</operation>
</interface>

<process id="ProcessWithExceptionHandlingError" name="Service Process"
isExecutable="true" processType="Private">
<!-- properties -->
<property id="serviceInputItem" itemSubjectRef="_stringItem"/> 8
<property id="exceptionInputItem" itemSubjectRef="_exceptionItem"/> 9

<!-- main process -->
<startEvent id="_1" name="Start" />
<serviceTask id="_2" name="Throw Exception" implementation="Other"
operationRef="_serviceOperation">

<!-- rest of the serviceTask element and process definition... -->

<subProcess id="_X" name="Exception Handler" triggeredByEvent="true" >
<startEvent id="_X-1" name="subStart">
<dataOutput id="_X-1_Output" name="event"/>
<dataOutputAssociation>
<sourceRef>_X-1_Output</sourceRef>
<targetRef>exceptionInputItem</targetRef> 10
</dataOutputAssociation>
<errorEventDefinition id="_X-1_ED_1" errorRef="_exception" /> 11
</startEvent>

<!-- rest of the subprocess definition... -->

</subProcess>

</process>
```

1 8 The **itemDefinition** element defines a data structure used in the **serviceInputItem** property of the Process.

2 3 The **message** element (1st reference) defines a message that contains the String defined by the **itemDefinition** element on the line above. The **interface** element below then refers to the **itemDefinition** element (2nd reference) in order to define what type of content the service (defined by the **interface**) expects.

4 11 The **error** element (1st reference) defines an error that is used to trigger the Event SubProcess of the Process. The content of the error is defined by the **itemDefintion** element defined below the **error** element.

5 6 7 9 10 This **itemDefinition** element (1st reference) defines an item that contains a **WorkItem** instance. The **message** element (2nd reference) then defines a message that uses this item definition to define its content. The **interface** element below that refers to the **message** definition (3rd reference) in order to define the type of content that the service expects. In the Process element itself, a **property** element (4th reference) that contains the initial **itemDefinition**. This allows the Event SubProcess to store the error it receives in that property (5th reference).

5.1.5.1.2. Exception handling classes

The **serviceTask** tasks use the org.jbpm.bpmn2.handler.ServiceTaskHandler class as its task handler class unless the **serviceTask** defines a custom **WorkItemHandler** implementation.

To catch and handle any technical exceptions a WorkItemHandler of a task might throw, wrap or decorate the handler class with a **SignallingTaskHandlerDecorator** instance.

RULES FOR SENDING SIGNALS

When sending a signal of an event to the Process Engine, consider the rules for signaling process events:

- Error events are signaled by sending an **Error-*ERRORCODE ATTRIBUTE VALUE*** value to the session.
- Signal events are signaled by sending the name of the signal to the session.
- If you wanted to send an error event to a Boundary Catch Error Event, the error type should be of the format: **"Error-" + $AttachedNodeID + "-" + $ERROR_CODE**. For example, **Error-SubProcess_1-888** would be a valid error type.
  However, this is *NOT* a recommended practice because sending the signal this way bypasses parts of the boundary error event functionality and it relies on internal implementation details that might be changed in the future. For a way to programmatically trigger a boundary error event when an Exception is thrown in **WorkItemHandler** see this KnowledgeBase article.

Example 5.5. Using SignallingTaskHandlerDecorator

The **ServiceTaskHandler** calls the **ExceptionService.throwException()** method to throw an exception (refer to the **_handlingServiceInterface** interface element in the BPMN2).

The **SignallingTaskHandlerDecorator** that wraps the **ServiceTaskHandler** sends to the Process instance the **error** with the set *error code*.

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

import org.jbpm.bpmn2.handler.ServiceTaskHandler;
import org.jbpm.bpmn2.handler.SignallingTaskHandlerDecorator;
import org.jbpm.examples.exceptions.service.ExceptionService;
```

```
import org.kie.api.KieBase;
import org.kie.api.io.ResourceType;
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.process.ProcessInstance;
import org.kie.internal.builder.KnowledgeBuilder;
import org.kie.internal.builder.KnowledgeBuilderFactory;
import org.kie.internal.io.ResourceFactory;

public class ExceptionHandlingErrorExample {

public static final void main(String[] args) {
runExample();
}

public static ProcessInstance runExample() {
KieSession ksession = createKieSession();

String eventType = "Error-code"; (1)
SignallingTaskHandlerDecorator signallingTaskWrapper (2)
= new SignallingTaskHandlerDecorator(ServiceTaskHandler.class,
eventType);
signallingTaskWrapper.setWorkItemExceptionParameterName(ExceptionService
.exceptionParameterName); (3)
ksession.getWorkItemManager().registerWorkItemHandler("Service Task",
signallingTaskWrapper);

Map<String, Object> params = new HashMap<String, Object>();
params.put("serviceInputItem", "Input to Original Service");
ProcessInstance processInstance =
ksession.startProcess("ProcessWithExceptionHandlingError", params);
return processInstance;
}

private static KieSession createKieSession() {
KnowledgeBuilder kbuilder =
KnowledgeBuilderFactory.newKnowledgeBuilder();
kbuilder.add(ResourceFactory.newClassPathResource("exceptions/ExceptionH
andlingWithError.bpmn2"), ResourceType.BPMN2);
KieBase kbase = kbuilder.newKnowledgeBase();
return kbase.newKieSession();
}
```

(1) Definition of the **Error-code** event to be sent to the process instance when the wrapped **WorkItemHandler** implementation throws an exception.

(2) Construction of the **SignallingTaskHandlerDecorator** class instance with the **WorkItemHandler** implementation and **eventType** as parameters: Note that a **SignallingTaskHandlerDecorator** class constructor that takes an instance of a **WorkItemHandler** implementation as its parameter is also available. This constructor is useful if the **WorkItemHandler** implementation does not allow a no-argument constructor.

(3) Registering the **WorkItemHandler** with the session: When an exception is thrown by the wrapped **WorkItemHandler**, the **SignallingTaskHandlerDecorator** saves it as a parameter in the **WorkItem** instance with a parameter name configured in the **SignallingTaskHandlerDecorator** (see the code below for the

ExceptionService).

5.1.5.1.3. Exception service

In Section 5.1.5.1.1, “Service Task handlers”, the BPMN2 process definition defines the exception service using the **ExceptionService** class as follows:

```
<interface id="_handlingServiceInterface"
name="org.jbpm.examples.exceptions.service.ExceptionService">
<operation id="_handlingServiceOperation" name="handleException">
```

The exception service uses the **ExceptionService** class to provide the exception handling abilities. The class is implemented as follows:

```
import org.kie.api.runtime.process.WorkItem;
...
public class ExceptionService {

  public static String exceptionParameterName =
"my.exception.parameter.name";
  public void handleException(WorkItem workItem) {
    System.out.println( "Handling exception caused by work item '" +
workItem.getName() + "' (id: " + workItem.getId() + ")");
    Map<String, Object> params = workItem.getParameters();
    Throwable throwable = (Throwable) params.get(exceptionParameterName);
    throwable.printStackTrace();
  }
  public String throwException(String message) {
    throw new RuntimeException("Service failed with input: " + message );
  }
  public static void setExceptionParameterName(String exceptionParam) {
    exceptionParameterName = exceptionParam;
  }

}
```

You can specify any Java class with the default or another no-argument constructor as the class to provide the exception service so that it is executed as part of a **serviceTask**.

5.1.5.1.4. Handling errors with Signals

In the example in Section 5.1.5.1.1, “Service Task handlers”, an *Error event* occurs during Process execution and the execution is interrupted immediately: no other Flows or Activities are executed.

However, you might want to complete the execution. In such case you can use a *Signal event* as the Process execution continues after the Signal is processed (that is, after the *Signal Event SubProcess* or another Activities that the Signal triggered, finish their execution). Also, the Process execution finished successfully, *not* in an aborted state, which is the case if an Error is used.

In the example process, we define the **error** element which is then used to throw the Error:

```
<error id="_exception" errorCode="code" structureRef="_exceptionItem"/>
```

To use a Signal instead, do the following:

1. Remove the line defining the **error** element and define a **<signal>** element:

   ```
   <signal id="exception-signal" structureRef="_exceptionItem"/>
   ```

2. Make sure to change all references from the "_exception`" **<error>** to the "exception-signal" **<signal>**.
   Change the **<errorEventDefinition>** element in the **<startEvent>**,

   ```
   <errorEventDefinition id="_X-1_ED_1" errorRef="_exception" />
   ```

   to a **<signalEventDefinition>**:

   ```
   <signalEventDefinition id="_X-1_ED_1" signalRef="exception-
   signal"/>
   ```

5.1.5.1.5. Extracting information from WorkflowRuntimeException

If a scripts in your Process definition may throw or threw an exception, you need to retrieve more information about the exception and related information.

If it is a **scriptTask** element that causes an exception, you can extract the information from the **WorkflowRuntimeException** as it is the wrapper of the scriptTask.

The **WorkflowRuntimeException** instance stores the information outlined in Table 5.1, "Information in WorkflowRuntimeException instances". Values of all fields listed can be obtained using the standard **get*** methods.

Table 5.1. Information in WorkflowRuntimeException instances

Field name	Type	Description
processInstanceId	**long**	The id of the **ProcessInstance** instance in which the exception occurred Note that the **ProcessInstance** may not exist anymore or be available in the database if using persistence.
processId	**String**	The id of the process definition that was used to start the process (that is, "ExceptionScriptTask" in `ksession.startProcess("ExceptionScriptTask");`)
nodeId	**long**	The value of the (BPMN2) id attribute of the node that threw the exception

Field name	Type	Description
nodeName	**String**	The value of the (BPMN2) name attribute of the node that threw the exception
variables	**Map<String, Object>**	The map containing the variables in the process instance (*experimental*)
message	**String**	The short message with information on the exception
cause	**Throwable**	The original exception that was thrown

The following code illustrates how to extract extra information from a process instance that throws a **WorkflowRuntimeException** exception instance.

```
import org.jbpm.workflow.instance.WorkflowRuntimeException;
import org.kie.api.KieBase;
import org.kie.api.io.ResourceType;
import org.kie.api.runtime.KieSession;
import org.kie.api.runtime.process.ProcessInstance;
import org.kie.internal.builder.KnowledgeBuilder;
import org.kie.internal.builder.KnowledgeBuilderFactory;
import org.kie.internal.io.ResourceFactory;

public class ScriptTaskExceptionExample {

public static final void main(String[] args) {
runExample();
}

public static void runExample() {
KieSession ksession = createKieSession();
Map<String, Object> params = new HashMap<String, Object>();
String varName = "var1";
params.put( varName , "valueOne" );
try {
ProcessInstance processInstance =
ksession.startProcess("ExceptionScriptTask", params);
} catch( WorkflowRuntimeException wfre ) {
String msg = "An exception happened in "
+ "process instance [" + wfre.getProcessInstanceId()
+ "] of process [" + wfre.getProcessId()
+ "] in node [id: " + wfre.getNodeId()
+ ", name: " + wfre.getNodeName()
+ "] and variable " + varName + " had the value [" +
wfre.getVariables().get(varName)
+ "]";
System.out.println(msg);
}
}
private static KieSession createKieSession() {
KnowledgeBuilder kbuilder = KnowledgeBuilderFactory.newKnowledgeBuilder();
```

```
kbuilder.add(ResourceFactory.newClassPathResource("exceptions/ScriptTaskEx
ception.bpmn2"), ResourceType.BPMN2);
KieBase kbase = kbuilder.newKnowledgeBase();
return kbase.newKieSession();
}
}
```

5.2. WORKFLOW PATTERNS

Workflow patterns are predefined blocks of Process elements that allow you to reuse once defined combination of Process elements: they include multiple nodes that are connected and form a common executable pattern that can be reused in a Process model.

Workflow patterns are available in the Shape Repository stencil set and can be drag-and-dropped on the canvas just like any other elements. To attach a pattern to an element on the canvas, select the element and then drag-and-drop the pattern from the palette onto the canvas. The pattern will be automatically connected to the element.

Multiple predefined workflow patterns are provided by default and you can define your own workflow patters as necessary. The definitions are defined as JSON objects in the **$JBOSS_HOME/standalone/deployments/business-central.war/org.kie.workbench.KIEWebapp/defaults/patterns.json** file.

5.2.1. Defining workflow patterns

To define custom workflow patterns, do the following:

1. In the stencil set of the Process Designer, locate the workflow pattern that resembles most to and that will use as base for your workflow pattern.
2. Open the **$JBOSSHOME/standalone/deployments/business-central.war/org.kie.workbench.KIEWebapp/defaults/patterns.json** file in a text editor.
3. Locate the JSON object with the description property set to the base workflow pattern name (for example, **"description" : "Sequence Pattern"**).
4. Copy the JSON object and modify its elements as needed. Note that all the JSON objects are nested in a pair of square brackets and are comma separated.

CHAPTER 6. SOCIAL EVENTS

In Red Hat JBoss BPM Suite, users can follow other users and gain an insight into what activities are being performed by those users. They can also listen for and follow timelines of regular events. This capability comes via the implementation of a Social Activities framework. This framework ensures that event notifications are generated by different activities within the system and that these notifications are broadcast for registered actors to view.

Multiple activities trigger events. These include: new repository creation, adding and updating resources and adding and updating processes. With the right credentials, a user can view these notifications once they are logged into Business Central.

Follow User

To follow a user, search for the user by entering his name in the search box in the **People** perspective. You get to this perspective by navigating to it from **Home** → **People**.

You must know the login name of the other user that you want to follow. As you enter the name in the search box, the system will try and auto-complete the name for you and display matches based on your partial entry. Select the user that you want to follow from these matches and the perspective will update to display more details about this user.

You can choose to follow the user by clicking on the **Follow** button. The perspective refreshes to showcase the user details and their recent activities.

Activity Timeline

Click on **Home** → **Timeline** to see a list of recent assets that have been modified (in the left hand window) and a list of changes made in the selected repository in the right hand side. You can click on the assets to directly open the editor for the assets (if you have the right permissions).

PART II. SIMULATION AND TESTING

CHAPTER 7. PROCESS SIMULATION

Process simulation allows users to simulate a business process based on the simulation parameters and get a statistical analysis of the process models over time in form of graphs. This helps to optimize pre and post execution of a process, minimizing the risk of change in business processes, performance forecast, and promote improvements in performance, quality and resource utilization of a process.

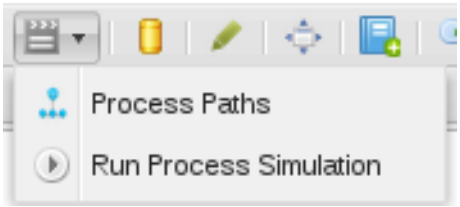
The simulation process runs in the Simulation engine extension, which relies on the possible execution paths rather than Process data. On simulation, the engine generates events for every simulated activity, which are stored in the simulation repository.

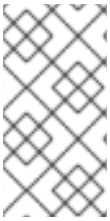
Simulation input data include general data about the Process simulation as well as simulation data for individual Process Elements. Process Elements executed by the engine automatically do not require any input data; however, the Process itself, Human Tasks, Intermediate Event, and Flows leaving a split Gateway, need such data: further information on Simulation data is available in Section C.1, “Process” and the subsequent sections.

7.1. PATH FINDER

Path Finder is a tool that allows you to identify all possible paths a Process execution can take.

Before you identify the paths, make sure your Process is valid. Then, on the toolbar, click **Process Simulation** () and **Process Paths**.



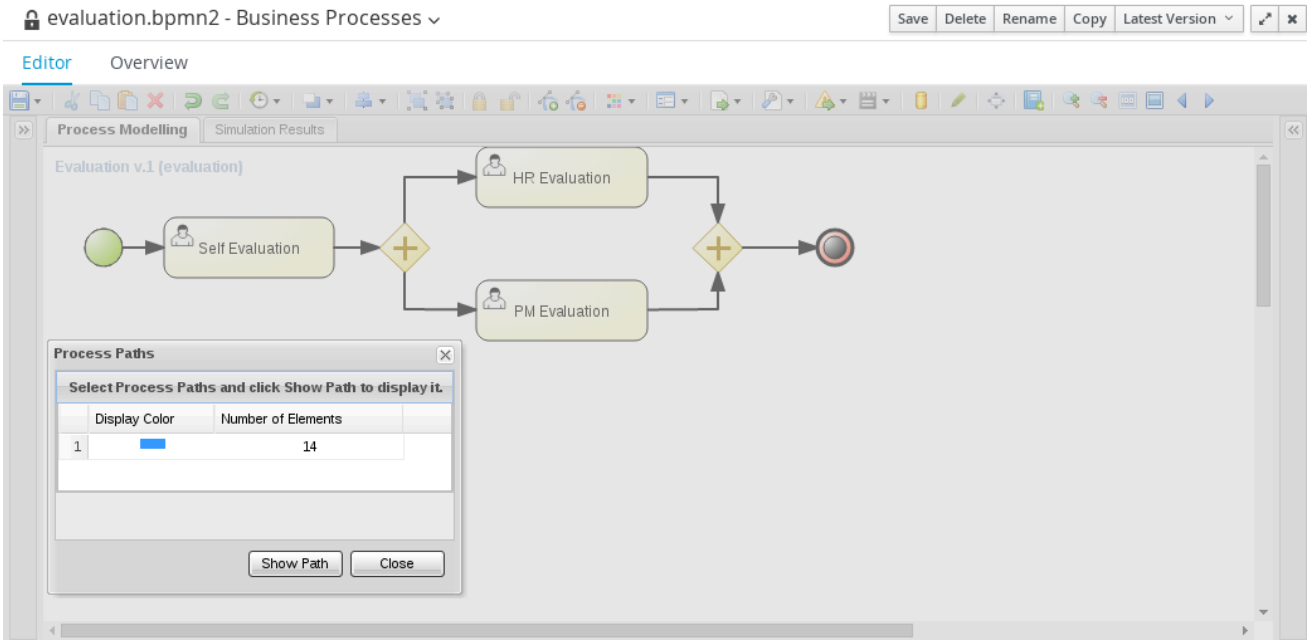


NOTE

Note that when you click this button only core process paths are searched for. In order to view Embedded or Event subprocess paths, you have to click on the subprocess, making sure that it is selected and then click the **Process Path** button. This will focus on paths that are specific to this subprocess.

A dialog with data on individual path appears: to visualize any of the identified paths, select the path in the dialog and click **Show Path**.

Figure 7.1. Process Paths



7.2. SIMULATING A PROCESS

7.2.1. Defining Simulation details on Elements

To define the input data for a Process simulation, you need to define the Simulation data on the Process and its Elements in the **Properties** tab.

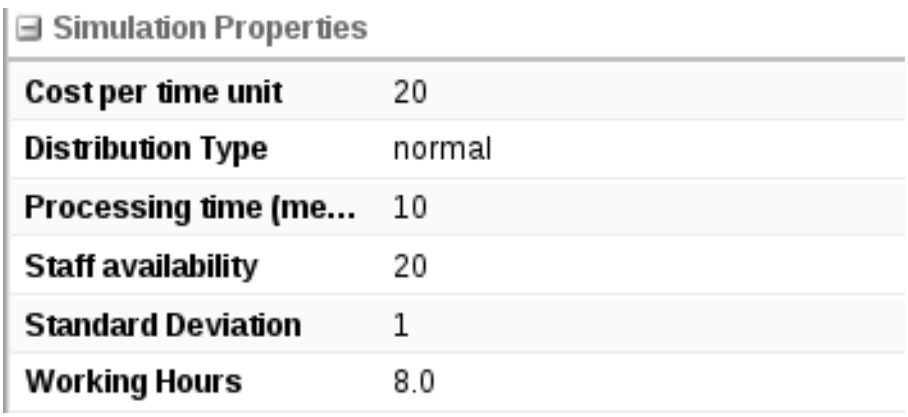
Information on Simulation data for individual Process Elements and the Process itself are available in Section C.1, “Process” and subsequent sections.

7.2.2. Running a Simulation

To run a Process Simulation, do the following:

1. Open the Process in the Process Designer and make sure you have defined the simulation parameters for individual Elements.

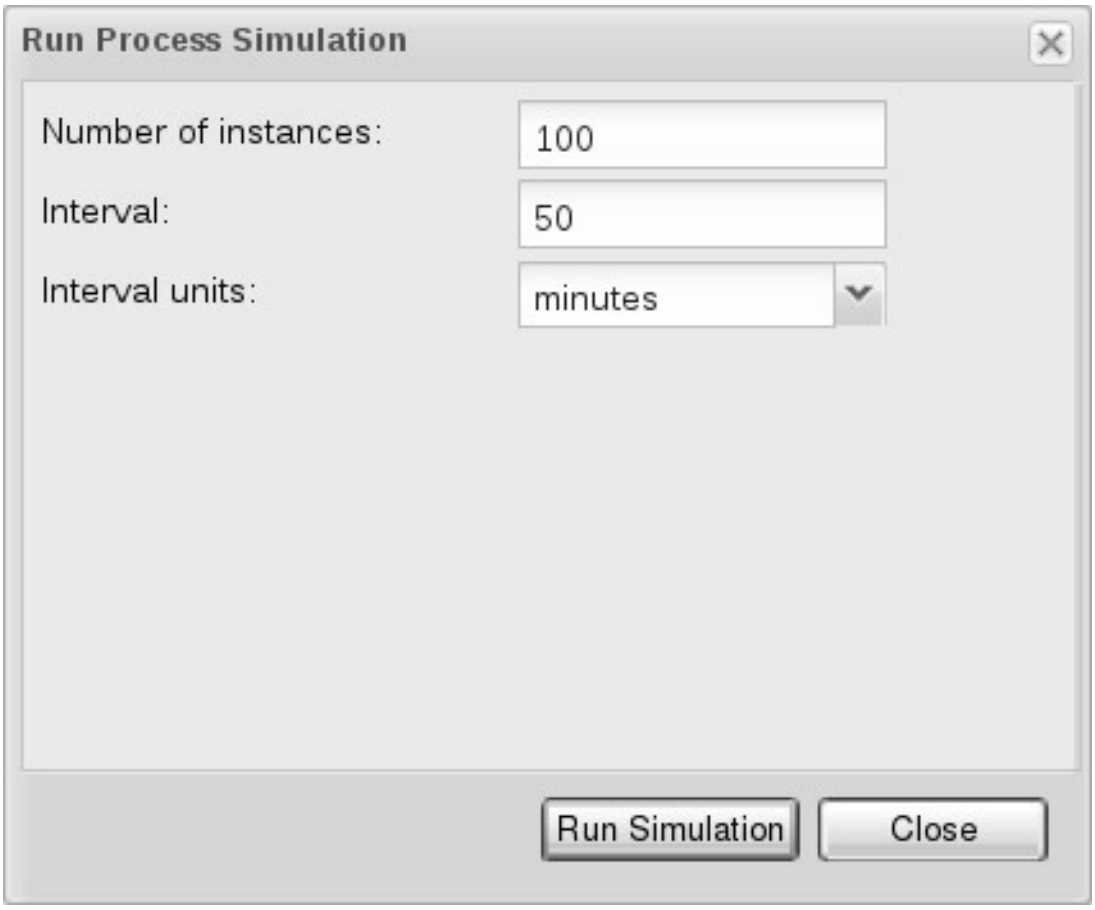
   Figure 7.2. Simulation Properties

2. Click **Process Simulation** ( ) icon in the toolbar and then click the **Run Simulation** entry.

3. In the **Run Process Simulation** dialog window, define the simulation session details:

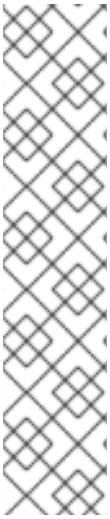
Figure 7.3. Run Process Simulation properties dialog window



   - Number of instance: number of Process instances to be created and triggered
   - Interval: interval between individual Process instantiations
   - Interval units: time of unit the **`Interval`** is defined in

4. Click **Run Simulation**.

After you start the simulation, the Process Designer focuses the **Simulation Results** tab with the simulation Process results displayed in the **Simulation Graph** pane on the right.



NOTE

If the simulation fails, the following message may be displayed:

```
Unable to perform simulation:undefined
```

If that happens, make sure your Process is not too complex—split it into multiple Processes connected by the Call Activity. Examples of complex Processes:

- A Process containing a complex series of XOR and AND gateways.
- A Process containing loops where one instance of the loop is not finished when another one is starting.

7.2.3. Examining Simulation results

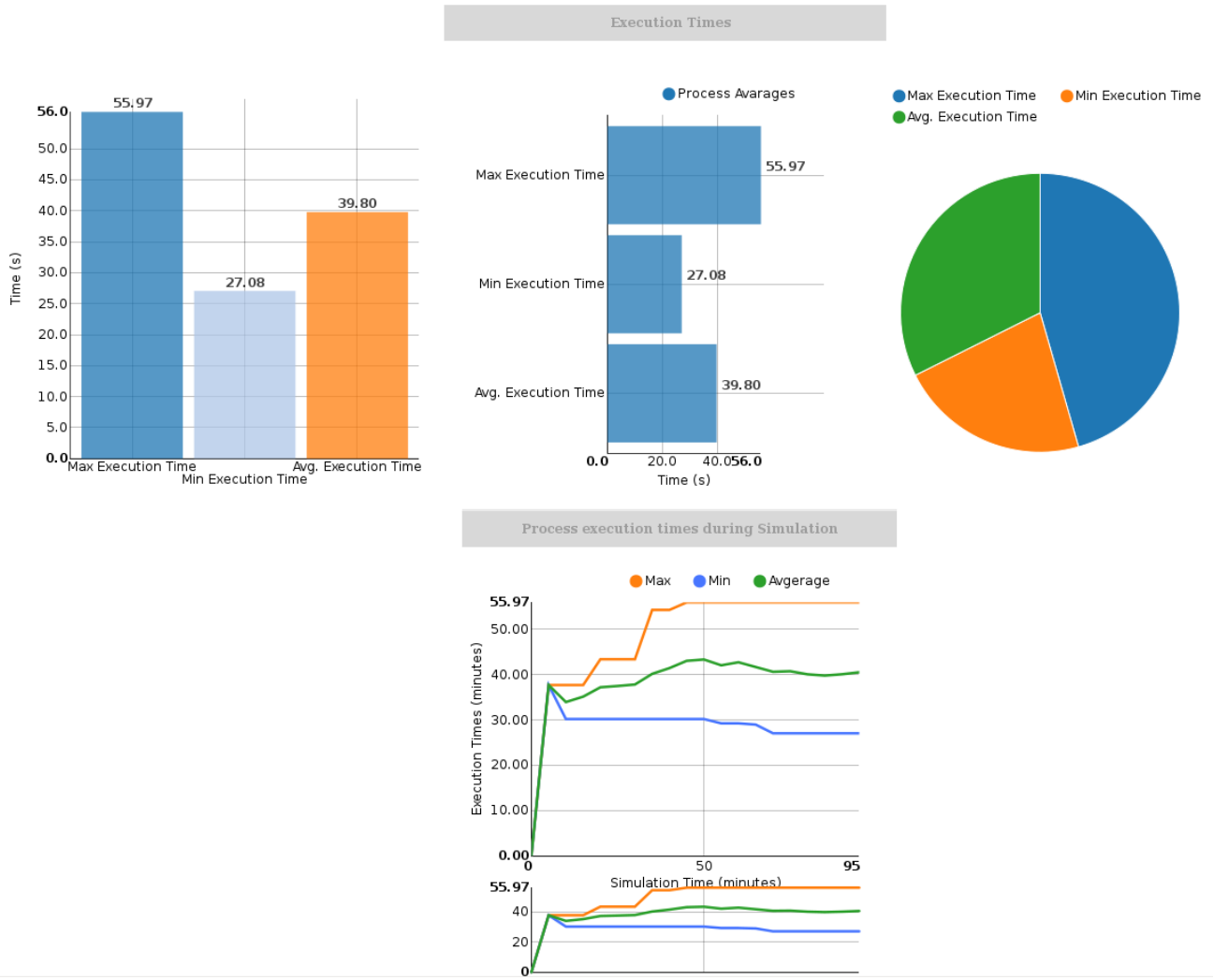
After you run a Process simulation, the Process Designer focuses the **Simulation Results** tab. The tab show the list of available simulation result in the **Simulation Graphs** on the right.

The results are divided into three sections with graphs:

- The Process section with general Process simulation graphs
- The Activities section with individual Activities' simulation graphs
  Activities graphs for Human Tasks include **Execution Time** with the Max, Min, and Average execution time for the given Activity, **Resource Utilization** for the hours a resource has been used, and the **Cost Parameters** graph if applicable (if you defined the Cost parameter for the Activity). For Script Tasks only the **Execution Time** with the Max, Min, and Averages execution time, is available.
- The Paths section with simulation graph of the Paths taken during the simulation. The graphs contain the Process model with the respective Path highlighted and execution statistics on the Path.

Graph types
Click a graph entry to display the graph in the canvas area: the graph is visualized as a vertical bar chart; however, you can change the graph visualization type by clicking on the respective icon in the upper right corner of the canvas area.

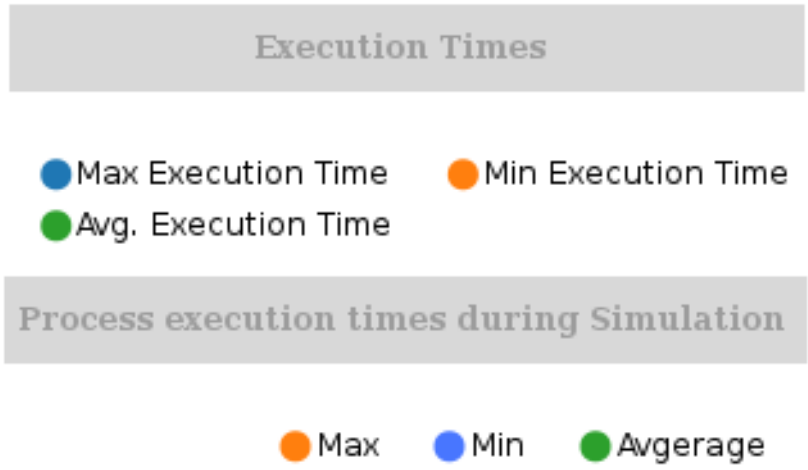
Figure 7.4. Simulation Graph types



Filters

To filter data in a chart, click the item radiobutton in the chart legend.

Figure 7.5. Graph item radiobutton



Timeline

The Timeline feature allows you to view the graph at the particular stage during simulation execution. Every event is included in the timeline as a new status.

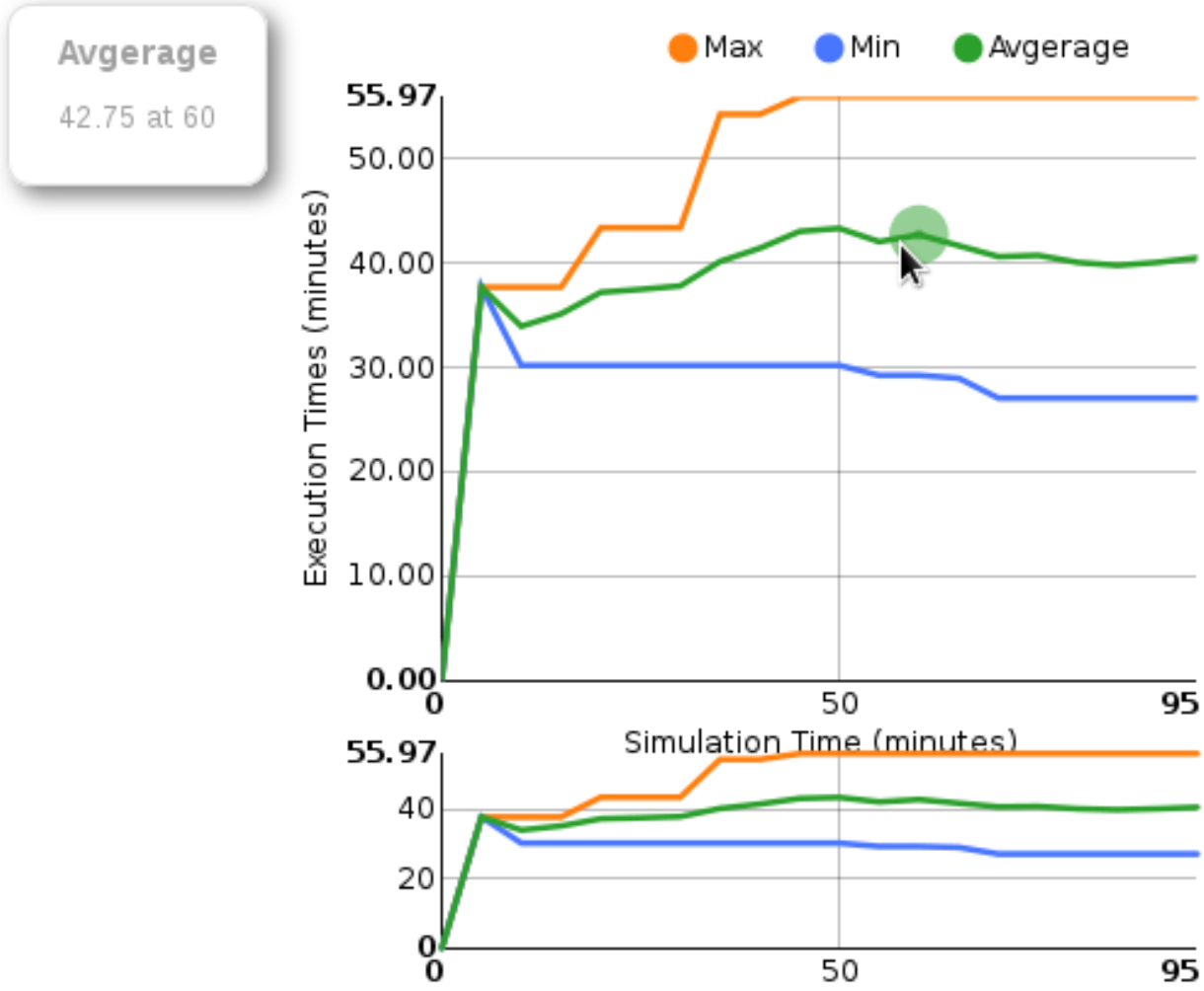
To activate the feature, click the **Timeline** in the upper right corner of the respective graph: The timeline depicting individual events is displayed in the lower part of the canvas. Click the arrows on the right and left from the chart to move through the timeline. The data current for the particular moment are applied to the chart depicted above instantly.

Figure 7.6. Process Simulation Timeline

Note that in line charts, you can point to a point on a line to see the value of the item at

the given time.

Figure 7.7. Line Chart



CHAPTER 8. TESTING

Even though business processes should not be viewed as code, should be as high-level as possible and should not contain implementation details, they also have a life cycle just like other development artefacts. Therefore testing you Process definitions is just as important as it is when programming.

8.1. UNIT TESTING

When unit testing your Process, you test whether the process behaves as expected in specific use cases; for example, you test the output based on the existing input. To simplify unit testing, the helper class **`org.jbpm.test.JbpmJUnitBaseTestCase`** is provided in the jbpm-bpmn2 test module that offers the following:

- helper methods to create a new kie base and session for given processes (Also, you can select if persistence is to be used.)
- assert statements to check among other also the following:
  - the state of a process instance (active, completed, aborted)
  - which node instances are currently active
  - which nodes have been triggered to check the path that has been followed
  - the value of variables

Example 8.1. JUnit test of the com.sample.bpmn.hello Process

```
  import org.kie.api.runtime.KieSession;
  import org.kie.api.runtime.manager.RuntimeEngine;
  import org.kie.api.runtime.process.ProcessInstance;

  public class MyProcessTest extends org.jbpm.test.JbpmJUnitBaseTestCase
{

    public void testProcess() {

      // create singleton runtime manager and load the given process(es)
      createRuntimeManager("sample.bpmn");

      // get the single kie session
      RuntimeEngine engine = getRuntimeEngine();
      KieSession ksession = engine.getKieSession();

      // start the process
      ProcessInstance processInstance =
      ksession.startProcess("com.sample.bpmn.hello");

      // check whether the process instance has completed successfully
     assertProcessInstanceCompleted(processInstance.getId(), ksession);

      // check whether the given nodes were executed during the process
execution
     assertNodeTriggered(processInstance.getId(), "StartProcess",
"Hello", "EndProcess");
```

```
        }
}
```

The JUnit test will create a new session, start the com.sample.bpmn.hello process and verify whether the Process instance completed successfully and whether the nodes StartProcess, Hello, EndProcess have been executed.

8.2. SESSION CREATION

To create a session, it is required to first create a RuntimeManager and RuntimeEngine from which you will get the session. The following methods can be used to create RuntimeManager:

- **createRuntimeManager(String… process)** creates default configuration of the RuntimeManager with singleton strategy and all processes added to the kie base. There will only be one RuntimeManager created during single test and the processes shall be added to the kie base.
- **createRuntimeManager(Strategy strategy, String identifier, String… process)** creates default configuration of RuntimeManager with given strategy and all processes being added to the kie base. **Strategy** selects the strategies that are supported, and **identifier** identifies the RuntimeManager.
- **createRuntimeManager(Map<String, ResourceType> resources)** creates default configuration of RuntimeManager with singleton strategy and all resources being added to kie base. The **resources** code identifies the processes, rules, etc that shall be added to the kie base.
- **createRuntimeManager(Map<String, ResourceType> resources, String identifier)** creates default configuration of RuntimeManager with singleton strategy and all resources added to kie base. Like the method above but with an **identifier** that identifies the RuntimeManager.
- **createRuntimeManager(Strategy strategy, Map<String, ResourceType> resources)** creates default configuration of RuntimeManager with given strategy and all resources being added to the kie base. There will be only one RuntimeManager created during single test. The **strategy** code is the selected strategy of those that are supported. The **resources** code are all the resources that shall be added to the kie base.
- **createRuntimeManager(Strategy strategy, Map<String, ResourceType> resources, String identifier)** creates default configuration of RuntimeManager with given strategy and all resources being added to kie base. There will be only one RuntimeManager created during single test. The **strategy** code selects the supported strategies. The **resources** code identifies the resources that shall be added to the kie base. The **identifier** code identifies the RuntimeManger.
- **createRuntimeManager(Strategy strategy, Map<String, ResourceType< resources, RuntimeEnvironment environment, String identifier)** is the lowest level of creation of RuntimeManager that expects to get RuntimeEnvironment to be given as an argument. It does not assume any particular configuration; that is, it allows you to configure every single piece of RuntimeManager manually. The **strategy** code selects the strategies of those that

are supported. The **resources** code identifies the resources added to the kie base. The **environment** code is the runtime environment used for RuntimeManager creation. The **identifier** code identifies the RuntimeManager.

The following methods can be used to get RuntimeEngine:

- **getRuntimeEngine()** returns a new RuntimeEngline built from the manager of the test case. It uses EmptyContext that is suitable for the following strategies: singleton and request.
- **getRuntimeEngine(Context<?> context)** returns a new RuntimeEngine built from the manager of the test case. Common use case would be to maintain the same session for the process instance and thus a ProcessInstanceIdContext shall be used. The **context** code is the instance of the context that shall be used to create RuntimeManager.

8.2.1. Assertions

The following assertions are available for testing the current state of a process instance:

- **assertProcessInstanceActive(long processInstanceId, KieSession ksession)**: checks whether the Process instance with the given id is active.
- **assertProcessInstanceCompleted(long processInstanceId, KieSession ksession)**: checks whether the Process instance with the given id has completed. successfully
- **assertProcessInstanceAborted(long processInstanceId, KieSession ksession)**: checks whether the Process instance with the given id was aborted.
- **assertNodeActive(long processInstanceId, KieSession ksession, String… name)**: checks whether the process instance with the given id contains at least one active node with the given node names.
- **assertNodeTriggered(long processInstanceId, String… nodeNames)**: checks for each given node name whether a node instance was triggered during the execution of the Process instance.
- **getVariableValue(String name, long processInstanceId, KieSession ksession)**: retrieves the value of the variable with the given name from the given Process instance.

8.2.2. Integration with external services

To test possible scenarios connected to collaboration of Process Tasks with external services, you can use **TestWorkItemHandler**. **TestWorkItemHandler** is provided by default can be registered to collect all Work Items of a given type, for example sending an email or invoking a service, and contains all the data related to that task).

This test handler can be queried during unit testing to check whether specific work was requested during the execution of the Process and that the data associated with the work was correct.

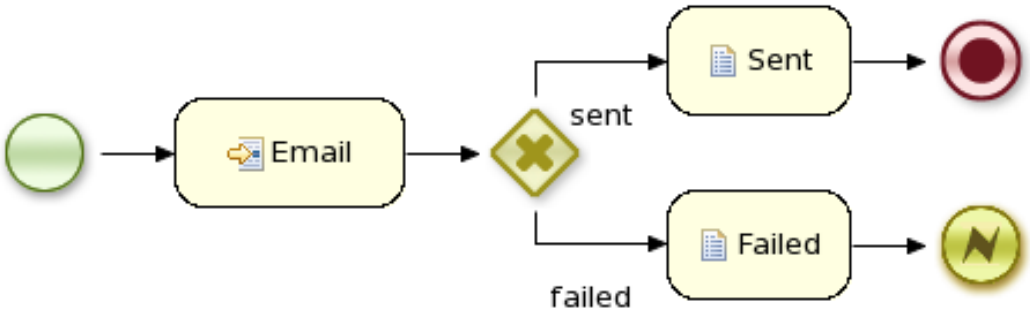
> **Example 8.2. Testing an Email Task**
>
> Let’s assume we want to test the Process depicted in Example 8.3, “Process with a

custom Email Service Task". The test should in particular check if an exception is raised when the email sending fails. The failure is simulated by notifying the engine that the sending the email could not be completed:

```
import org.kie.api.runtime.manager.RuntimeManager;
import org.kie.api.runtime.manager.RuntimeEngine;
import org.kie.api.runtime.KieSesssion;
import org.kie.api.runtime.process.WorkItem;
import org.kie.api.runtime.process.WorkItemHandler;
import org.kie.api.runtime.process.ProcessInstance;
import org.kie,api.runtime.process.WorkItemManger;

  public void testProcess2() {

  // create runtime manager with single process - hello.bpmn
  createRuntimeManager("sample-process.bpmn");

  // take RuntimeManager to work with process engine
  RuntimeEngine runtimeEngine = getRuntimeEngine();

  // get access to KieSession instance
  KieSession ksession = runtimeEngine.getKieSession();

  // register a test handler for "Email"
  TestWorkItemHandler testHandler = getTestWorkItemHandler();
  ksession.getWorkItemManager().registerWorkItemHandler("Email",
testHandler);

  // start the process
  ProcessInstance processInstance =
ksession.startProcess("com.sample.bpmn.hello2");
  assertProcessInstanceActive(processInstance.getId(), ksession);
  assertNodeTriggered(processInstance.getId(), "StartProcess", "Email");

  // check whether the email has been requested
  WorkItem workItem = testHandler.getWorkItem();
  assertNotNull(workItem);
  assertEquals("Email", workItem.getName());
  assertEquals("me@mail.com", workItem.getParameter("From"));
  assertEquals("you@mail.com", workItem.getParameter("To"));

  // notify the engine the email has been sent
  ksession.getWorkItemManager().abortWorkItem(workItem.getId());
  assertProcessInstanceAborted(processInstance.getId(), ksession);
  assertNodeTriggered(processInstance.getId(), "Gateway", "Failed",
"Error");

}
```

The test case uses a test handler that registers when an email is requested and allows you to test the data related to the email. Once the engine has been notified the email could not be sent by the **abortWorkItem(..)** method call, the unit test verifies that the Process handles this case by logging the fact and terminating with an error.

Figure 8.1. Process with a custom Email Service Task



8.2.3. Persistence

To simplify unit testing, jBPM includes a helper class called **JbpmJUnitBaseTestCase** in the jbpm-test module that greatly simplifies your junit testing. This helper class provides methods to create a new **RuntimeManager** and **RuntimeEngine** for a given process or set of processes. By default, persistence is not used, and it is controlled by the super constructor. The helper method allows you to select whether or not you wish to use persistence. The example below shows a helper class to allow persistence:

Example 8.3. Process with a custom Email Service Task

```
public class ProcessHumanTaskTest extends JbpmJUnitBaseTestCase {

    public ProcessPersistenceTest() {

        // setup data source, enable persistence

        super(true, true);

    }
     ....
```

CHAPTER 9. INTELLIGENT PROCESS SERVER

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

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

You can provision the Intelligent Process Server instances through Business Central. In this chapter, you will go through the steps required to setup the Intelligent Process Server, provision and connect to this server through Business Central, control what rule artifacts go in each instance and go through its lifecycle.

9.1. DEPLOYING THE INTELLIGENT PROCESS SERVER

The Intelligent Process Server is distributed as a web application archive (WAR) file with the name of **kie-server.war**. When you install Red Hat JBoss BPM Suite, this WAR file is installed and deployed in your web container by default.

You can copy this WAR file and deploy it in any other web container (Red Hat JBoss Web Server or another Red Hat JBoss EAP install, for example) and have this server available from other web containers. Note that you must deploy the WAR file that is compatible for your web container.

1. Once you have deployed the WAR file (or if you have it deployed on the same web container where Red Hat JBoss BPM Suite is deployed), create a user with the role of **kie-server** in this web container.

2. Test that you can access the decision engine by navigating to the endpoint in a browser window: **http://SERVER:PORT/kie-server/services/rest/server/** with the web container running. You will be prompted for your username/password that you created in the previous step.

3. Once authenticated, you will see an XML response in the form of engine status, similar to this:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response type="SUCCESS" msg="Kie Server info">
 <kie-server-info>
  <capabilities>BPM</capabilities>
 <capabilities>KieServer</capabilities>
 <capabilities>BRM</capabilities>
 <location>http://localhost:8230/kie-
server/services/rest/server</location>
 <name>KieServer@/kie-server</name>
 <id>15ad5bfa-7532-3eea-940a-abbbbc89f1e8</id>
 <version>6.4.0.Final-redhat-5</version>
 </kie-server-info>
</response>
```

9.2. INSTALLING THE INTELLIGENT PROCESS SERVER IN

OTHER CONTAINERS

9.2.1. JBoss Web Server 2.x/3.x, Tomcat 8.x/9.x

Use the following procedure to install the Intelligent Process Server in a Tomcat container.

Procedure:

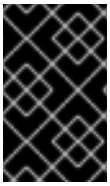
1. Download and unzip the Tomcat distribution.
2. Download **kie-server-6.4.0.Final-webc.war** and place it into **TOMCAT_HOME/webapps**.
3. Configure user(s) and role(s). Make sure that **TOMCAT_HOME/conf/tomcat-users.xml** contains the following username and role definition. The username and password should be unique, however ensure that the user has the role **kie-server**:

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

4. Start the server by running **TOMCAT_HOME/bin/startup.[sh|bat]**. You can check out the Tomcat logs in **TOMCAT_HOME/logs** to see if the application deployed successfully. See Section 9.3.1, “Bootstrap Switches” for the bootstrap switches that can be used to properly configure the instance. For instance:

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

5. Verify the server is running. Access **http://SERVER:PORT/kie-server/services/rest/server/** and type the specified username and password. You should see simple XML message with basic information about the server.



IMPORTANT

You cannot leverage the JMS interface when running on Tomcat, or any other Web container. The Web container version of the WAR contains only the REST interface.

9.3. INTELLIGENT PROCESS SERVER SETUP

9.3.1. Bootstrap Switches

The Intelligent Process Server accepts a number of bootstrap switches (system properties) to configure the behavior of the server. The following is a table of all the supported switches:

NOTE

Some properties are marked as required when using a controller, so you need to set these properties when you handle Intelligent Process Server container creation and removal in Business Central. If you use the Intelligent Process Server separately—without any interaction with Business Central—you do not have to set these properties.

Intelligent Process Server Bootstrap Switches

kie.maven.settings.custom

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

Values	Default
Path	N/A

kie.server.jms.queues.response

The JNDI name of response queue for JMS.

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

org.drools.server.ext.disabled

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

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

org.drools.server.filter.classes

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

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

org.jbpm.ht.callback

Specifies the implementation of user group callback to be used:

- **`mvel`**: Default; mostly used for testing.
- **`ldap`**: LDAP; requires additional configuration in the **`jbpm.usergroup.callback.properties`** file.

- **db**: Database; requires additional configuration in the **jbpm.usergroup.callback.properties** file.
- **jaas**: JAAS; delegates to the container to fetch information about user data.
- **props**: A simple property file; requires additional file that will keep all information (users and groups).
- **custom**: A custom implementation; you must specify the fully qualified name of the class in the **org.jbpm.ht.custom.callback**.

Values	Default
mvel, **ldap**, **db**, **jaas**, **props**, or **custom**	**mvel**

org.jbpm.ht.custom.callback

A custom implementation of the **UserGroupCallback** interface in case the **org.jbpm.ht.callback** property is set to **custom**.

Values	Default
Fully qualified name	N/A

org.jbpm.server.ext.disabled

When set to **true**, disables the BPM support (for example, processes support).

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

org.jbpm.ui.server.ext.disabled

When set to **true**, disables the Intelligent Process Server UI extension.

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

org.kie.executor.disabled

Disables the Red Hat JBoss BPM Suite executor.

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

org.kie.executor.interval

The time between the moment the Red Hat JBoss BPM Suite executor finishes a job and

the moment it starts a new one, in a time unit specified in `org.kie.executor.timeunit`.

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

org.kie.executor.pool.size

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

Values	Default
Number (**`Integer`**)	1

org.kie.executor.retry.count

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

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

org.kie.executor.timeunit

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

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

org.kie.server.bypass.auth.user

Allows to bypass the authenticated user for task-related operations, for example queries.

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

org.kie.server.controller

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

Values	Default
Comma-separated list	N/A

org.kie.server.controller.connect

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

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

org.kie.server.controller.pwd

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

Values	Default
String	**kieserver1!**

org.kie.server.controller.token

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

Values	Default
String	N/A

org.kie.server.controller.user

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

Values	Default
String	**kieserver**

org.kie.server.domain

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

Values	Default
String	N/A

org.kie.server.id

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

Values	Default
String	N/A

org.kie.server.location

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

Values	Default
URL	N/A

org.kie.server.persistence.dialect

The Hibernate dialect to be used. You must set this property when enabling BPM support.

Values	Default
String	N/A

org.kie.server.persistence.ds

A data source JNDI name. You must set this property when enabling BPM support.

Values	Default
String	N/A

org.kie.server.persistence.schema

The database schema to be used.

Values	Default
String	N/A

org.kie.server.persistence.tm

A transaction manager platform for Hibernate properties set. You must set this property when enabling BPM support.

Values	Default
String	N/A

org.kie.server.pwd

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

Values	Default
String	**kieserver1!**

org.kie.server.repo

The location where Intelligent Process Server state files will be stored.

Values	Default
Path	**.**

org.kie.server.sync.deploy

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

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

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

org.kie.server.token

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

Values	Default
String	N/A

org.kie.server.user

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

Values	Default
String	**kieserver**

org.optaplanner.server.ext.disabled

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

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

IMPORTANT

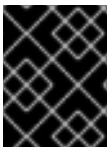
A new DataSource for the Intelligent Process Server must point to a different database schema than the DataSource used by Business Central.

If you are running both the Intelligent Process Server and the Business Central, make sure that each uses a different DataSource (by modifying the **org.kie.server.persistence.ds** property), in order to avoid conflicts.

9.3.2. Managed Intelligent Process Server

A managed instance requires an available controller to start up the Intelligent Process Server.

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



IMPORTANT

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

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

NOTE

To run the Intelligent Process Server in standalone mode without connecting to controllers, see Section 9.3.4, “Unmanaged Intelligent Process Server”.

9.3.3. Registering Intelligent Process Server

To register a new managed Intelligent Process Server instance, set the following properties in **standalone.xml**:

```
<property name="org.kie.server.user" value="anton"></property>
<property name="org.kie.server.pwd" value="password1!"></property>
<property name="org.kie.server.location" value="http://localhost:8080/kie-
server/services/rest/server"></property>
<property name="org.kie.server.controller"
value="http://localhost:8080/business-central/rest/controller"></property>
<property name="org.kie.server.controller.user"
value="kieserver"></property>
<property name="org.kie.server.controller.pwd"
value="kieserver1!"></property>
<property name="org.kie.server.id" value="local-server-123"></property>
```

The **standalone.xml** file is located in the **$SERVER_HOME/standalone/configuration/** directory.

org.kie.server.user must have the **kie-server** role assigned.

WARNING

org.kie.server.location points to the same host as **org.kie.server.controller**. This is not suitable for production.

To create the **kieserver** user:

1. Change into **$SERVER_HOME/bin/**.
2. Execute the following command:

```
$ ./add-user.sh -a --user kieserver --password kieserver1! --role
kie-server
```

To verify successful start of the Intelligent Process Server, send a GET request to **http://SERVER:PORT/kie-server/services/rest/server/**. Once authenticated, you will see an XML response:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<response type="SUCCESS" msg="Kie Server info">
    <kie-server-info>
        <capabilities>BRM</capabilities>
        <capabilities>BPM-UI</capabilities>
        <capabilities>BPM</capabilities>
        <capabilities>BRP</capabilities>
        <capabilities>KieServer</capabilities>
        <location>http://localhost:8080/kie-
server/services/rest/server</location>
        <messages>
            <content>Server KieServerInfo{serverId='local-server-123',
```

```
version='6.4.0.Final-redhat-3', location='http://localhost:8080/kie-
server/services/rest/server'}started successfully at Fri Jun 03 13:48:44
CEST 2016</content>
            <severity>INFO</severity>
            <timestamp>2016-06-03T13:48:44.606+02:00</timestamp>
        </messages>
        <name>local-server-123</name>
        <id>local-server-123</id>
        <version>6.4.0.Final-redhat-3</version>
    </kie-server-info>
</response>
```

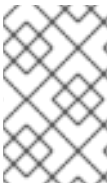
To verify successful registration, log into the Business Central and select **Deploy → Rule Deployments**. If successful, you will see the registered server ID.

9.3.4. Unmanaged Intelligent Process Server

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

The configuration is updated during the following operations:

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



NOTE

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

9.4. CREATING A CONTAINER

Once your Intelligent Process Server is registered, you can start adding containers to it. Containers are self-contained environments that have been provisioned to hold instances of your packaged and deployed rule instances. To create a container, follow these steps:

1. Log into Business Central.
2. Click **Deploy → Rule Deployment**.
3. In the **SERVER TEMPLATES** section on the left, select your server.
4. Click **Add Container** to see the **New Container** wizard.
5. Add a name of your container under **Name**.
6. Click **Search** and select a built project you wish to deploy into the container. Alternatively, enter **`Group Name`**, **`Artifact Id`** and **`Version`** manually.

WARNING

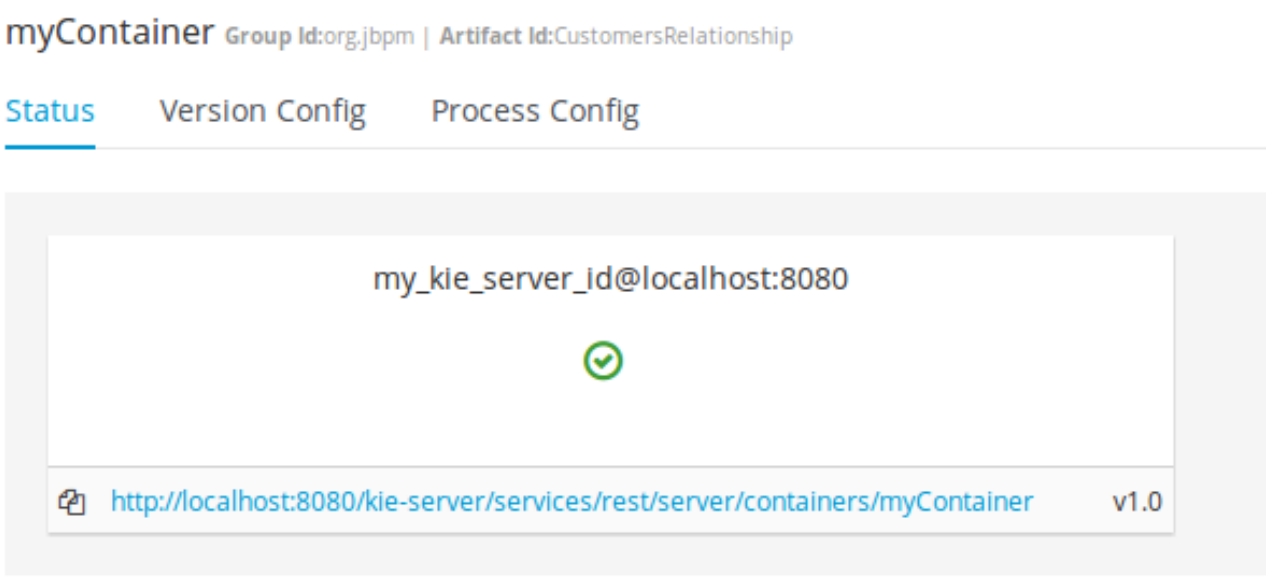
When you enter the container version number, do not use the **LATEST** or **RELEASE** keywords. This feature has been deprecated and can cause deployment issues.

7. Click **Next** to configure Runtime strategy, Kie Base and Kie Session for your container.
   Alternatively, click **Finish** to skip configuration of Runtime strategy, Kie Base and Kie Session for your container.

8. Click **Finish**.

After you successfully create your container, click **Start**.

Figure 9.1. Container in Started Mode



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

Example 9.1. Server Response

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

```
        <artifact-id>CustomersRelationship</artifact-id>
        <group-id>org.jbpm</group-id>
        <version>1.0</version>
      </release-id>
      <resolved-release-id>
        <artifact-id>CustomersRelationship</artifact-id>
        <group-id>org.jbpm</group-id>
        <version>1.0</version>
      </resolved-release-id>
      <scanner status="DISPOSED"/>
    </kie-container>
  </response>
```

9.5. MANAGING CONTAINERS

Containers within the Intelligent Process Server can be started, stopped, and updated from within Business Central.

9.5.1. Starting, Stopping, and Deleting a Container

A container is stopped by default. To start it, follow these steps:

1. Log into Business Central.
2. Click **Deploy** → **Rule Deployment**.
3. In the **SERVER TEMPLATES** section on the left, click on your Intelligent Process Server ID.
4. In the **KIE CONTAINERS** section, click on the container you want to start.
5. Click **Start**. Alternatively, click **Stop** to stop a running container. Once a container is stopped, you can click **Remove** to remove it.

9.5.2. Upgrading a Container

You can update deployed containers without needing to restart the Intelligent Process Server. This is useful in cases where the business rule changes cause new versions of packages to be provisioned. You can have multiple versions of the same package provisioned and deployed. To upgrade a container, follow these steps:

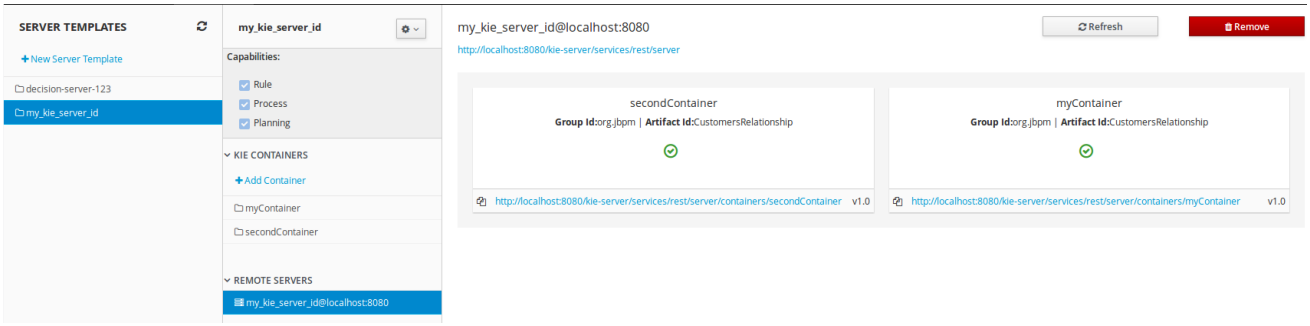
1. Log into Business Central.
2. Click **Deploy** → **Rule Deployment**.
3. In the **KIE CONTAINERS** section, click on the container you want to upgrade.
4. Click on the **Version Config** tab.
5. Enter the new version in the **Version** textfield and click **Upgrade**.

9.5.3. Managing Multiple Containers

You can create and provision multiple containers on your Intelligent Process Server.

Select your server in the **REMOTE SERVERS** section to view all the containers on your server and their status.

Figure 9.2. Managing Multiple Containers



PART III. PLUG-IN

CHAPTER 10. PLUG-IN

Red Hat JBoss BPM Suite comes with a plug-in for Red Hat JBoss Developer Studio to provide support for the development of business processes in the Eclipse-based environment, such as debugging and testing. It also provides a graphical Process Designer for business process editing.

Note that the repository structure follows the maven structure and is described in Chapter 3, *Project*.

For instructions on how to install and set up the plug-in, see the *Red Hat JBoss BPM Suite Installation Guide*.

10.1. CREATING BPM PROJECT

Prerequisite

Ensure that you have installed Red Hat JBoss BPM Suite and Red Hat JBoss BRMS plug-ins and runtime environments. For more information, see *Red Hat JBoss BPM Suite Installation Guide*.

To create a BPM project:

1. On the main menu of Red Hat JBoss Developer Studio, click **File → New → Other...**.
2. Choose **jBPM → jBPM project**.
3. In the **Create New jBPM Project** dialog, select the required content and click **Next**.
4. If you did not decide for project with online examples, specify the project name, location, and type:
   - **`Java and jBPM Runtime classes`**: select the runtime to be used by the project or click **Manage Runtime Definitions...** and define a new runtime (for details on runtime resources, see the *Red Hat JBoss BPM Suite Installation Guide*).
   - **`Maven`**: specify maven properties of your project.

10.2. CREATING PROCESS

In JBoss Developer Studio with the Red Hat JBoss BPM Suite plug-in, a process is created the same way as other resources:

1. Choose **File → New → Other...**.
2. Select **jBPM → jBPM Process Diagram**.
3. In the displayed dialog box, define the name, package, and container of the process. The rest of the fields is completed automatically. Note that you must follow the Maven structure.

Once created, the process is opened for editing in the graphical Process Designer.

10.3. CHECKING SESSION LOGS

You can check the session logs in the audit log, which is a log of all events that were logged from the session. Audit log is an XML-based log file which contains a log of all the events that occurred while executing a specific ksession.

Procedure: Creating Logger

1. To create a logger, use **KieServices** and attach the logger to a**ksession**, for example:

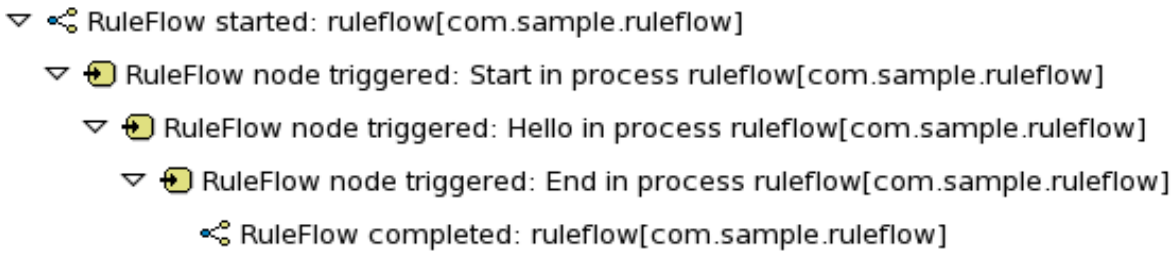
```
KieRuntimeLogger logger =
KieServices.Factory.get().getLoggers().newThreadedFileLogger(ksessio
n, "mylogfile", 1000);
// Do something with the ksession here.
logger.close();
```

2. Do not forget to close the logger when you finish using it.

Procedure: Using Audit View

1. To use **Audit View**, open **Window** → **Show View** → **Other...**.
2. Under the **Drools** category, select **Audit**.
3. To open a log file in **Audit View**, select the log file using the **Open Log** action in the top right corner, or simply drag and drop the log file from the *Package Explorer* or *Navigator* into the *Audit View*.
4. A tree-based view is generated based on the data inside the audit log. Depicted below is an example tree-based view:

   Figure 10.1. Tree-Based View



5. An event is shown as a subnode of another event if the child event is caused by a direct consequence of the parent event.

FILE-BASED LOGGER

The file-based logger will only save the events on close (or when a certain threshhold is reached). If you want to make sure the events are saved on a regular interval (for example during debugging), make sure to use a threaded file logger, so the audit view can be updated to show the latest state. When creating a threaded file logger, you can specify the interval after which events should be saved to the file (in milliseconds).

PART IV. DEPLOYMENT AND RUNTIME MANAGEMENT

CHAPTER 11. DEPLOYING AND MANAGING PROJECTS

Once you have created a project with your process definition and relevant resources, you need to build it and deploy it to the process engine. Once deployed, you can create process instances based on the deployed resources.

11.1. DEPLOYING A PROJECT

To deploy your project from Business Central, do the following:

1. Open the **Project Editor** in your project (navigate to your project using **Project Explorer** and click **Open Project Editor**).
2. You can define the Kie Base and Kie Session properties. If not, the default kbase and ksession will be used.
3. On the title bar, click **Build** → **Build & Deploy**.

NOTE

From the 6.1 version of Red Hat JBoss BPM Suite, deployment units are stored inside the database instead of the GIT repository. To override this behavior, set the **`org.kie.git.deployments.enabled`** property to true.

11.1.1. Duplicate GAV Detection

Every time you perform any of the operations listed below, all Maven repositories are checked for duplicate **`GroupId`**, **`ArtifactId`**, and **`Version`**. If a duplicate exists, the performed operation is cancelled.

The duplicate GAV detection is executed every time you:

- Create a new managed repository.
- Save a project definition in the Project Editor.
- Add new modules to a managed multi-module repository.
- Save the **`pom.xml`** file.
- Install, build, or deploy a project.

The following Maven repositories are checked for duplicates:

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

Users with the **`admin`** role can modify the list of affected repositories. To do so, open your project in the Project Editor and click **Project Settings: Project General Settings** → **Validation**.

Figure 11.1. List of Repositories to Be Checked

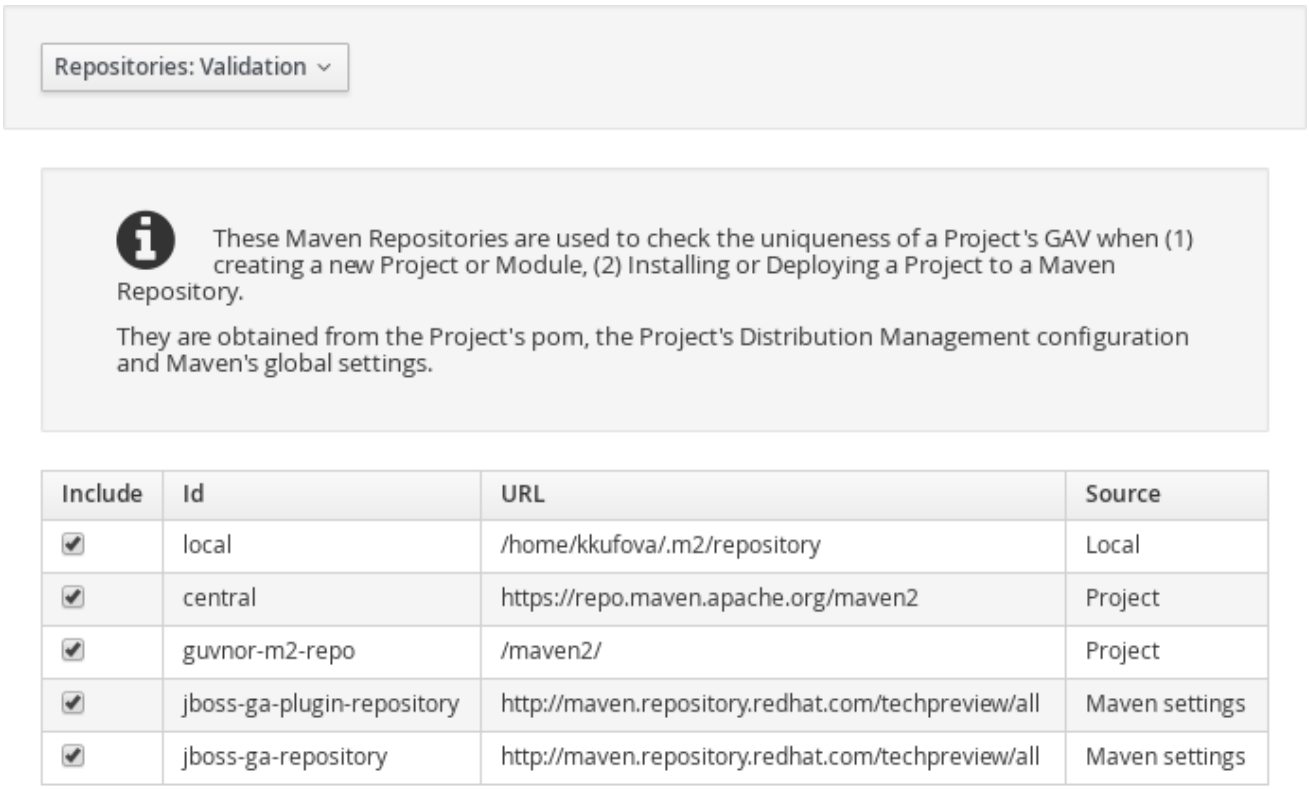
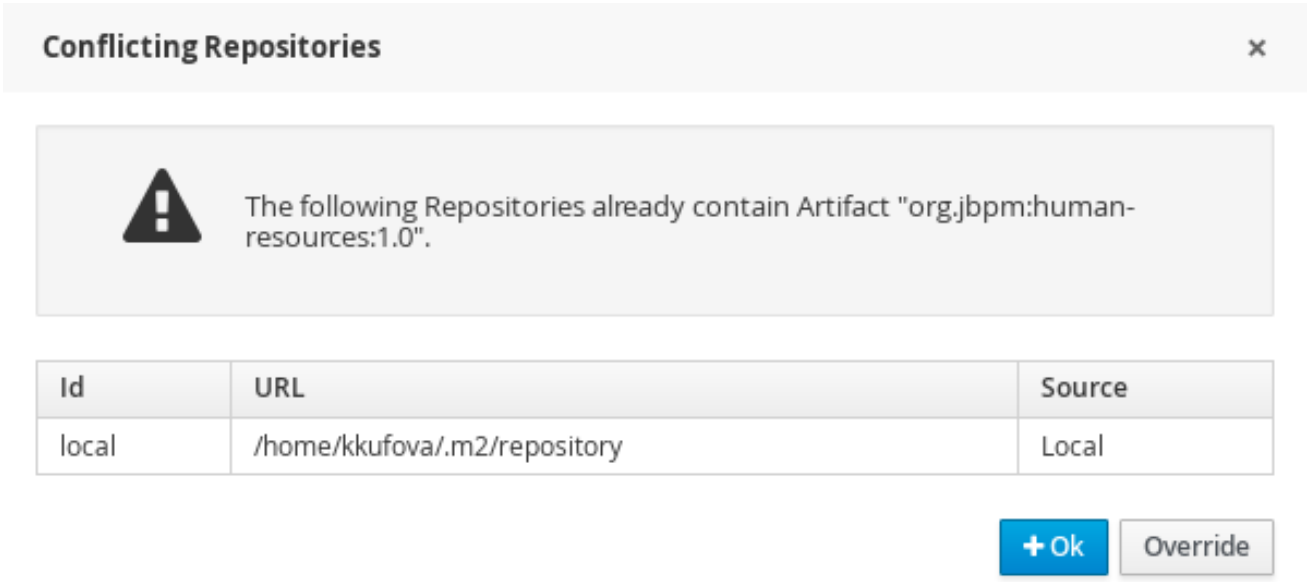
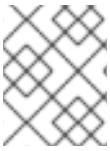


Figure 11.2. Duplicate GAV Detected





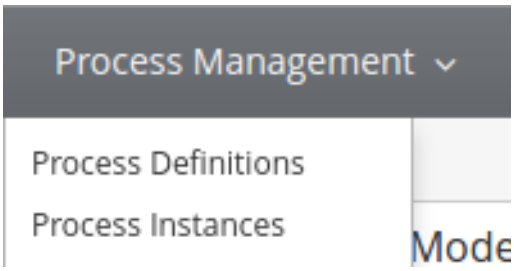
NOTE

To disable this feature, set the **`org.guvnor.project.gav.check.disabled`** system property to **`true`**.

11.2. PROCESS MANAGEMENT

The following sections describe the features provided by the options available under the Process Management menu in Business Central.

Figure 11.3. Process Management



11.2.1. Process Definitions

Once you have created, configured, and deployed your project comprising your business processes, you can view the list of all the process definitions in the **Process Definition List** under **Process Management → Process Definitions**. The process definition view comprises two main sections:

- Process Definition Lists
- Process Definition Details

The process definition list shows all the available process definitions that are deployed into the platform. If you click on any of the process definition listed in the Process Definitions List, the corresponding Process Definition details displays information about the process definition such as if there is a sub-process associated with it, or how many users and groups exist in the process definition. In the Process Definition details section, you can navigate to **Options → View Process Instances** to view associated process instances.

11.2.2. Process Instances

You can create new process instances from the **Process Definition List** , from the **Process Definition Detail** view or from the **Process Instance** view. When you create a new Process Instance, a new window opens that requires you to provide information required by the process to be started. Once you provide the required information and click on the **Submit** button, the instance is created and the details of the process instance is displayed in the **Process Instance Details** on the right.

You can further manage the instance during runtime, monitor its execution, and work with the tasks the instance produces for users with the proper roles assigned.

Additionally, Business Central allows you to easily sort and filter a list of tasks for any given process. You can create custom filters that allow you to define queries by user, business attributes (such as amount, customer segmentation), Process ID, correlation key and so on.

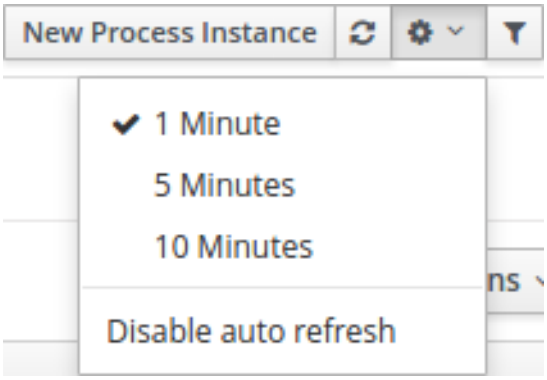
You can view the list of all the running process instances in the Process Instance List under **Process Management → Process Instances**. The process instances view comprises two main sections:

- Process Instance Lists
- Process Instance Details

The process instance list displays the process instances and this view is customizable. The customizable elements comprise columns that are displayed, number of rows displayed per page, name of the tabs, and title description. The views are available as tabs. When you click on a tab, the related parameters are applied to the data grid and the corresponding

process instances are listed. You can remove the default tabs and add your own with the required filter criteria. The Process Instances view also has features like auto refresh and restore default views. Auto refresh allows you to define how frequently the data grid refreshes. You can select one of the different values (1, 5 or 10 minutes), or disable this feature by clicking the **Disable** button:

Figure 11.4. Features in the Process Instances List View



Each row in the process instance list represents a running process instance from a particular process definition. Each execution is differentiated from all the others by the internal state of the information that the process is manipulating. In order to view this information, you can click on any one of the process instances and view the corresponding details in the **Process Instance Details** section. The **Process Instance Details** provides several tabs with the runtime information related to the process.

- The **Instance Details** tab: This gives you a quick overview about what is going on inside the process. It displays the current state of the instance and the current activity that is being executed.
- The **Process Variables** tab: This displays all the process variables that are being manipulated by the instance, with the exception of the variables that contain documents. You can move the mouse pointer over the **Value** field to view a full value of the process variable. Additionally, you can edit the process variable value and view its history.
- The **Documents** tab: This displays process documents if the process contains a variable of the type **org.jbpm.Document**. This enables easy access, download, and manipulation of the attached documents. You can not attach a new document to currently running instances using this view, but it can be achieved by Human task form in the tasks perspective.
- The **Logs** tab: This displays business and technical logs for the respective end users. In order to track a process through the logs, you can also open the Process Model that shows the completed activies in grey and the current activities highlighted in red.

11.2.2.1. Searching Process Instances by Partial Correlation Key

To create a filter to search by correlation key or partial correlation key, do the following:

1. On the top menu of the Business Central, go to **Process Management → Process Instances**.
2. In the list on the **Process Instances** tab, click  .

The **New Process Instance List** dialog box opens.

3. In the **New Process Instance List** dialog box:

   a. Provide the name and description for your search process instance list in the **Labels** tab.

   b. Click the **Filter** tab to create new query filter.

      i. Click **Add New**.

      ii. From the list of filter values, select **CORRELATIONKEY**. If you want to create a search filter using partial correlationKey, select the **like** query operator and provide the value as **`partial-correlation-key%`** where **`partial-correlation-key`** is the value you are searching for.

      iii. Click **Ok**.

   A new tab is created that displays your custom process instance list.

11.2.2.2. Searching Process Instances Based on Business Data

You can add process variables as columns in the process instance list in order to enable flexible filtering of definitions based on business data. To achieve this, do the following:

1. On the top menu of the Business Central, go to **Process Management** → **Process Instances**.

2. In the list on the **Process Instances** tab, click  . The **New Process Instance List** dialog box opens.

3. In the **New Process Instance List** dialog box, perform the following:

   a. Provide the name and description for your search process instance list in the **Labels** tab.

   b. Add a new query filter in the **Filter** tab:

      i. Click **Add New**.

      ii. From the list of filter values, select **processId** and **equalsTo**.

      iii. Provide a valid **`processId`** value and click **Ok**.

   A new tab is created that displays your custom process instance list in a tabular form. This new tab provides process instance variables (business data) as selectable columns. You can view the variables corresponding to each process instance in the table by enabling these columns, which are disabled by default.

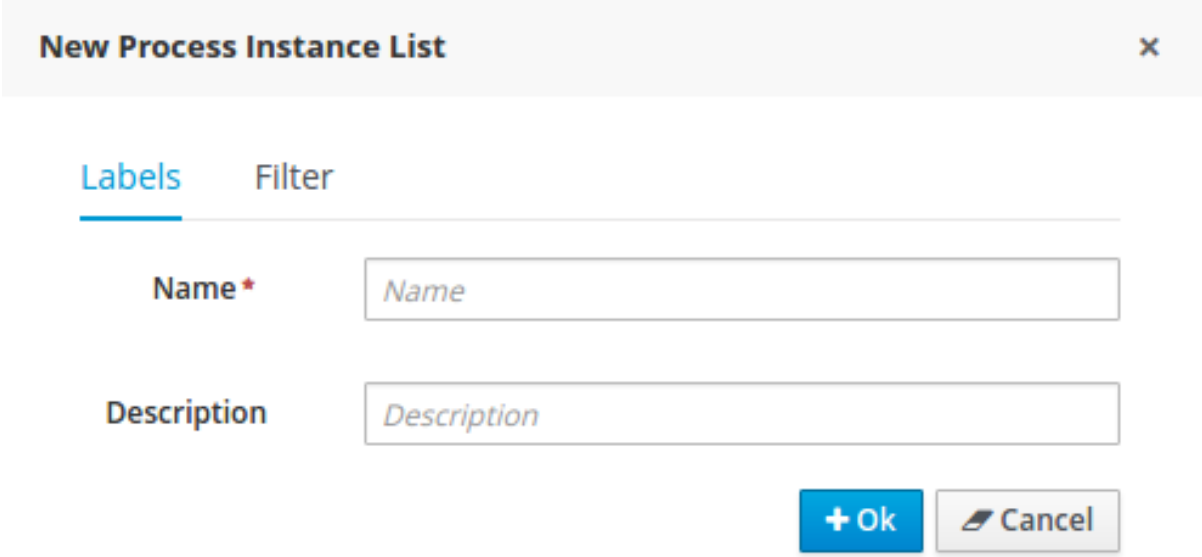
11.2.3. Creating a New Process Instance List

To create a custom process instance list, do the following:

1. On the top menu of the Business Central, go to **Process Management** → **Process Instances**.

2. In the list on the **Process Instances** tab, click the + button.
The following **New Process Instance List** dialog box opens:

Figure 11.5. New Process Instance List



3. In the **New Process Instance List** dialog box:

   a. Provide the name and description for your process instance list in the **Labels** tab.

   b. Click the **Filter** tab to create new query filter.

      i. Click **Add New**.

      ii. From the list of filter values, select the appropriate filter condition and its value. You can add more filters by clicking **Add New**.

      iii. Once you have specified all your filter conditions, click **Ok**.

   A new tab is created that displays your custom process instance list.

11.2.4. Aborting a Process instance

You can abort a running Process instance either using the provided API or from the Business Central.

Aborting a Process instance using API
To abort a Process instance using the Kie Session API, use the **`void abortProcessInstance(long processInstanceId)`** call on the parent Kie Session.

Aborting a Process instance from the Business Central
To abort a Process instance from the Business Central, do the following:

1. On the top menu of the Business Central, go to **Process Management → Process Instances**.

2. In the list on the **Process Instances** tab, locate the required Process instance and click the **Abort** button in the instance row.

11.3. SIGNALING PROCESS INSTANCE

You can signal a running process instance using either API or Business Central.

Signaling Process Instance Using API

To signal a process instance using the Kie Session API, use the **`void signalEvent(String type, Object event)`** call on the parent Kie Session. The call triggers all active signal event nodes waiting for that event type in the Kie Session. The runtime strategy determines the number of processes which receive the signal.

If you need to signal a specific process instance, use **`void signalEvent(String type, Object event, long processInstanceId)`**.

NOTE

If you use the **`Throwing Intermediate`** event of type **`Signal`**, the execution engine calls **`void signalEvent(String type, Object event)`**.

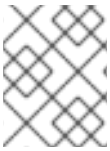
If you do not want the signal to be delivered to all the listening processes, replace the **`Throwing Intermediate`** event with a **`Script Task`**:

```
kcontext.getKieRuntime().signalEvent("signalRefId", data,
processInstanceId);
```

Signaling Process Instance from Business Central

To signal a process instance from Business Central, do the following:

1. Log into Business Central.
2. Click **Process Management** → **Process Instances**.
3. Locate the required process instance and click **Signal** in the instance row.
4. Fill the following fields:
   - **Signal Name**: corresponds to the **`SignalRef`** or **`MessageRef`** attributes of the signal. This field is required.
   - **Signal Data**: corresponds to data accompanying the signal. This field is optional.



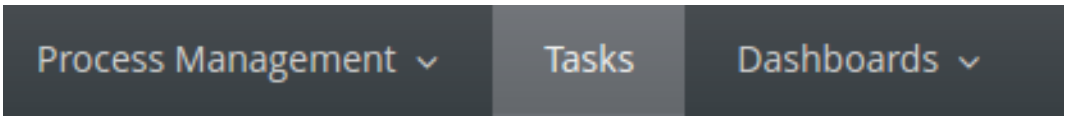
NOTE

You can also send a **`Message`** event to the process. To do so, add the **`Message-`** prefix in front of the **`MessageRef`** value.

11.4. TASK MANAGEMENT

The following sections describe the features provided by the options available under the Tasks menu in Business Central.

Figure 11.6. Task Management



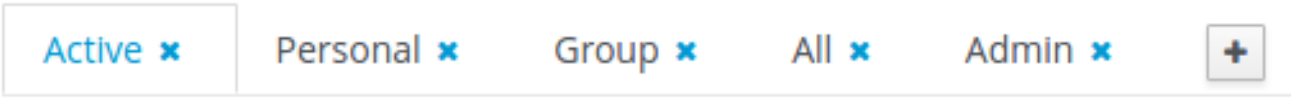
11.4.1. Tasks List

A User Task represents a piece of work the given user can claim and perform. User Tasks can be handled within the **Tasks** perspective of the Business Central: the view displays the Task List for the given user. You can think about it as a to-do item. The User Task appears in your list either because the User Task element generated the User Task as part of Process execution or because someone has created the User Task directly in the Business Central console.

A User Task can be assigned to a particular actor, multiple actors, or to a group of actors. If assigned to multiple actors or a group of actors, it is visible in the Task Lists of all the actors and any of the possible actors can claim the task and execute it. The moment the Task is claimed by one actor, it disappears from the Task List of other actors.

Task Client
User Tasks are displayed in the Tasks perspective, that are an implementation of a Task client, in the Business Central console: to display the Tasks perspective, click **Tasks** . You can filter out the Tasks based on their status using the following tabs:

Figure 11.7. Task Lists Tabs



- **Active**: Displays all the active tasks that you can work on. This includes personal and group tasks.
- **Personal**: Displays all your personal tasks.
- **Group**: Displays all the group tasks that need to be claimed by you in order to start working on them.
- **All**: Displays all the tasks. This also includes completed tasks but not the ones that belongs to a process that is already finished.
- **Admin**: Displays all the tasks for which you are the business administrator.

In addition to these, you can create custom filters to filter tasks based on the query parameters you define. For further information about custom tasks filters, see Section 11.4.2, “Creating Custom Tasks Filters”.

The **Tasks List** view is divided into two sections,**Task List** and **Task Details**. You can access the **Task Details** by clicking on a task row. You can modify the details (such the Due Date, the Priority or the task description) associated with a task. The Task Details section comprises the following tabs:

- **Work**: Displays basic details about the task and the task owner. You can click the **Claim** button to claim the task. To undo the claim process, click the**Release** button.

- **Details**: Displays information such as task description, status, and due date.
- **Process Context**: If the task is associated with a process, the information about it is shown here. You can also navigate to process instance details from here.
- **Assigments**: Displays the current owner of the task and allows you to delegate the task to another person or group.
- **Comments**: Displays comments added by task user(s). It allows you to delete an existing comment and add a new comment.
- **Logs**: Displays task logs containing task lifecycle events (such as task started, claimed, completed), updates made to task fields (such as task due date and priority).

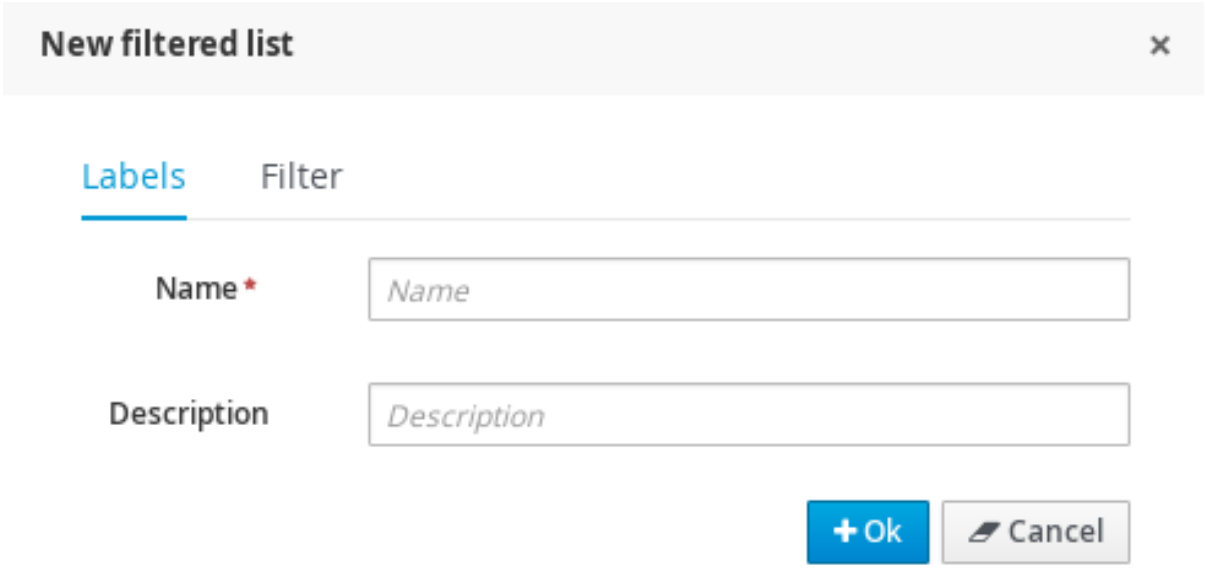
11.4.2. Creating Custom Tasks Filters

It is possible to create a custom task filter based on a provided query. The newly created filter is then added as a tab to the Tasks List.

The following procedure shows how to create a custom filter which allows you to view a list of tasks with a specified name.

Procedure: Filtering Tasks by Name

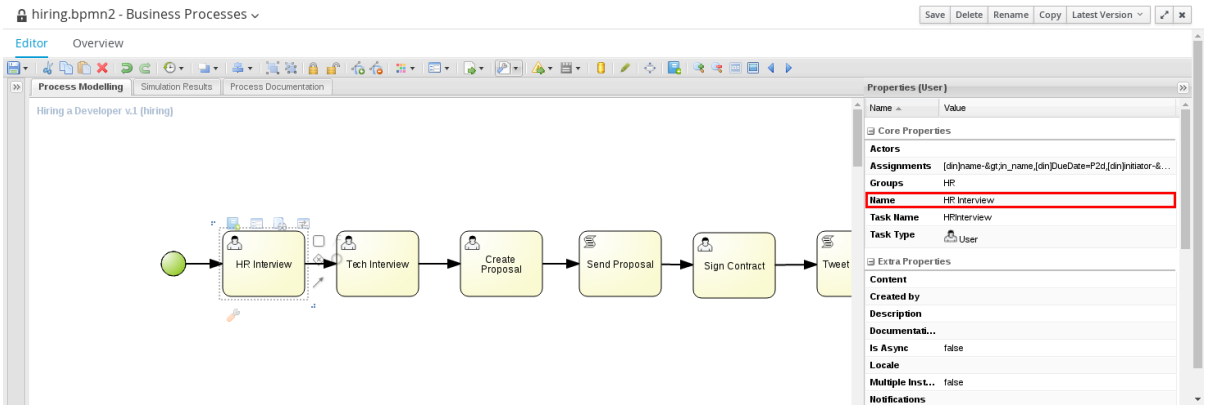
1. In the main menu of Business Central, click **Tasks**.
2. Click the + button on the right side of the Tasks Lists tabs.
   The **New filtered list** pop-up window is displayed.



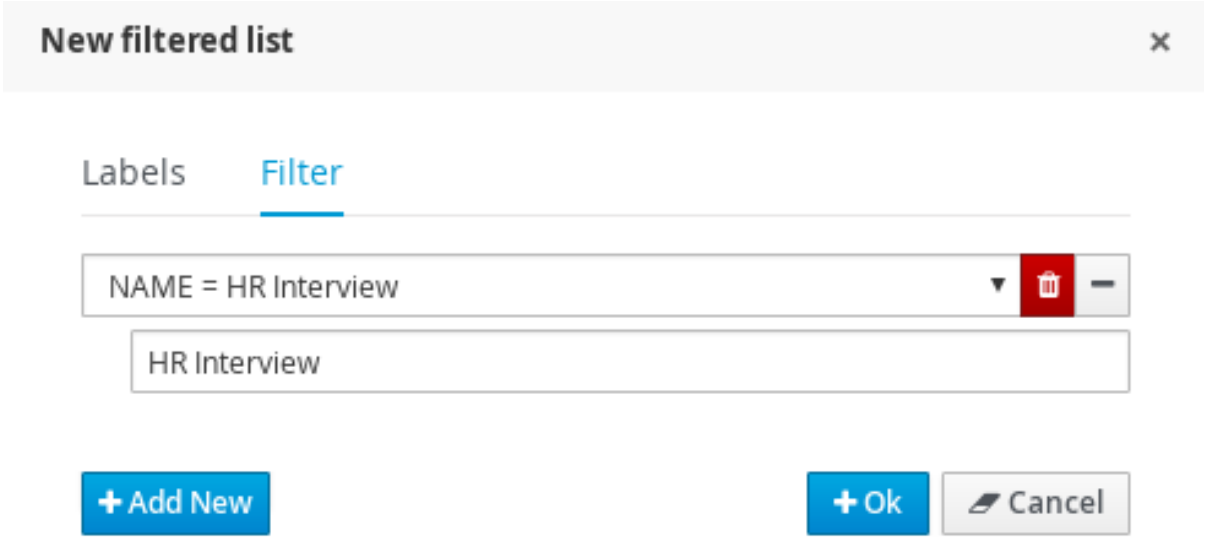
3. Fill in the **Name** (this is the label of the new Tasks Lists tab) and click **Filter**.
4. Click **Add New**.
5. In the **Select column** drop-down menu, choose **NAME**.
   The content of the drop-down menu changes to `NAME != value1`.
6. Click on the drop-down menu again and choose **equals to**.

The content of the drop-down menu changes to **`NAME = value1`**.

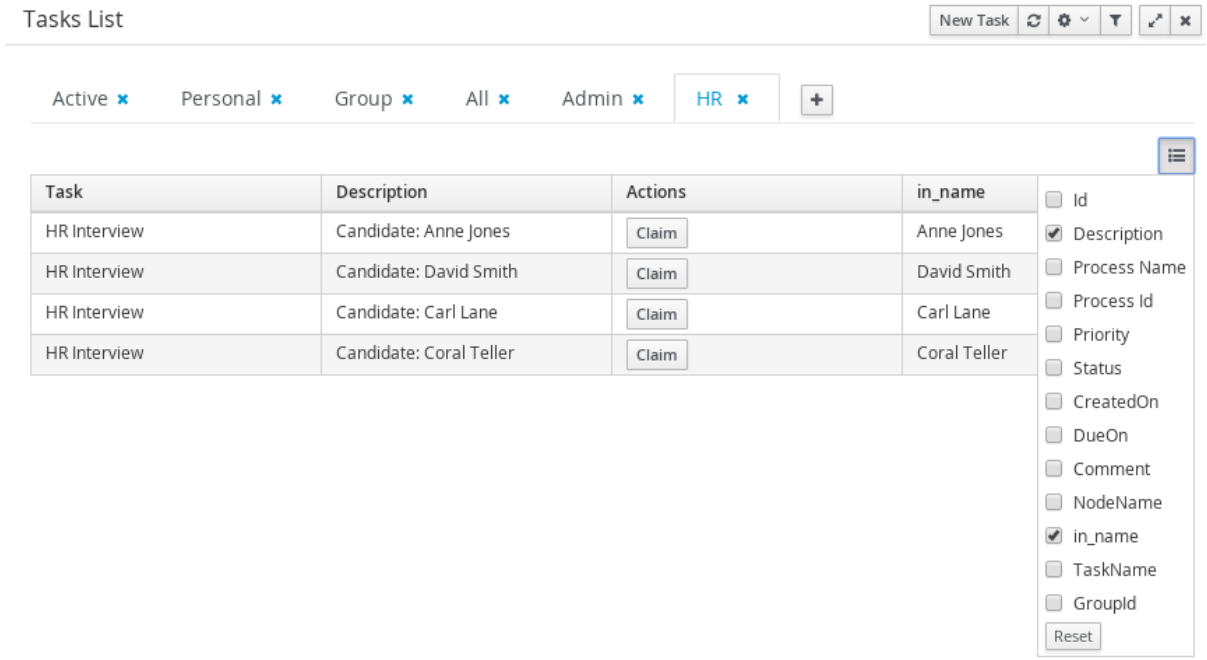
7. Rewrite the value of the text field to the name of the task you want to filter. Note that the name must match the value defined in the **Process Modelling** view of a business process. See the following screenshot:



8. Click **Ok**.



After the filter with a specified restriction is applied, variables associated with the task appear in the list of selectable columns.

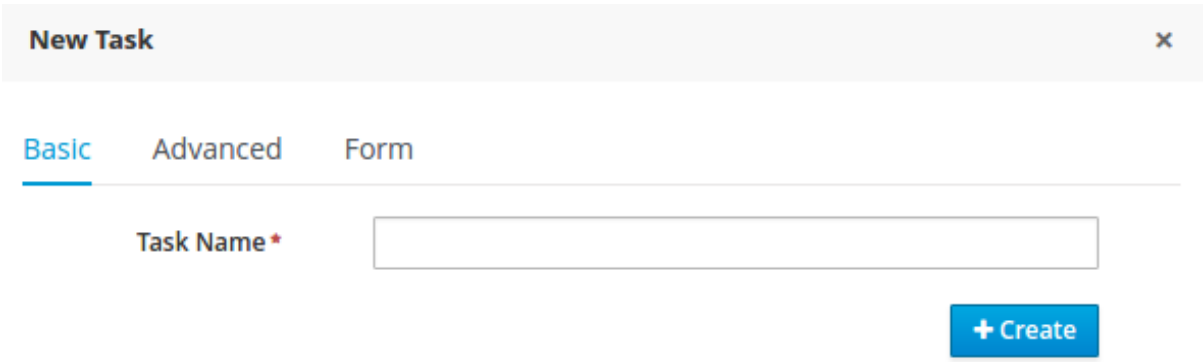


11.4.3. Creating a User Task

A user task can be created either by a User Task element executed as part of a process instance or directly in Business Central. To create a user task in Business Central, do the following:

1. On the top menu of the Business Central, click **Tasks**.
2. On the Tasks List tab, click **New Task** and define the task parameters. This opens a **New Task** window with the following tabs:

Figure 11.8. New Task Window



- **Basic** tab
  - **Task Name**: The task display name.
- **Advanced** tab
  - **Due On**: Add due date of the task.
  - **Priority**: Select task priority.
  - **Add User** button: Click to add more users. Note that a task cannot be created without a user or a group.

- **Add Group** button: Click to add more groups.
- **User**: Add the name of the person who executes the task.
- **Remove User** button: Click to remove the existing user.

- **Form** tab
  - **Task form deploymentId**: Select the deployment Id of the form from the list of available deployment Ids.
  - **Task form name**: Select the name of the associated task form from the list of available forms.
    If tasks are part of a Business Process, they have an associated form that collects data from you and propagates that to the business process for further usage. You can create forms for specific tasks using the Form Modeler. If there is no form provided for a task, a dynamic form is created based on the information that the task needs to handle. If you create a task as an ad-hoc task, which is not related with any process, there will be no such information to generate a form and only basic actions will be provided.

3. Click the **Create** button.

11.4.4. Task Variables as Expressions

You can refer and use the task variables in task properties as soon as you create a task. For example, once your task has been created, you can define a task name that refers to a **taskId**. Task variables are resolved at both task creation time and notification time, unlike process variables, which are resolved only at task creation time. The ability of using task variables while creating tasks minimizes your Java code, such as calling Red Hat JBoss BPM Suite APIs.

Task variables are available as task instances and you can get access to task information using the following expression:

```
${task.id}
```

You can use this expression in data input of user task from within the process definition.

For example, the following expression can be used for accessing the **processInstanceId** variable:

```
${task.taskData.processInstanceId}
```

CHAPTER 12. LOGGING

Logs with execution information are created based on events generated by the process engine during execution. It is the engine providing a generic mechanism listening to events. Information about the caught event can be extracted from these logs and then persisted in a data storage. To restrain the logged information, the log filtering mechanism is provided (for further information, see the *Red Hat JBoss BPM Suite Administration and Configuration Guide*).

CHAPTER 13. EXAMPLES

Red Hat JBoss BPM Suite comes with a project with assets examples to demonstrate the possible usage and capabilities of the product.

Also, the project contains Junit tests for each Element, which are simple working examples. These test processes can serve as simple examples. The entire list can be found in the src/test/resources folder for the jbpm-bpmn2 module. Note that each of the processes is accompanied by a junit test that tests the implementation of the construct.

PART V. BAM

CHAPTER 14. RED HAT JBOSS DASHBOARD BUILDER

Red Hat JBoss Dashboard Builder is a web-based dashboard application that provides Business Activity Monitoring (BAM) support, that is, visualization tools for monitored metrics (Key Performance Indicators, or KPIs) in real time. It comes integrated in the Business Central environment under the **Dashboards** menu.

It comes with a dashboard that requests information from the Red Hat JBoss BPM Suite Execution Engine and provides real-time information on its runtime data; however, you can also create custom dashboards over other data sources, which leaves the application relatively standalone.

What is Business Activity Monitoring?

Business Activity Monitoring (BAM) software helps to monitor business activities that take place on a computer system in real time. The software monitors particular metrics, such as the status of running processes, the number of invoices to be processed, processing times and other. It provides tools for visualization of the collected data (for example in graphs or tables).

14.1. BASIC CONCEPTS

Dashboard Builder can establish connections to external data sources such as databases. These connections are then used for creating data providers that obtain data from the data sources. Dashboard Builder is connected to the local JBoss BPM Suite engine by default and acquires the data for its JBoss BPM Suite Dashboard indicators (widgets with visualizations of the data available on the pages of the JBoss BPM Suite Dashboard workspace).

The data providers keep all the data. If operating over a database, the data provider uses an SQL query to obtain the data. Operating over a CSV file enables the data provider automatically obtain all the data from the file.

Data from the data providers can then be visualized as graphs or tables in indicators: special panels, that can be arranged on Dashboard Builder managed pages. Pages are contained within a workspace and can define permission access rights. The number of pages within a workspace is arbitrary. A set of pages that presents related information on similar KPIs is referred to as a dashboard.

14.2. ACCESSING DASHBOARD BUILDER

Dashboard Builder is accessible through Business Central and as a standalone application.

Within Business Central, Dashboard Builder can be accessed directly from the **Dashboards** menu at the top. The **Dashboards** menu contains two items:

- **Process & Task Dashboard** displays a pre-defined dashboard based on runtime data from the Execution Server. To learn more about Process & Task Dashboard, see Section 14.3, “Process & Task Dashboard”.
- **Business Dashboards** displays an environment in which you can create your own dashboards. This chapter contains procedures that provide instructions on how to create a custom dashboard.

Dashboard Builder can be accessed as a standalone application as well.

1. Start the server.

2. After the server has successfully started, navigate to **https://*HOST_NAME:PORT*/dashbuilder** in a web browser. For example https://localhost:8080/dashbuilder.

3. Log in with the user credentials created during installation.

After you log in, you are redirected to the **Showcase** workspace with the welcome page displayed.

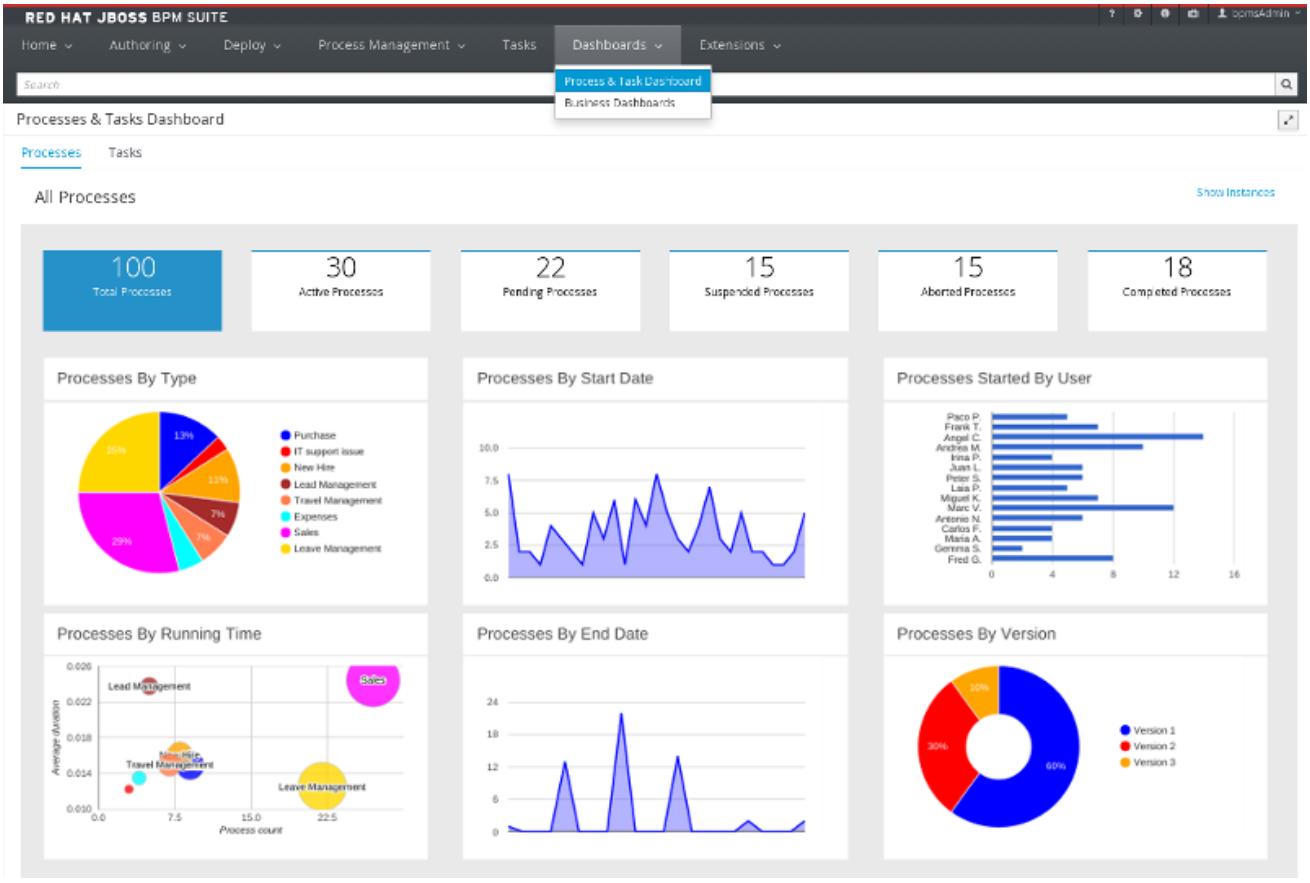
At the top of the page, you can change the workspace, the page, as well as find general configuration buttons. This interface area is common for all workspaces.

Dashboard area with variable content, a lateral menu on the left and the main dashboard area on the right, is located below the common interface area.

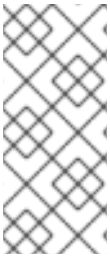
14.3. PROCESS & TASK DASHBOARD

The **Process & Task Dashboard** contains several performance indicators monitoring the jBPM Execution Engine. The data used by the dashboard comes from two tables of the database belonging to the engine: **processinstancelog** and **bamtasksummary**.

Figure 14.1. The Process & Task Dashboard Main Screen



Every time the information stored into the database is updated, the data becomes automatically available to the dashboard indicators.



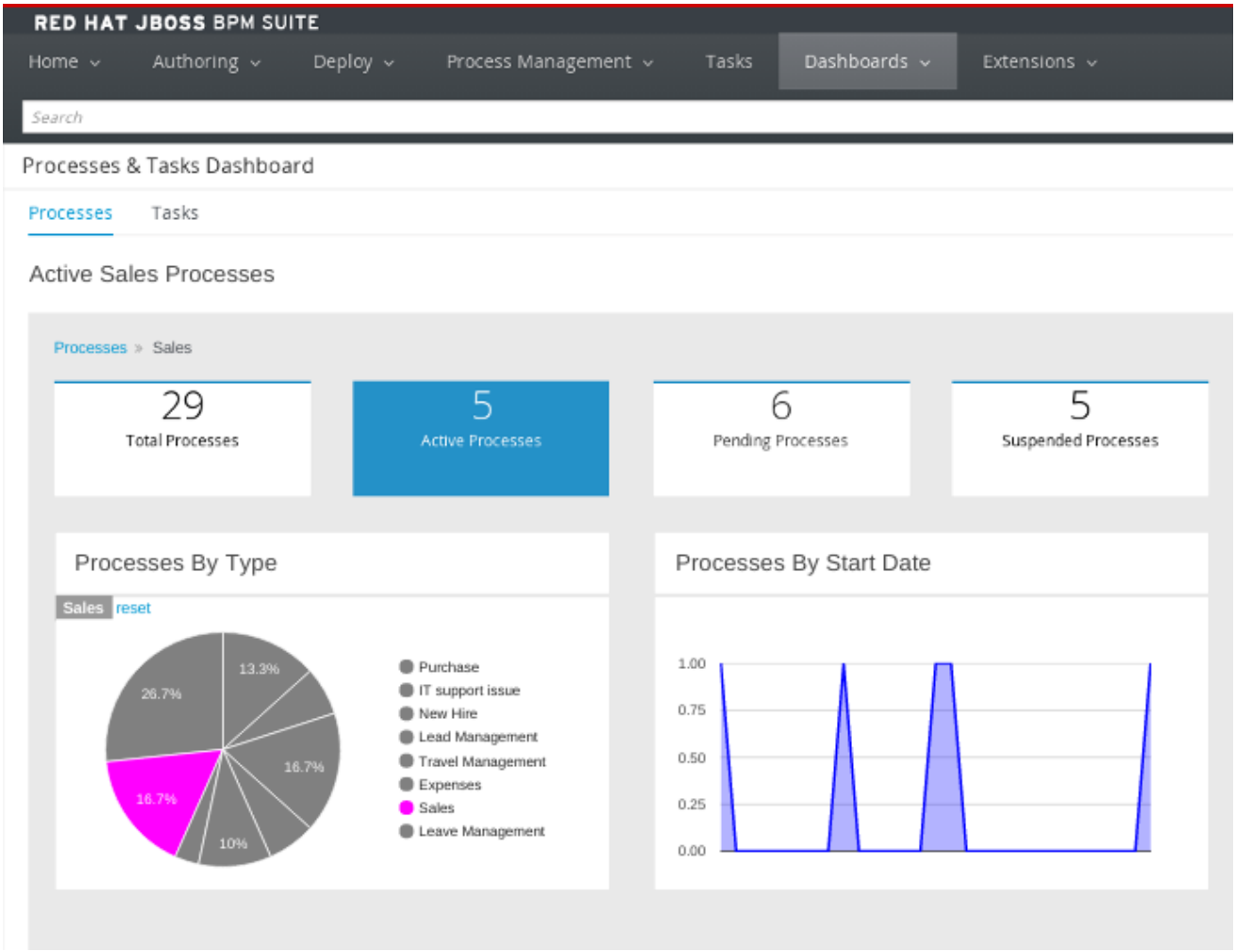
NOTE

All the metrics are generic and do not belong to any specific business process. However, it is possible to modify or extend the generic dashboard for your own use: the jBPM Process Dashboard can serve as a template for building a custom dashboard, which works with both data of the jBPM Engine and data coming from your own business domain.

At the top of the **Process & Task Dashboard** main screen, you can choose whether you want to view indicators related to **Processes** or **Tasks**.

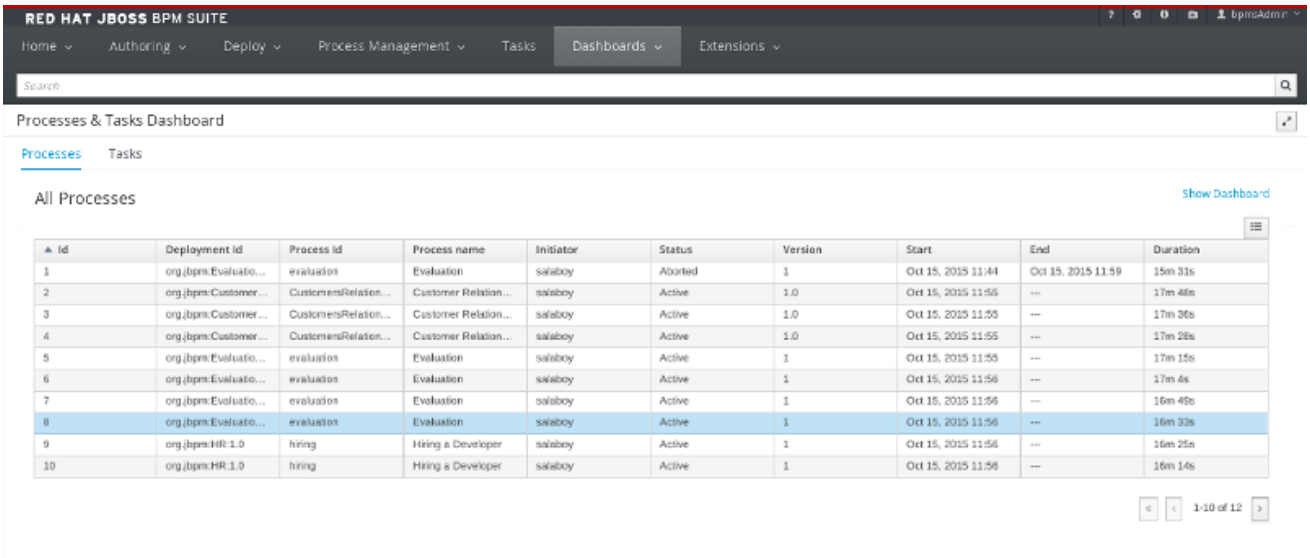
You can filter the data by clicking the charts, for example if you want to select a particular process or status. Every time a filter is applied, all the indicators are automatically updated and synchronized to show the selected criteria. The following picture shows an example dashboard with the **Sales** process and the **Active** status selected.

Figure 14.2. Active Sales Processes



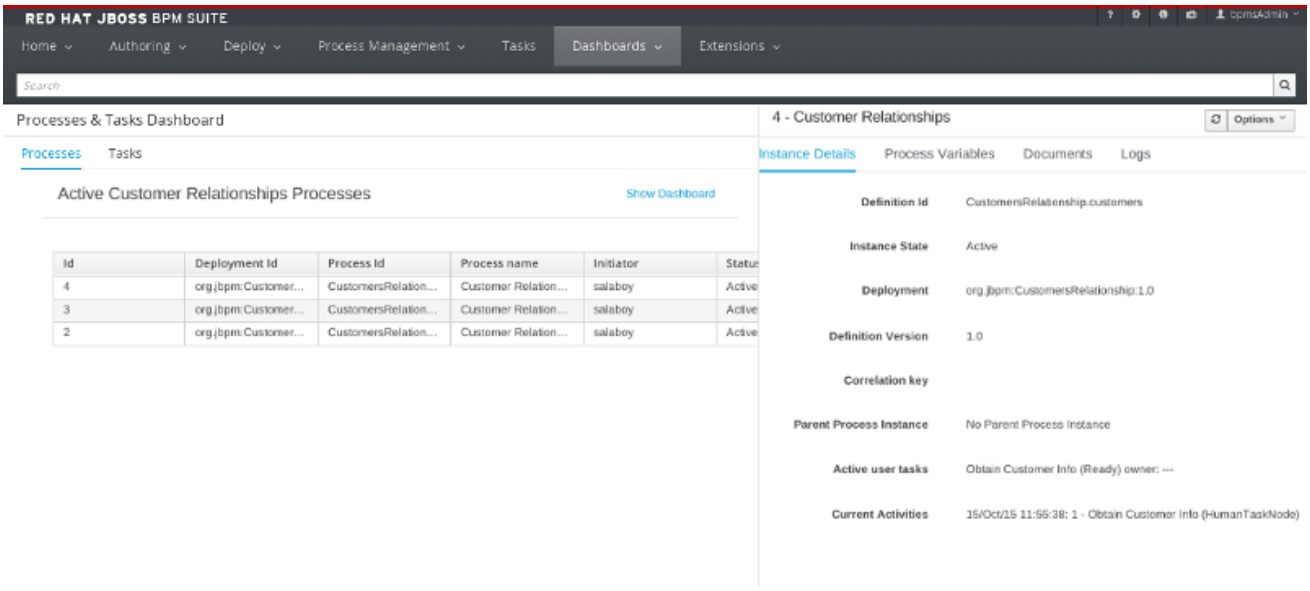
It is also possible to display a list of instances at any time by clicking the **Show Instances** link in the upper right hand corner of the screen. You can then switch to the original screen by clicking the **Show Dashboard** link.

Figure 14.3. Process Instances List



You can sort the instances by clicking any column header. Details about a particular instance are shown on the right side of the page after selecting a row. Note that the displayed details are not editable. If you want to manage a process instance, go to **Process Management → Process Instances** in Business Central.

Figure 14.4. Process Instance Details Panel



Tasks Dashboard

To view the **Tasks** dashboard, click the **Tasks** tab at the top of the screen. This dashboard provides the same features as introduced above, but related to the tasks only.

14.4. DATA SOURCES

Red Hat JBoss Dashboard Builder can be connected to an external database, either using the container's JNDI data source or connecting directly using the JDBC driver to access the database. Connections to databases can be configured in workspace *Showcase* on page *External Connections*. After you have established the connection to the database, you need to create a data provider that will collect the data from the database and allow you to visualize it as an indicator in the dashboard area of a page.

When connecting to CSV files to acquire data, the connection is established directly through the data provider.

Note that Red Hat JBoss Dashboard Builder makes use of its own local internal database to store its local data. This database is read-only for Dashboard Builder, but is accessible from outside.

14.4.1. Connecting to Data Sources

You can connect either to a JNDI data source, that is, a data source set up and accessible from the application container, or directly to the data source as a custom data source, if the application container has the correct JDBC driver deployed.

To connect to an external data source, do the following:

1. Make sure the data source is up and running and that the application server has access to the data source. (Check the driver, the login credentials, etc. In Red Hat JBoss EAP 6, you can do so in the Management Console under **Subsystems → Connector → Datasources**)

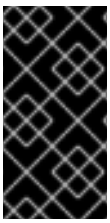
2. In Dashboard Builder, on the Tree Menu (by default located on the of the Showcase perspective), go to **Administration → External connections**.

3. On the displayed External Connection panel, click the **New DataSource**  button.

4. Select the data source type (JNDI or Custom DataSource) and provide the respective data source parameters below.

If you wish the jBPM Dashboard to use the new data source, modify also the respective data providers (jBPM Count Processes, jBPM Process Summary, jBPM Task Summary). Note that the data source needs to have access to jBPM history.

14.4.2. Security Considerations



IMPORTANT

When creating an external datasource using JBoss Dashboard Builder, it needs to use the local connection so that the user can be passed through. Otherwise, with a connection that uses <host>:<port>, every user would have the same virtual database (VDB) permissions.

14.4.3. Building a Dashboard for Large Volumes of Data

You can connect Red Hat JBoss Dashboard Builder to external databases and load data for generating reports and charts. Generally, if the volume of data is small (up to 2MB), Red Hat JBoss Dashboard Builder preloads the data into (local) memory and uses this data for report and chart generation. However, in case of large volumes of data, it is not possible to load the entire data set into the Dashboard Builder's local memory.

Based on the volume of data you are dealing with, you can choose to query the database to build a dashboard report in any one of the following strategies:

- The in-memory strategy
  The in-memory strategy is to create a data provider that loads all the required data

from the database by executing a single SQL query on the relevant tables, into the Dashboard Builder's memory. In this case, every indicator on the Dashboard Builder shares the same data set. When you use filters from the Dashboard Builder user interface to access specific data from this data set, the Dashboard Builder fetches the data from the internal memory and does not execute another SQL query again on the database. This strategy has a simple data retrieval logic as it deals with creating a single data provider. As all the data set properties are available to you at once, it allows you to configure KPIs faster. However, this approach is not suitable for large data sets as it would lead to poor performance.

- The native strategy
  The native approach is to create a data provider for every indicator in the Dashboard Builder and does not require loading all the data into the internal memory at once. So each time you use a filter from the Dashboard Builder user interface, the corresponding SQL queries get executed and fetches the required data from the database. So there is no data in the Dashboard Builder's internal memory. This strategy works best in case of large volumes of data, however it needs proper indexing on the database tables. Also, setting up data providers for multiple KPIs is complicated as compared to creating a single data provider in case of in-memory strategy.

Example

Let us consider a case when you want to create a stock exchange dashboard comprising the following charts and reports:

- Bar chart for Average price per company
- Area chart for Sales price evolution
- Pie chart for Companies per country
- Table report for Stock prices at closing date

For these charts and reports, let us assume that the Dashboard Builder accesses data from the following tables:

- Company: Comprising columns ID, NAME, and COUNTRY.
- Stock: Comprising columns ID, ID_COMPANY, PRICE_PER_SHARE, and CLOSING_DATE.

For the in-memory strategy of building a dashboard, the following SQL query fetches all the required data from these two tables:

```
SELECT C.NAME, C.COUNTRY, S.PRICE_PER_SHARE, S.CLOSING_DATE
  FROM COMPANY C JOIN STOCK S ON (C.ID=S.ID_COMPANY)
```

The output of this query is saved in the Dashboard Builder's local memory. The Dashboard accesses this data every time a filter is run.

On the other hand, if you are using the native strategy for huge volumes of data, an SQL query is executed on every filter request made by the Dashboard Builder and corresponding data is fetched from the database. In this case here is how each filter accesses the database:

- For the bar chart on *Average price per company*, the following SQL query is executed:

```
SELECT C.NAME, AVG(S.PRICE_PER_SHARE)
  FROM COMPANY C JOIN STOCK S ON (C.ID=S.ID_COMPANY)
  WHERE {sql_condition, optional, c.country, country}
  AND {sql_condition, optional, c.name, name}
  GROUP BY C.NAME
```

- For the area chart on *Sales price evolution*, the following SQL query is executed:

```
SELECT S.CLOSING_DATE, AVG(S.PRICE_PER_SHARE)
  FROM COMPANY C JOIN STOCK S ON (C.ID=S.ID_COMPANY)
  WHERE {sql_condition, optional, c.country, country}
  AND {sql_condition, optional, c.name, name}
  GROUP BY CLOSING_DATE
```

- For the pie chart on *Companies per country*, the following SQL query is executed:

```
SELECT COUNTRY, COUNT(ID)
  FROM COMPANY
  WHERE {sql_condition, optional, country, country}
  AND {sql_condition, optional, name, name}
  GROUP BY COUNTRY
```

- For the table report on *Stock prices at closing date*, the following SQL query is executed:

```
SELECT C.NAME, C.COUNTRY, S.PRICE_PER_SHARE, S.CLOSING_DATE
  FROM COMPANY C JOIN STOCK S ON (C.ID=S.ID_COMPANY)
  WHERE {sql_condition, optional, c.country, country}
  AND {sql_condition, optional, c.name, name}
```

For each of these queries, you need to create a separate SQL data provider.

In the examples above, each KPI delegates the filter and group by operations to the database through the **{sql_condition}** clauses. The signature of the**{sql_condition}** clause is the following:

```
{sql_condition, [optional | required], [db column], [filter property]}
```

Here,

- optional: This indicates that if there is no filter for the given property, then the condition is ignored.
- required: This indicates that if there is no filter for the given property, then the SQL returns no data.
- db column: This indicates the database column where the current filter is applied.
- filter property: This indicates the selected UI filter property.

When a filter occurs in the UI, the Dashboard Builder parses and injects all the SQL data providers referenced by the KPIs into these SQL statements. Every time a filter occurs in

the UI, the Dashboard Builder gets all the SQL data providers referenced by the KPIs and injects the current filter selections made by the user into these SQLs.

14.4.4. Data Providers

Data providers are entities that are configured to connect to a data source (a CSV file or database), collect the required data, and assign them the data type. You can think about them as database queries.

The collected data can be then visualized in indicators on pages, exported as XLS or CSV, etc.

14.4.4.1. Creating Data Providers

To create a new data provider, do the following:

1. In the Tree Menu (the panel in the lateral menu of the Showcase workspace), click **Administration → Data providers**.
2. In the **Data Providers** panel, click the **Create new data provider**  button.
3. In the updated **Data Providers** panel, select in the **Type** dropdown menu the type of the data provider depending on the source you want the data provider to operate on.
4. Define the data provider parameters:

   Data provider over a CSV file

   - Name: user-friendly name and its locale.
   - CSV file URL: the URL of the file (for example, **`file:///home/me/example.csv`**).
   - Data separator: the symbol used as separator in the CSV file (the default value is semicolon; if using comma as the separator sign, make sure to adapt the number format if applicable).
   - Quoting symbol: the symbol used for quotes (the default value is the double-quotes symbol; note that the symbol may vary depending on the locale).
   - Escaping symbol: the symbol used for escaping the following symbol in order to keep its literal value.
   - Date format: the date and time format.
   - Number format: the number format pattern as used in **`java.text.DecimalFormat`**.

   Data provider over a database (SQL query)

   - Name: user-friendly name and its locale
   - Data source: the data source to query (the default value is **`local`**, which allows you to query the Dashboard Builder database)

- Query: query that returns the required data

5. Click **Attempt data load** 

to verify the parameters are correct.

6. Click **Save**.

7. In the table with the detected data, define the data type and if necessary provide a user-friendly name for the data. Click **Save**.

The data provider can now be visualized in an indicator on a page of your choice.

14.4.5. Workspace

A workspace is a container for pages with panels or indicators.

By default, the Showcase and Red Hat JBoss BPM Suite Dashboard workspaces are available.

To switch between workspaces, select the required workspace in the Workspace drop-down box in the top panel on the left. To create a new workspace, click the **Create workspace** icon () in the top menu on the left. You can also edit the current workspace properties, delete the current workspace, and duplicate the current workspace using icons in the top panel.

Every workspace uses a particular skin and envelope, which define the workspace's graphical properties.

14.4.5.1. Creating Workspace

To create a new workspace, do the following:

1. Click the **Create workspace** button on the top menu.
The management console with the **Workspace** node expanded and workspace management area with workspace details on the right is displayed.

2. In the **Create workspace** table on the right, set the workspace parameters:

   - Name: workspace name and its locale
   - Title: workspace title and its locale
   - Skin: skin to be applied on the workspace resources
   - Envelope: envelope to be applied on the workspace resources

3. Click **Create workspace**.

4. Optionally, click the workspace name in the tree menu on the left and in the area with workspace properties on the right define additional workspace parameters:

   - URL: the workspace URL
   - User home search: the home page setting
   If set to **`Role assigned page`**, the home page as in the page permissions is

applied; that is, every role can have a different page displayed as its home page. If set to **`Current page`**, all users will use the current home page as their home page.

14.4.5.2. Pages

Pages are units that live in a workspace and provide space (dashboard) for panels. By default, you can display a page by selecting it in the Page dropdown menu in the top panel.

Every page is divided in two main parts: the lateral menu and the central part of the page. The parts are divided further (the exact division is visible when placing a new panel on a page). Note that the lateral menu allows you to insert panels only below each other, while in the central part of the page you can insert panels below each other as well as tab them.

A page also has a customizable header part and logo area.

14.4.5.2.1. Creating Pages

To create a new page, do the following:

1. Make sure you are in the correct workspace.
2. Next to the **Page** dropdown box 

in the top menu, click the **Create new page**  button.
3. The management console with the **Pages** node expanded and page management area with page details on the right is displayed.
4. In the **Create new page** table on the right, set the page parameters:
   - Name: page name and its locale
   - Parent page: parent page of the new page
   - Skin: skin to be applied on the page
   - Envelope: envelope to be applied on the page
   - Page layout: layout of the page
5. Click **Create new page**.
6. Optionally, click the page name in the tree menu on the left and in the area with workspace properties on the right define additional page parameters:
   - URL: the page URL
   - Visible page: visibility of the page
   - Spacing between regions and panels

14.4.5.2.2. Defining Page Permissions

Although users are usually authorized using the authorization method setup for the underlying application container (on Red Hat JBoss EAP, the **`other`** security domain by default), the Red Hat JBoss Dashboard Builder has its own role-based access control (RBAC)

management tool to facilitate permission management on an individual page or multiple pages.

To define permissions on a page or all workspace pages for a role, do the following:

1. On the top menu, click the **General configuration** button: the management console is displayed.
2. Under the **Workspace** node on the left, locate the page or the **Pages** node.
3. Under the page/pages node, click the **Page permissions** node.
4. In the **Page permissions** area on the right, delete previously defined permission definition if applicable and define the rights for the required role:
    a. In the **Permission assignation** table, locate the **Select role** dropdown menu and pick the respective role.
    b. In the **Actions** column of the table, enable or disable individual permissions.
5. Click **Save**.

14.4.5.3. Panels

A panel is a GUI widget, which can be placed on a page. There are three main types of panels:

Dashboard panels

are the primary BAM panels and include the following:

- Data provider manager: a panel with a list of available data providers and data provider management options
- Filter and Drill-down: a panel that displays all KPIs and their values to facilitate filtering in indicators on the given page defined over a data provider
- HTML Editor panel: a panel with static content
- Key Performance Indicator (indicator): a panel that visualizes the data of a data provider

Navigation panels

are panels that provide navigation functions and include the following:

- Breadcrumb: a panel with the full page hierarchy pointing to the current page
- Language menu: a panel with available locales (by default in the top center)
- Logout panel: a panel with the name of the currently logged-in user and the logout button
- Page menu custom: a panel with vertically arranged links to all pages in the workspace (the list of pages can be adjusted) and general controls for the HTML source of the page

- Page menu vertical: a panel with vertically arranged links to all pages in the workspace (the list of pages can be adjusted)
- Page menu horizontal: a panel with horizontally arranged links to all pages in the workspace (the list of pages can be adjusted)
- Tree menu: a panel with the links to essential features such as Administration, Home (on the Home page of the Showcase workspace displayed on the left, in the lateral menu)
- Workspace menu custom: a panel with links to available workspaces (the list of workspaces can be adjusted) and general controls for the HTML source of the workspace
- Workspace menu horizontal: a horizontal panel with links to available workspaces (the list of workspaces can be adjusted)
- Workspace menu vertical: a vertical panel with links to available workspaces (the list of workspaces can be adjusted)

System panels

are panels that provide access to system setting and administration facilities and include the following:

- Data source manager: a panel for management of external data sources
- Export dashboards: a panel export of dashboards
- Export/Import workspaces: a panel for exporting and importing of workspaces

14.4.5.3.1. Adding Panels

To add an existing panel to a page or to create a new panel, do the following:

1. Make sure the respective page is open (in the **Page** dropdown menu of the top menu select the page).
2. In the top menu, click the **Create a new panel in current page**  button.
3. In the displayed dialog box, expand the panel type you want to add (**Dashboard**, **Navigation**, or **System**) and click the panel you wish to add.
4. From the **Components** menu on the left, drag and drop the name of an existing panel instance or the **Create panel** item into the required location on the page. If inserting a new indicator, the Panel view with the graph settings will appear.Define the graph details and close the dialog.

   If adding an instance of an already existing indicator, you might not be able to use it, if it is linked to the KPIs on the particular original page. In such a case, create a new panel.
5. If applicable, edit the content of the newly added panel.

14.5. IMPORT AND EXPORT

Dashboard Builder provides the ability to export and import workspaces, KPIs, and data sources between two Dashboard Builder installations.

In general, it is possible to export the mentioned assets only using the Dashboard Builder web user interface. However, you can import the assets either in the web user interface, or by using the deployment scanner.

The deployment scanner is a subsystem of Red Hat JBoss Enterprise Application Platform that allows you to place the exported assets into the given folder inside the web application. Once the application has started, it scans the deployment folder and imports all the available assets. Note that the assets can be imported only during the deployment and *not* during the runtime.

14.5.1. Importing and Exporting Workspaces

By importing or exporting workspaces, you can move a set of pages between two Dashboard Builder installations. The procedure moves an envelope being currently used by the workspace, all the sections that compose the workspace and all the panels used in the workspace sections.

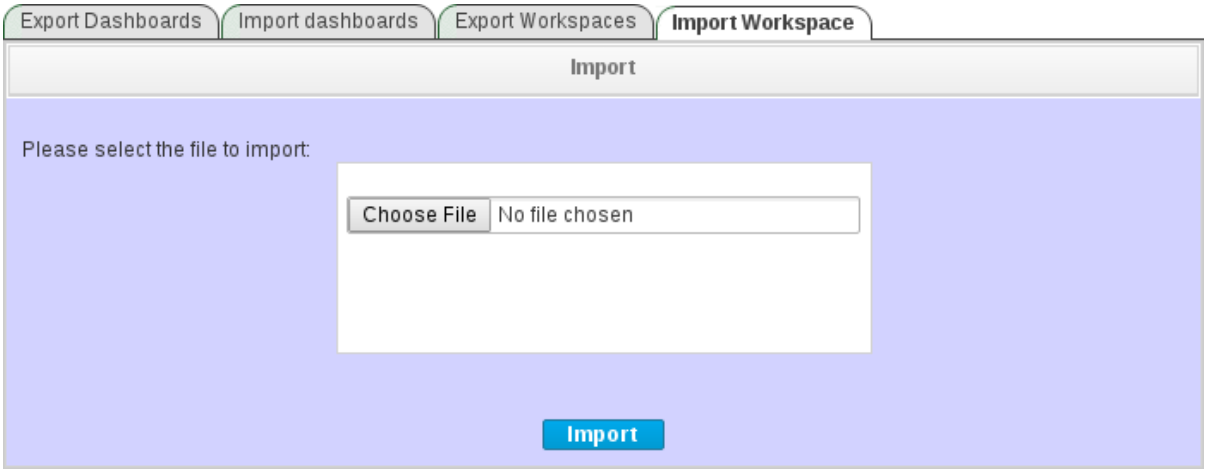
Procedure: Exporting Workspaces

1. In Business Central, go to **Dashboards** → **Business Dashboards**.
2. In the menu on the left, click **Administration** → **Import and export**.
3. Choose the **Export Workspaces** tab.
4. In the list of all existing workspaces that opens, select the ones you want to export and click **Export**.

5. Click **Download** to download a single XML file containing the workspace definitions.

Procedure: Importing Workspaces Using Web UI

1. In Business Central, go to **Dashboards** → **Business Dashboards**.
2. In the menu on the left, click **Administration** → **Import and export**.
3. Choose the **Import Workspace** tab.



4. Upload an XML file that contains one or more workspace definitions. Uploading a ZIP archive is supported as well for backward compatibility.

5. Click **Import**.

Procedure: Importing Workspaces Using Deployment Scanner

1. Make sure that the XML workspace definition file has the extension **`.workspace`**.

2. Move the file into the **`/jboss-eap-6.4/standalone/deployments/dashbuilder.war/WEB-INF/deployments`** directory. If the workspace already exists (there is a workspace with the same logic identifier), the file will be overwritten. Note that these two files do not have to have the same name in order to be replaced.

The workspaces are imported once during the application deployment.

14.5.2. Importing and Exporting KPIs

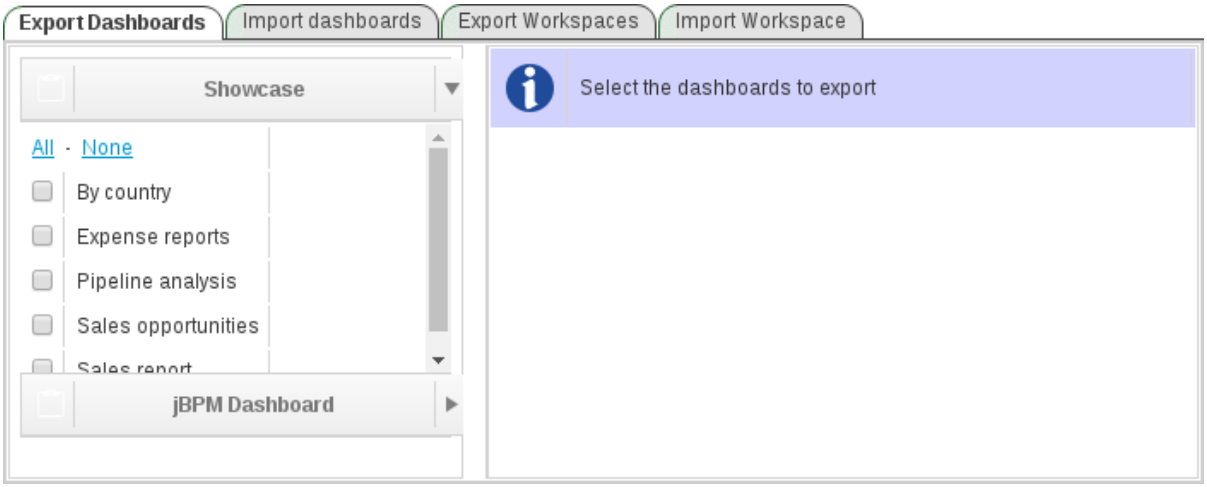
By importing and exporting KPIs, you can move key performance indicator definitions (the KPI type, its columns and display configuration) and their data providers between two Dashboard Builder installations.

Procedure: Exporting KPIs

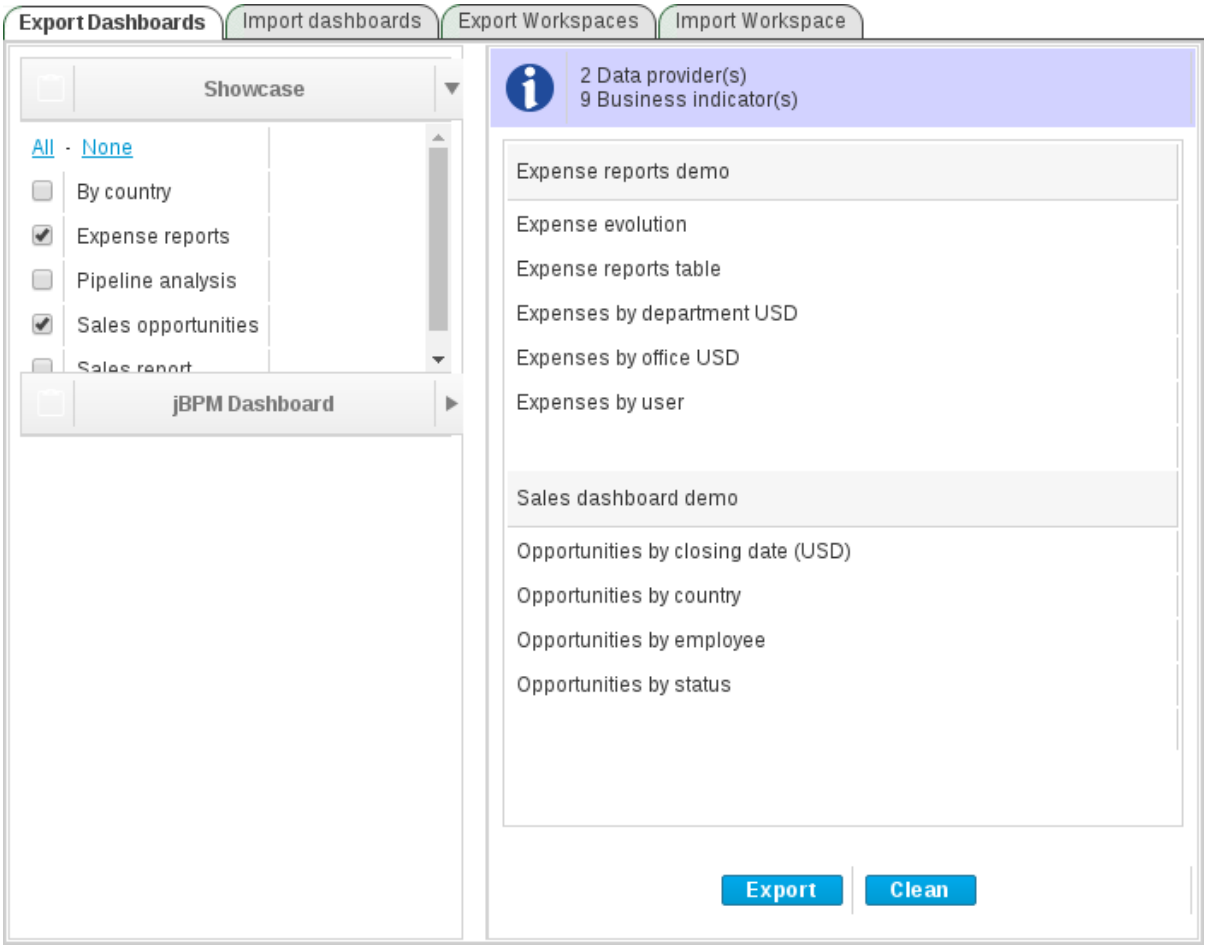
1. In Business Central, go to **Dashboards** → **Business Dashboards**.

2. In the menu on the left, click **Administration** → **Import and export**.

3. Choose the **Export Dashboards** tab.
A list of all KPI definitions in your application opens. You can export one or more of them into a single XML file.

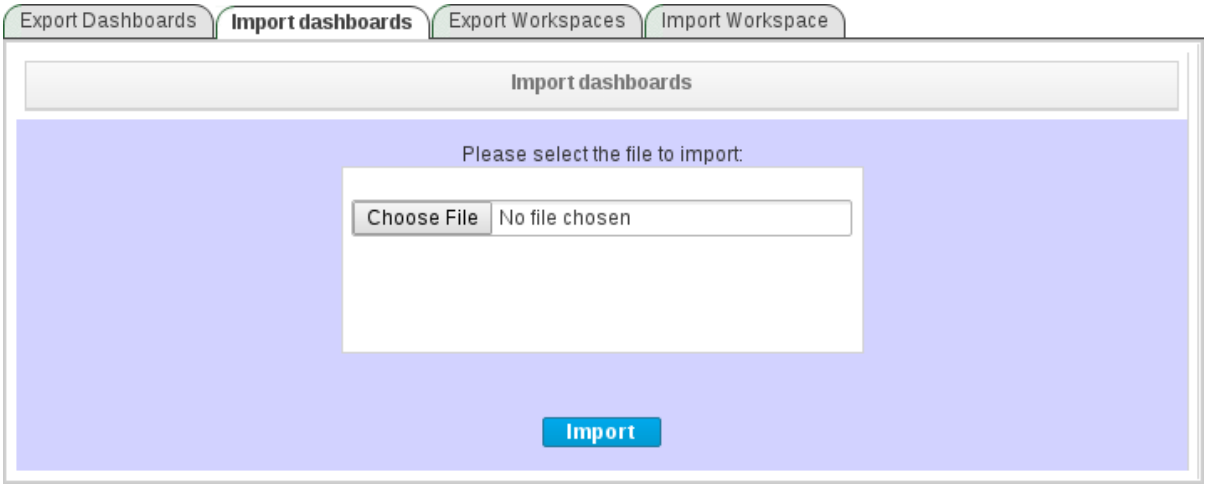


4. Select the KPIs you want to export and click **Export** .



Procedure: Importing KPIs Using Web UI

1. In Business Central, go to **Dashboards** → **Business Dashboards**.
2. In the menu on the left, click **Administration** → **Import and export**.
3. Choose the **Import dashboards** tab.



4. Upload an XML file that contains one or more KPI definitions and click **Import.**

Procedure: Importing KPIs Using Deployment Scanner

1. Make sure that the XML KPI definition file has the extension **`.kpis`**.
2. Move the file into the **`/jboss-eap-6.4/standalone/deployments/dashbuilder.war/WEB-INF/deployments`** directory. If the KPI or the data provider already exists (there is a file that contains a KPI or a data provider with the same logic identifier), the file will be overwritten. Note that these two files do not have to have the same name in order to be replaced.

The KPIs are imported once during the application deployment.

14.5.3. Importing Data Sources

NOTE

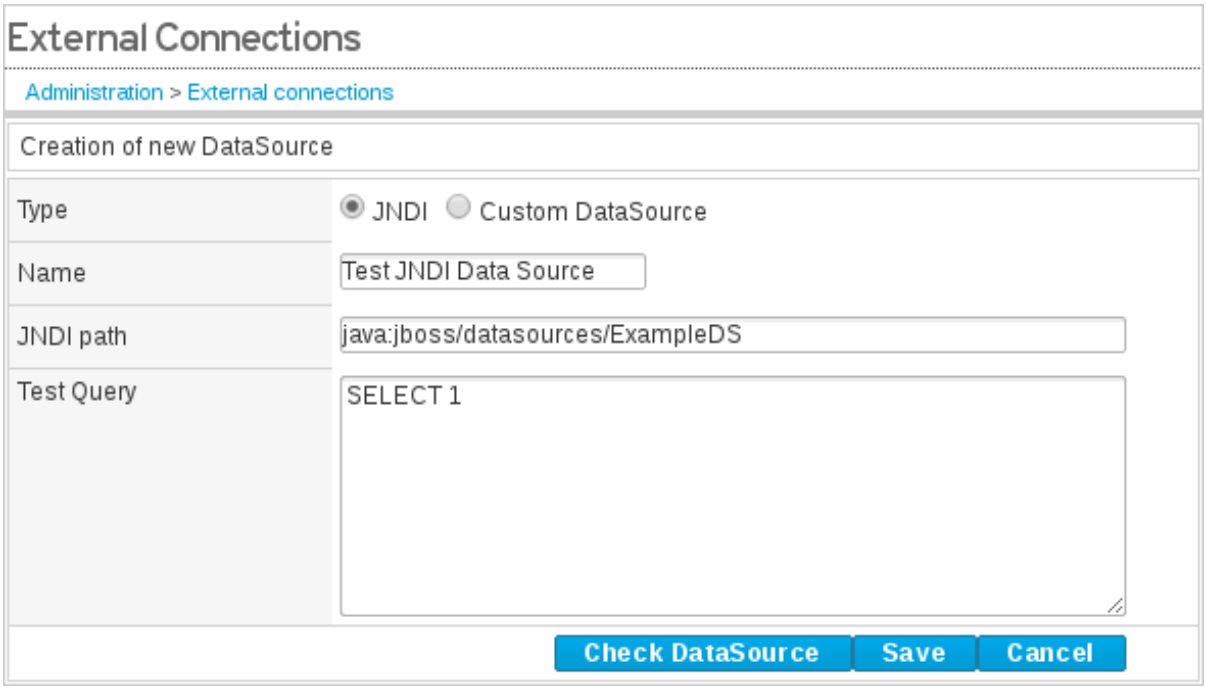
At present, it is *not* possible to export data sources.

By importing and exporting data sources, you can move one or more external data source connection configurations between two Dashboard Builder installations.

Since the data sources definitions consist of a very small number of attributes, it is possible to create them in your target environment manually by using the **External connections** panel.

Procedure: Creating Data Sources Manually Using Web UI

1. In Business Central, go to **Dashboards** → **Business Dashboards**.
2. In the menu on the left, click **Administration** → **External connections**.
3. Select the type of a new data source (either the JNDI or a Custom DataSource) and fill in the data source details.



4. Click **Check DataSource** to validate the details.
   If the validation ends up successfully, the following message appears:

   ```
   The DataSource is well configured.
   ```

5. Click **Save**.

Procedure: Importing Data Sources Using Deployment Scanner

1. Create the data sources definition files with the following supported properties:
   - common properties:
     - **type**: the type of the data source (**JNDI** or **JDBC**),
     - **name**: the data source name,
     - **testQuery**: a definition of a query used for testing the data source during the instantiation.
   - JNDI data source properties:
     - **jndiPath**: the data source bean path.

       Example 14.1. JNDI Data Source Descriptor

       ```
       type = JNDI
       ```

```
name = myCompanyDataSource
testQuery = SELECT count(*) FROM CUSTOMER
jndiPath = java:comp/env/jdbc/mycompany
```

- JDBC data source properties:
  - **`jdbcUrl`**: the JDBC URL for the connection,
  - **`driverClass`**: a fully qualified class name of the used driver,
  - **`user`**: the connection user name,
  - **`password`**: the connection password.

Example 14.2. JBDC Data Source Descriptor

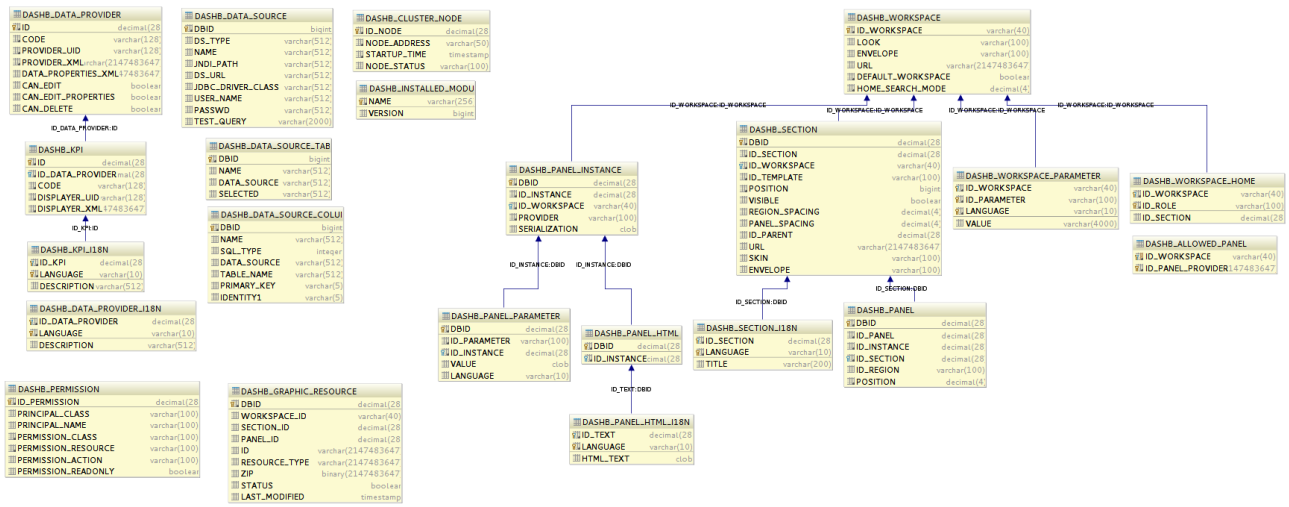
```
type = JDBC
name = myCompanyDataSource
testQuery = SELECT count(*) FROM CUSTOMER
jdbcUrl = jdbc:postgresql://mydomain.com:5432/mycompany
driverClass = org.postgresorg.postgresql.Driver
user = system
password = dba
```

2. Make sure that the data source definition file has the extension **`.datasource`**.
3. Move the file into the **`/jboss-eap-6.4/standalone/deployments/dashbuilder.war/WEB-INF/deployments`** directory. If the data source already exists (there is a file that contains a data source with the same logic identifier), the file will be overwritten. Note that these two files do not have to have the same name in order to be replaced.

The data sources are imported once during the application deployment.

14.6. DASHBOARD BUILDER DATA MODEL

The following image illustrates the Dashboard Builder data model:



NOTE

Dashboard Builder data model stores only metadata, *not* actual runtime data.

Table 14.1. Dashboard Builder Data Model

Table	Attributes	Description
dashb_data_source	**dbid**, **ds_type**, **name**, **jndi_path**, **ds_url**, **jdbc_driver_class**, **user_name**, **passwd**, **test_query**	Stores data source instances, either JNDI or JDBC.
dashb_data_source_table	**dbid**, **name**, **data_source**, **selected**	Currently not used. Stores a set of tables available for a given data source.
dashb_data_source_column	**dbid**, **name**, **sql_type**, **data_source**, **table_name**, **primary_key**, **identity1**	Currently not used. Stores a set of columns within a table.
dashb_permission	**id_permission**, **principal_class**, **principal_name**, **permission_class**, **permission_resource**, **permission_action**, **permission_readonly**	Stores permissions for different user interface resources (workspaces, pages, panels, and graphic resources).
dashb_graphic_resource	**dbid**, **workspace_id**, **section_id**, **panel_id**, **id**, **resource_type**, **zip**, **status**, **last_modified**	Stores graphic resource definitions (envelopes, layouts, and skins).
dashb_workspace	**id_workspace**, **look**, **envelope**, **url**, **default_workspace**, **home_search_mode**	Stores workspace instances.
dashb_workspace_home	**id_workspace**, **id_role**, **id_section**	Stores a home page for each role.
dashb_workspace_parameter	**id_workspace**, **id_parameter**, **language**, **value**	Stores workspace-related parameters.
dashb_allowed_panel	**id_workspace**, **id_panel_provider**	Stores a set of panel types a workspace can use.

Table	Attributes	Description
dashb_section	**dbid**, **id_section**, **id_workspace**, **id_template**, **position**, **visible**, **region_spacing**, **panel_spacing**, **id_parent**, **url**, **skin**, **envelope**	Refers to the **dashb_workspace** table.
dashb_section_i18n	**id_section**, **language**, **title**	Stores information for internationalization and localization.
dashb_panel_instance	**dbid**, **id_instance**, **id_workspace**, **provider**, **serialization**	Stores reusable panel instances. It is *not* tied to any specific page.
dashb_panel	**dbid**, **id_panel**, **id_instance**, **id_section**, **id_region**, **position**	Stores page panels. Refers to the **dashb_panel_instance** and **dashb_section** tables. It *is* tied to a particular page and layout region.
dashb_panel_parameter	**dbid**, **id_parameter**, **id_instance**, **value**, **language**	Stores page panels and *is* tied to a particular page and layout region.
dashb_panel_html	**dbid**, **id_instance**	Stores an HTML panel definition.
dashb_panel_html_i18n	**id_text**, **language**, **html_text**	Stores information for internationalization and localization.
dashb_data_provider	**id**, **code**, **provider_uid**, **provider_xml**, **data_properties_xml**, **can_edit**, **can_edit_properties**, **can_delete**	Stores data provider definitions (SQL and CSV).
dashb_data_provider_i18n	**id_data_provider**, **language**, **description**	Stores information for internationalization and localization.
dashb_kpi	**id**, **id_data_provider**, **code**, **displayer_uid**, **displayer_xml**	Stores all types of KPI definitions (pie, bar, line, and table).

Table	Attributes	Description
dashb_kpi_i18n	**id_kpi**, **language**, **description**	Stores information for internationalization and localization.
dashb_installed_module	**name**, **version**	Stores installed or imported modules used for automatic importing of assets.
dashb_cluster_node	**id_node**, **node_address**, **startup_time**, **node_status**	Stores running nodes and is needed for cluster setups.

CHAPTER 15. MANAGEMENT CONSOLE

Click **General Configuration** at the upper right hand corner of the standalone Dashbuilder application to access the management console.

The management console is inaccessible through the **Process & Task Dashboard** in Business Central.

The management console page contains a tree menu with the main administration resources on the left:

- Workspaces tree with individual workspaces and their pages (general item settings are displayed on the right)
- Graphic resources tree with options for upload of new graphic resources and management of the existing ones
- General permissions with access roles definitions and access permission management

To switch back to the last workspace page, click **Workspace**  in the upper left corner.

CHAPTER 16. GRAPHIC RESOURCES

Red Hat JBoss Dashboard Builder uses the following components to define the environment appearance and thus separate the presentation resources from content and data:

- Skins define a set of style sheets, images, and icons
- Region layouts define layouts of regions for pages
- Envelopes define an HTML template used as page frames

Graphic Resources Definitions

All graphics components are deployed as zip files as part of the Red Hat JBoss Dashboard Builder in the **`$DEPLOYMENT_LOCATION/dashbuilder.war/WEB-INF/etc/`** directory.

Every component definition contains the following:

- properties file that defines the name of the component for individual supported locales, the name of the css file to be applied on the component, and mapping of file to individual component elements
- JSP, HTML, CSS files, and image and icon resources referenced from the properties file

When creating custom graphic resources, it is recommended to download one of the existing components and modify it as necessary. This will prevent unnecessary mistakes in your definition.

16.1. WORKING WITH GRAPHIC RESOURCES

1. On the top menu, click the **General configuration** button.
2. Under the **Graphic resources** node on the left, click the component type you want to work with (**Skins** , **Layouts** , **Envelopers**). The window on the right will display the content relevant for the given component type.
3. On the right, you can now do the following:
    a. Upload a new component: you need to provide a unique ID for the component and the resource zip file. Then click **Add**.
    b. Download a component definition or preview the component: in the table below the Add view, click the respective icon in the **Actions** column.

APPENDIX A. PROCESS ELEMENTS

A Process Element is a node of the Process definition. The term covers the nodes with execution semantics as well as those without.

A.1. PROCESS

A Process is a named element defined in a Process Definition. It exists in a Knowledge Base and is identified by its ID.

A Process represents a namespace and serves as a container for a set of modeling elements: it contains elements that specify the execution workflow of a Business Process or its part using Flow objects and Flows. Every Process must contain at least one Start Event and one End Event.

A Process is accompanied by its BPMN Diagram, which is also part of the Process Definition and defines how the Process execution workflow is depicted when visualized, for example in the Process Designer.

Apart from the execution workflow and Process attributes, a Process can define Process variables, which store Process data during runtime. For more information on Process variables, see Section 4.9, “Variables”.

Runtime

During runtime, a Process serves as a blueprint for a Process instance (the concept of class and its object in OOP). A Process instance lives in a Session that may contain multiple Process instances. This allows the instances to share data, for example, using Globals that live in the Session instance, not in the Process instance. Every Process instance has its own context and ID.

Knowledge Runtime, called **kcontext**, holds all the Process runtime data. You can call it in your code, for example, in Action scripts, to obtain or modify the runtime data:

- Getting the currently executed Element instance so as to query further Element data, such as its name and type, or cancel the Element instance.

  Example A.1. Getting the currently executed Element

  ```
  NodeInstance element = kcontext.getNodeInstance();
  String name = element.getNodeName();
  ```

- Getting the currently executed Process instance so as to query further Process instance data, such as, its name, ID, or abort or send an event, such as a Signal.

  Example A.2. Getting the currently executed Process and sending it a Signal event

  ```
  ProcessInstance proc = kcontext.getProcessInstance();
  proc.signalEvent( type, eventObject );
  ```

- Getting and setting the values of variables

- Execute calls on the Knowledge runtime, for example, start Process instances, insert data, etc.

A Process instance goes through the following life cycle:

1. The **`createProcessInstance`** method is called on a Process: a new process instance based on a Process is created and Process variables are initialized . The process instance is in status **`CREATED`**.

2. The **`start()`** method is called on the ProcessInstance: the execution of the Process instance is triggered (the token on the Start Event is generated). If the Process was instantiated manually, the token is generated only on its None Start Event. If it is instantiated using another mechanism, such as Signal, Message, or Error, the token is generated on the Start Event of the respective type that is defined to handle the particular object. The process instance becomes **`ACTIVE`**.

3. Once there is no token in the flow (tokens are consumed by End Events and destroyed by Terminating Events), the Process instance is finished and becomes **`CANCELED`**.

The runtime state of a Process instance can be made persistent, for example, in a database. This allows to restore the state of execution in case of environment failure, or to temporarily remove running instances from memory and restore them later. By default, process instances are not made persistent. For more information on persistence see the *Red Hat JBoss BPM Suite Administration and Configuration Guide*

Properties

ID

Process ID defined as a String unique in the parent Knowledge Base.
Example value: **`org.jboss.exampleProcess`**.

It is recommended to use the ID form ***`<PACKAGENAME>.<PROCESSNAME>.<VERSION>`***.

Name

Process display name.

Version

Process version.

Package

Parent package to which the process belongs (that is process namespace).
The package attribute contains the location of the modeled process in form of a String value.

Target Namespace

BPMN2 xsd location.

Executable

Type of the process as concerns its executability.
Possible values: **`true`**, **`false`**.

Imports

Imported process.

Documentation

Documentation is a generic element attribute that can contain element description. It has no impact on runtime.

AdHoc

Boolean property defining whether a process is an ad-hoc process.
If set to **`true`**, the flow of the process execution is controlled exclusively by a human user.

Executable

Boolean property defining whether a process is intended for execution. If set to **`false`**, the process cannot be instantiated.

Globals

Set of global variables visible for other processes to allow data sharing.

A.2. EVENTS MECHANISM

During process execution, the Process Engine makes sure that all the relevant tasks are executed according to the Process definition, the underlying work items, and other resources. However, a Process instance often needs to react to a particular event it was not directly requesting. Such events can be created and caught by the Intermediate Event elements (see the *Red Hat JBoss BPM Suite User Guide*). Explicitly representing these events in a Process allows the author to specify how the particular Event should be handled.

An Event must specify the type of event it should handle. It can also define the name of a variable, which will store the data associated with the event. This allows subsequent elements in the Process to access the event data and take appropriate action based on this data.

An event can be signaled to a running instance of a process in a number of ways:

- Internal event: Any action inside a process (for example, the action of an action node, or an on-entry or on-exit action of some node) can signal the occurrence of an internal event to the surrounding Process instance.

  Example A.3. Schema of the call sending an event to the Process instance

  ```
  kcontext.getProcessInstance().signalEvent(type, eventData);
  ```

- External event: A process instance can be notified of an event from outside

  Example A.4. Schema of the call notifying a Process instance about an external event

  ```
  processInstance.signalEvent(type, eventData);
  ```

- External event using event correlation: Instead of notifying a Process instance directly, you can notify the entire Session and let the engine determine which Process instances might be interested in the event using event correlation. Event correlation is determined based on the event type. A Process instance that contains

an Event element listening to external events of some type is notified whenever such an event occurs. To signal such an event to the process engine, write code such as:

Example A.5. Schema of the call notifying a Session about an external event

```
ksession.signalEvent(type, eventData);
```

Events can also be used to start a process. Whenever a Message Start Event defines an event trigger of a specific type, a new process instance starts every time that type of event is signalled to the process engine.

This mechanism is used for implementation of the Intermediate Events and can be used to define custom Events if necessary.

A.3. COLLABORATION MECHANISMS

Elements with execution semantics make use of general collaboration mechanisms. Different Elements allow you to access and use the mechanism in a different way; for example, there is a mechanism called signalling: a Signal is sent by a Throw Signal Intermediate Event Element and received by a Catch Signal Intermediate Event (two Elements with execution semantics make use of the same Signal mechanism).

Collaboration mechanism includes the following:

Signals

General, mainly inter-process instance communication

Messages

Messages are used to communicate within the process and between process instances. Messages are implemented as signals which makes them scoped only for a given **`KSession`** instance.
For external system interaction send and receive task should be used with proper handler implementation.

Escalations

Mainly signalling between processes to trigger escalation handling.

Errors

Mainly inter-process signalling of escalation to trigger escalation handling.

All the events are managed by the signaling mechanism. To distinguish individual objects of individual mechanism the signal use different signal codes or names.

A.3.1. Signals

Signals in Red Hat JBoss BPM Suite correspond to the Signal Event in BPMN 2.0, and are the most flexible of the listed mechanisms. Signals can be consumed by an arbitrary number of elements both within its process instance and outside of it. Signals can also be consumed by any element in any session within or cross the current deployment, depending on the scope of the event that throws the signal.

A.3.1.1. Triggering Signals

The following Throw Events trigger signals:

Types of Signal Throw Events

- Intermediate Throw Event
- End Throw Event

Every signal defines its signal reference (**`SignalRef`**), which is unique in the respective session.

A signal can have one of the following scopes, which restricts its propagation to the selected elements:

Signal Scopes

Default (ksession)

Signal only propagates to elements within the given ksession. The behavior varies depending on what runtime strategy is used:

- **`Singleton`**: All instances available for the ksession are signalled.
- **`Per Request`**: Signal propagates within the currently processed process instance and process instances with Start Signal Events.
- **`Per Process Instance`**: Same as per request.

Process Instance

The narrowest possible scope, restricting the propagation of the signal to the given process instance only. No catch events outside that process instance will be able to consume the signal.

Project

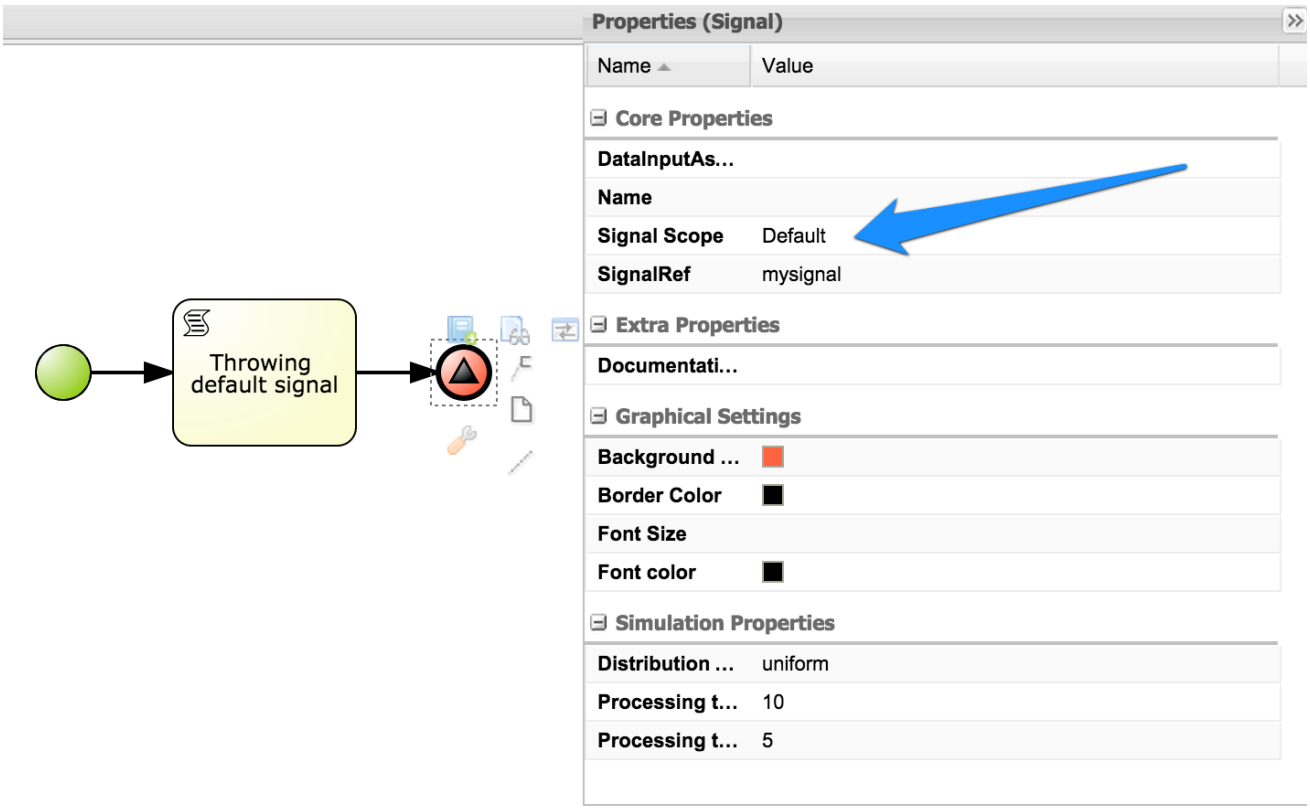
Signals all active process instances of given deployment and start signal events (regardless of the strategy).

External

Allows to signal elements both within the Project and across deployments. The **`External`** scope requires further setup.

To select the scope in the Process Designer, click **Signal Scope** under **Core Properties** of a Signal Throw Event:

Figure A.1. Selecting Signal Scope (Default)



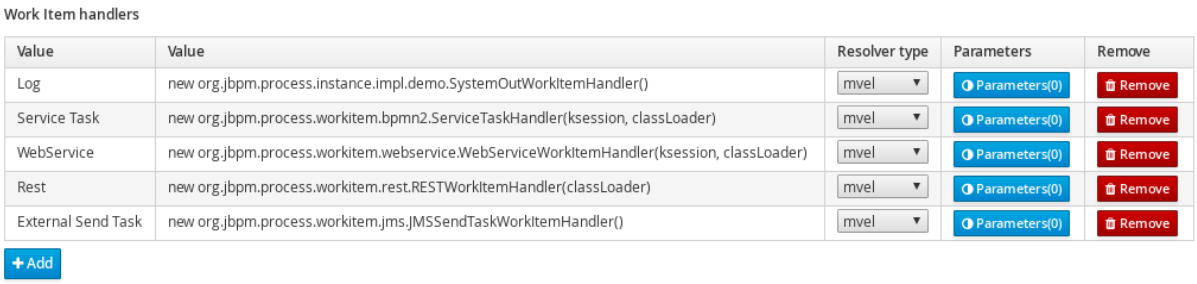
Signalling External Deployments

When creating an external signal event, you need to specify the Work Item handler for External Send Task manually. Use **`org.jbpm.process.workitem.jms.JMSSendTaskWorkItemHandler`**, which is shipped with Red Hat JBoss BPM Suite. It is not registered by default because each supported application server handles JMS differently, mainly due to different JNDI names for queues and connection factories.

Procedure: Registering External Send Task Handler

1. In Business Central, open your project in the Project Editor and click **Project Settings: Project General Settings → Deployment descriptor**.
2. Find the list of **Work Item handlers** and click **Add**.
3. Provide these values:
   - **Name**: `External Send Task`
   - **Value**: `new org.jbpm.process.workitem.jms.JMSSendTaskWorkItemHandler()`
   - **Resolver type**: `mvel`

Figure A.2. Registered External Send Task Handler



This will generate a corresponding entry in the **kie-deployment-descriptor.xml** file.

The **JMSSendTaskWorkItemHandler** handler has five different constructors. The parameterless constructor used in the procedure above has two default values:

- Connection factory: **java:/JmsXA**
- Destination queue: **queue/KIE.SIGNAL**

You can specify custom values using one of the following constructors instead:

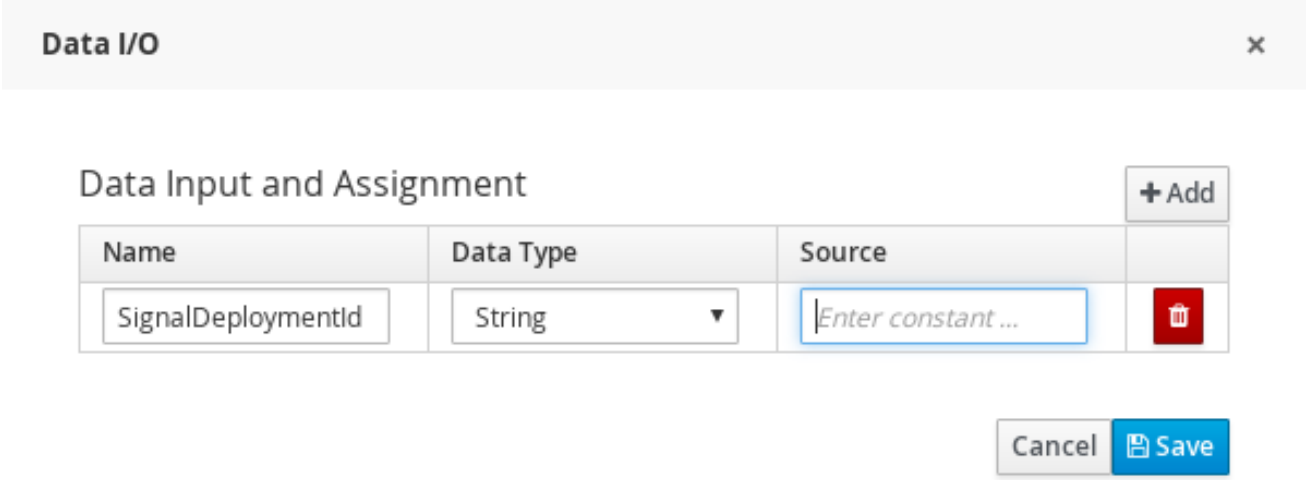
- **new org.jbpm.process.workitem.jms.JMSSendTaskWorkItemHandler("*CONNECTION_FACTORY_NAME*", "*DESTINATION_NAME*")**
- **new org.jbpm.process.workitem.jms.JMSSendTaskWorkItemHandler("*CONNECTION_FACTORY_NAME*", "*DESTINATION_NAME*", *TRANSACTED*)**, where ***TRANSACTED*** can only have two values: **true** or **false**. The argument affects the relevant JMS session. See the Interface Connection Javadoc for more information.

Both cross-project signalling and process instance signalling within a project is supported. To do so, you can specify the following data inputs in the **DataInputAssociations** property of your Signal event in the Process Designer. See Section A.3.1.2, “Catching and Processing Signals” for more information.

- **Signal**: the name of a signal which will be thrown. This value should match the **SignalRef** property in the signal definition.
  SignalWorkItemId: the ID of a Work Item which will be completed.

  These two data inputs are mutually exclusive.
- **SignalProcessInstanceId**: the target process instance ID (optional).
- **SignalDeploymentId**: the target deployment ID (required).

Figure A.3. Specifying SignalDeploymentId Data Input



The data inputs provide information about the signal, target deployment, and target process instance. For external signalling, the deployment ID is required, because an unrestricted broadcast would negatively impact the performance in large environments.

To send signals and messages in asynchronous processes, you need to configure a receiver of the signals accordingly: that is to limit a number of session for a given endpoint. By default, the receiver message-driven bean (**org.jbpm.process.workitem.jms.JMSSignalReceiver**) does not limit a concurrent processing.

Open the ***EAP_HOME*/standalone/deployments/business-central.war/WEB-INF/ejb-jar.xml** file and add the following activation specification property to the **JMSSignalReceiver** message-driven bean:

```
<activation-config-property>
  <activation-config-property-name>maxSession</activation-config-property-
name>
  <activation-config-property-value>1</activation-config-property-value>
</activation-config-property>
```

The message-driven bean should look like the following:

```
<message-driven>
  <ejb-name>JMSSignalReceiver</ejb-name>
  <ejb-class>org.jbpm.process.workitem.jms.JMSSignalReceiver</ejb-class>
  <transaction-type>Bean</transaction-type>
  <activation-config>
    <activation-config-property>
      <activation-config-property-name>destinationType</activation-config-
property-name>
      <activation-config-property-value>javax.jms.Queue</activation-
config-property-value>
    </activation-config-property>
    <activation-config-property>
      <activation-config-property-name>destination</activation-config-
property-name>
      <activation-config-property-
value>java:/queue/KIE.SIGNAL</activation-config-property-value>
    </activation-config-property>
```

```
    <activation-config-property>
      <activation-config-property-name>maxSession</activation-config-
property-name>
      <activation-config-property-value>1</activation-config-property-
value>
    </activation-config-property>
  </activation-config>
</message-driven>
```

This setting ensures that all messages, even the ones that were sent concurrently, will be processed serially and that notifications sent to the parent process instance will be delivered and will not cause any conflicts.

A.3.1.2. Catching and Processing Signals

Signals are caught by the following Catch Event types:

Types of Signal Catch Events

- Start Catch Event
- Intermediate Catch Event
- Boundary Catch Event

To catch and process a signal, create an appropriate Catching Signal Event in the Process Designer, and set the following properties:

SignalRef

The Signal's Reference.
Value: The same as the Throwing Signal Event's **SignalRef**.

DataOutputAssociations

The variables used to store the output of the received signal, if applicable.
To assign a data output:

1. Select the appropriate catch event type in the Business Process Designer.
2. Click  to open the **Properties** tab.
3. click the drop down menu next to the **DataOutputAssociations** property, and click **Add**.
4. In the new row, fill in a name for the association.
5. Select the expected data type from the dropdown menu. Selecting **Custom...** enables you to type in any class name.
6. Select the target process variable, where the output will be stored.
7. Click **Save** to save the association.
   For more information about setting process variables, see Section 4.9, "Variables".

A.3.1.3. Triggering Signals using API

To signal a process instance directly, that is equivalent to the process Instance scope, use the following API function:

```
ksession.signalEvent(eventType, data, processInstanceId)
```

Here, the parameters used are as follows:

eventType

The signal's reference, **SignalRef** in Process Designer.
Value: A **`String`**. You can also reference a process variable using the string **`#{myVar}`** for a process variable **`myVar`**.

data

The signal's data.
Value: Instance of a data type accepted by the corresponding Catching Signal Event. Typically an arbitrary **`Object`**.

processInstanceId

The Process ID of the Process being signalled.

You can use a more general version of the above function, which does not specify the parameter **`processInstanceId`**. That results in signalling all processes in the given ksession, that is equivalent to the Default scope:

```
ksession.signalEvent(eventType, data)
```

The usage of the arguments **`eventType`** and **`data`** is the same as above.

To trigger a Signal from a script, that is a Script Task, or using on-entry or on-exit actions of a node, use the following API function:

```
kcontext.getKieRuntime().signalEvent(
    eventType, data, kcontext.getProcessInstance().getId());
```

The usage of the arguments **`eventType`** and **`data`** is the same as above.

A.3.2. Messages

> A Message represents the content of a communication between two Participants. In BPMN 2.0, a Message is a graphical decorator (it was a supporting element in BPMN 1.2). An ItemDefinition is used to specify the Message structure.[1]

Messages are similar objects to Signals; the main difference is that when you are throwing the message, you must uniquely identify the recipient of the Message. In Red Hat JBoss BPM Suite, this is achieved by specifying both the element ID and the Process Instance ID. For this reason, Messages do not benefit from the scope feature of Signals.

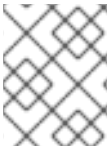
A.3.2.1. Sending Messages

Like Signals, Messages are sent by Throw Events of one of the following types:

Types of Message Throw Events

- Intermediate Throw Event
- End Throw Event
- Send Task

When creating the appropriate Throw Event, register a custom handler for the Send Task Work Item. Red Hat JBoss BPM Suite provides only dummy implementation by default. It is recommended to use the JMS-based **`org.jbpm.process.workitem.jms.JMSSendTaskWorkItemHandler`**.



NOTE

If necessary, you can emulate the message-sending mechanism using signals and their scopes so that only one element can receive the given signal.

A.3.2.2. Catching Messages

The process for catching messages does not differ from receiving signals, except you use the **MessageRef** element property instead of **SignalRef**. See Section A.3.1.2, “Catching and Processing Signals”.

WARNING

When catching messages through the API, the **MessageRef** property of the Catching Event is not the same as the **`eventType`** parameter of the API call. See Section A.3.2.3, “Sending Messages using API”.

A.3.2.3. Sending Messages using API

To send a message using the API, use the following function:

```
ksession.signalEvent(eventType, data, processInstanceId)
```

Here, the parameters used are as follows:

eventType

A **`String`** that starts with **`Message-`** and contains the message’s reference (**MessageRef**). You can also reference a process variable using the string **`#{myVar}`** for a process variable **`myVar`**.
Examples:

- **`Message-SampleMessage1`** for **MessageRef** **`SampleMessage1`**.
- **`#{myVar}`** for process variable **`myVar`**. The value of **`myVar`** must be a **`String`** starting with **`Message-`**.

data

The message's data.
Value: An arbitrary **`Object`**.

processInstanceId

The Process ID of the process being messaged.

To send a Message from a script (a Script Task) or using on-entry or on-exit actions of a node, use the following API function:

```
kcontext.getKieRuntime().signalEvent(
    eventType, data, kcontext.getProcessInstance().getId());
```

The usage of the arguments **`eventType`** and **`data`** is the same as above.

A.3.3. Escalation

> "An Escalation identifies a business situation that a Process might need to react to." [2]

The Escalation mechanism is intended for the handling of events that need the attention of someone of higher rank, or require additional handling.

Escalation is represented by an escalation object that is propagated across the process instances. It is produced by the Escalation Intermediate Throw Event or Escalation End Event, and can be consumed by exactly one Escalation Start Event or Escalation Intermediate Catch Event. Once produced, it is propagated within the current context and then further up the contexts until caught by an Escalation Start Event or Escalation Intermediate Catch Event, which is waiting for an Escalation with the particular Escalation Code. If an escalation remains uncaught, the process instance is **`ABORTED`**.

Mandatory Attributes

Escalation Code

string with the escalation code

A.4. TRANSACTION MECHANISMS

A.4.1. Errors

An Error represents a critical problem in a Process execution and is indicated by the Error End Event. When a Process finishes with an Error End Event, the event produces an Error object with a particular Error Code that identifies the particular error end that occurred. The Error End Event represents an unsuccessful execution of the given Process or Activity. Once generated, it is propagated as an object within the current context and then further up the contexts until caught by the respective catching Error Intermediate Event or Error Start Event, which is waiting for an Error with the particular Error Code. If the Error is not caught and is propagated to the upper-most Process context, the Process instance becomes ABORTED.

Every Error defines its Error Code, which is unique in the respective Process.

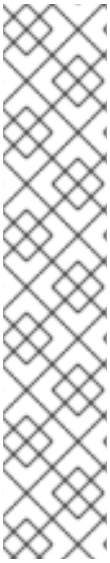
Attributes

Error Code

Error Code defined as a String unique within the Process.

A.4.2. Compensation

Compensation is a mechanism that allows you to handle business exceptions that might occur in a Process or Sub-Process (Business transactions), that is to compensate for a failed transaction, where the transaction is presented by the Process or Sub-Process, and then continues the execution using the regular Flow path. Note, that compensation is triggered only after the execution of the transaction has finished and that either with a Compensation End Event or with a Cancel End Event.



NOTE ON BUSINESS EXCEPTIONS

Consider implementing handling of business exceptions in the following cases:

- When an interaction with an external party or 3rd party system may fail or be faulty
- When you can not fully check the input data received by your Process (for example, a client’s address information)
- When there are parts of your Process that are particularly dependent on one of the following:
  - Company policy or policy governing certain in-house procedures
  - Laws governing the business process (such as, age requirements)

If a business transaction finishes with a Compensation End Event, the Event produces a "request" for compensation handling. The compensation request is identified by ID and can be consumed only by the respective Compensation Intermediate Event placed on the boundary of the transaction Elements and Compensation Start Event. The Compensation Intermediate Event is connected with an Association Flow to the Activity that defines the compensation, such as a Sub-Process or Task. The execution flow either waits for the compensation activity to finish or resumes depending on the **`Wait for completion`** property set on the Compensation End Event of the business transaction that is being compensated.

If a business transaction contains an Event Sub-Process that starts with a Compensation Start Event, the Event Sub-Process is run as well if compensation is triggered.

The Activity the Compensation Intermediate Event points to, might be a Sub-Process. Note that the Sub-Process must start with the Compensation Start Event.

If running over a multi-instance Sub-Process, compensation mechanism of individual instances do not influence each other.

A.5. TIMING

Timing is a mechanism for scheduling actions and is used by Timer Intermediate and Timer Start events. It allows you to delay further execution of a process or task.

NOTE

A timer event can be triggered only after the transaction commits, while the timer countdown starts right after entering the node (the attached node in case of a boundary event). In other words, a timer event is only designed for those use cases where there is a wait state, such as a "User Task". If you want to be notified of the timeout of a synchronous operation without a wait state, *a boundary timer event is not suitable*.

The timing strategy is defined by the following timer properties:

Time Duration

defines the period for which the execution of the event is put on hold. The execution continues after the defined period has elapsed. The timer is applied only once.

Time Cycle

This defines the time between subsequent timer activations. If the period is **`0`**, the timer is triggered only once.

The value for these properties can be provided as either Cron or as an expression by defining the, *Time Cycle Language* property.

Cron

[#d][\#h][\#m][\#s][#[ms]]

Example A.6. Timer period with literal values

```
1d 2h 3m 4s 5ms
```

The element will be executed after 1 day, 2 hours, 3 minutes, 4 seconds, and 5 milliseconds.

Any valid **`ISO8601`** date format that supports both one shot timers and repeatable timers can be used. Timers can be defined as date and time representation, time duration or repeating intervals. For example:

Date

2013-12-24T20:00:00.000+02:00 - fires exactly at Christmas Eve at 8PM

Duration

PT2S - fires 1 after 2 seconds

Repetable Intervals

R/PT1S - fires every second, no limit, alternatively R5/PT1S will fire 5 times every second

None

#{expression}

Example A.7. Timer period with expression

```
myVariable.getValue()
```

> The element will be executed after time period returned by the call **`myVariable.getValue()`**.

A.6. PROCESS ELEMENTS

DISCLAIMER

This chapter contains introduction to BPMN elements and their semantics. By no means does it aspire to be an exhaustive language specification. For details about BPMN see the Business Process Model and Notation, Version 2.0: The BPMN 2.0 specification is an OMG specification that defines standards on how to graphically represent a business process, defines execution semantics for the elements along with an XML format of process definitions source.

The specification also includes details on choreographies and collaboration. Note that Red Hat JBoss BPM Suite focuses exclusive on executable processes and supports a significant subset of the BPMN elements including the most common types that can be used inside executable processes.

A Process Element is a node of the Process definition. The term covers the nodes with execution semantics as well as those without. Elements with execution semantics define the execution workflow of the Process, while Elements without execution semantics (Artifacts) allow users to provide notes and further information on the Process or any of its Elements so as to accommodate collaboration of multiple users with different roles, such as, business analyst, business manager, process designer.

All Elements with execution semantics define their generic properties.

Generic Process Element Properties

ID

ID defined as a String unique in the parent Knowledge Base

Name

Element display name

A.7. EVENT TYPES

Events are triggers, which when occur, impact a business process. Events are classified as start events, end events, and intermediate events. A start event indicates the beginning of a business process. An end event indicates the completion of a business process. And intermediate events drive the flow of a business process. Every event has an event ID and a name. You can implement triggers for each of these event types to identify the conditions under which an event is triggered. If the conditions of the triggers are not met, the events are not initialized, and hence the process flow does not complete.

A.7.1. Start Event

Every process must have at least one Start Event with no incoming and exactly one outgoing flow.

Multiple Start Event types are supported:

- None Start Event
- Signal Start Event
- Timer Start Event
- Conditional Start Event
- Message Start Event
- Compensation Start Event
- Error Start Event
- Escalation Start Event

All start events, except for the None Start Event, define a trigger. When you start a process, the trigger needs to be fulfilled. If no start event can be triggered, the process is never instantiated.

A.7.1.1. Start Event types

A.7.1.1.1. None Start Event

The None Start Event is a Start Event without a trigger condition.

A Process or Sub-Process can contain at most one None Start Event, which is triggered on Process or Sub-Process start by default and the outgoing flow is taken immediately.

When used in a Sub-Process, the execution is transferred from the parent Process into the Sub-Process and the None Start Event is triggered (the token is taken from the parent Sub-Process Activity and the None Start Event of the Sub-Process generates a token).

A.7.1.1.2. Message Start Event

A Process or an Event Sub-Process can contain multiple Message Start Events, which are triggered when triggered by a particular Message. The Process instance with a Message Start Event only starts its execution from this event after it has received the respective Message: The Process is instantiated and its Message Start Event is executed immediately (its outgoing Flow is taken).

As a Message can be consumed by an arbitrary number of Processes and Process elements, including no Elements, one Message can trigger multiple Message Start Events and therefore instantiate multiple Processes.

Attributes

Message

ID of the expected Message object

A.7.1.1.3. Timer Start Event

The Timer Start Event is a Start Event with a Timing definition (for details on Timing see Section A.5, “Timing”).

A Process can contain at multiple Timer Start Events, which is triggered on Process start by default and then the Timing is applied.

When used in a Sub-Process, the execution is transferred from the parent Process into the Sub-Process and the Timer Start Event is triggered: the token is taken from the parent Sub-Process Activity and the Timer Start Event of the Sub-Process is triggered and waits for the Timing to be fulfilled. Once the time defined by the Timing definition has been reached, the outgoing Flow is taken.

Attributes

Timer

Timing definition

A.7.1.1.4. Escalation Start Event

The Escalation Start Event is a Start Event that is triggered by an Escalation with a particular Escalation code (see Section A.3.3, “Escalation”).

Process can contain multiple Escalation Start Events. The Process instance with an Escalation Start Event only starts its execution from this event after it has received the respective Escalation object: The Process is instantiated and its Escalation Start Event is executed immediately (its outgoing Flow is taken).

Attributes

Escalation Code

Expected Escalation Code

A.7.1.1.5. Conditional Start Event

The Conditional Start Event is a Start Event with a Boolean condition definition. The Process execution with such a Start Event continues only if the condition is evaluated to **`true`** after the Start Event has been instantiated. The execution is triggered always when the condition is evaluated to **`false`** and then to **`true`**.

A Process can contain at multiple Conditional Start Events.

Attributes

Condition

Boolean condition

A.7.1.1.6. Error Start Event

An Error Start Event can be used to start a Process or Sub-Process. These can contain multiple Error Start Events, which are triggered when an Error object with a particular ErrorRef is received. The Error object can be produced by an Error End Event and signalizes an incorrect Process ending. The Process instance with the Error Start Event starts execution after it has received the respective Error object so as to handle such incorrect ending: The Error Start Event is executed immediately (its outgoing Flow is taken).

Attributes

ErrorCode

code of the expected Error object

A.7.1.1.7. Compensation Start Event

A Compensation Start Event is used to start an Compensation Event Sub-Process when using a Sub-Process as the target Activity of a Compensation Intermediate Event.

A.7.1.1.8. Signal Start Event

The Signal Start Event is a Start Event that is triggered by a Signal with a particular Signal Code (see Section A.3.1, “Signals”).

Process can contain multiple Signal Start Events. The Signal Start Event only starts its execution within the Process instance after the instance has received the respective Signal: on Signal receiving, the Signal Start Event is executed immediately (its outgoing Flow is taken).

Attributes

SignalCode

Expected Signal Code

A.7.2. Intermediate Events

A.7.2.1. Intermediate Events

“... the Intermediate Event indicates where something happens (an Event) somewhere between the start and end of a Process. It will affect the flow of the Process, but will not start or (directly) terminate the Process.[3]”

Intermediate Event handles a particular situation that occurs during Process execution. The situation is the trigger of the Intermediate Event.

In a Process, Intermediate Events can be placed as follows:

in a Process workflow with one optional incoming and one outgoing Flow

The event is executed as part of the workflow. If the Event has no incoming Flow, its execution is triggered always when the respective trigger occurs during the entire Process instance execution. If the Event has an incoming Flow it is executed as part of the Process workflow. Once triggered, the Event’s outgoing Flow is taken only after the respective Event has occurred.

on an Activity boundary with one outgoing Flow

If the Event occurs while the Activity is being executed, the Event triggers its execution to the outgoing Flow. One Activity may have multiple boundary Intermediate Events. Note that depending on the behavior you require from the Activity with the boundary Intermediate Event, you can use either of the following Intermediate Event type:

- interrupting: the Activity execution is interrupted and the execution of the Intermediate Event is triggered.
- non-interrupting: the Intermediate Event is triggered and the Activity execution continues.

Based on the type of Event cause the execution of the Intermediate Event (triggers), the following Intermediate Events are distinguished:

Timer Intermediate Event

Delays the execution of the outgoing flow.

Conditional Intermediate Event

Is triggered when its condition evaluates to **`true`**.

Error Intermediate Event

Is triggered by an Error object with the given Error Code.

Escalation Intermediate Event

has two subtypes: Catching Escalation Intermediate Event that is triggered by a Escalation and Throwing Escalation Intermediate Event that produces an Escalation when executed.

Signal Intermediate Event

has two subtypes: Catching Signal Intermediate Event that is triggered by a Signal and Throwing Signal Intermediate Event that produces a Signal when executed.

Message Intermediate Event

has two subtypes: Catching Message Intermediate Event that is triggered by a Message and Throwing Message Intermediate Event that produces a Message when executed.

A.7.2.2. Intermediate Event types

A.7.2.2.1. None Intermediate Event

None Intermediate Event is an abstract Intermediate Event and displays all possible Intermediate Event properties.

A.7.2.2.2. Timer Intermediate Event

A timer intermediate event allows you to delay workflow execution or to trigger the workflow execution periodically. It represents a timer that can trigger one or multiple times after a given period of time. When triggered, the timer condition (the defined time) is checked and the outgoing flow is taken. For more information about timing, see Section A.5, “Timing”.

When placed in the process workflow, a timer intermediate event has one incoming flow and one outgoing flow. Its execution starts when the incoming flow transfers to the event. When placed on an activity boundary, the execution is trigger at the same time as the activity execution.

The timer is canceled if the timer element is canceled, for example, by completing or aborting the enclosing process instance.

Attributes

Timer Date

Starts process at the specified date.

Timer Delay

Time delay before the event triggers its outgoing flow for the first time.

Timer Period

Period between two subsequent triggers.
If set to **0**, the event execution is not repeated.

A.7.2.2.3. Conditional Intermediate Event

A Conditional Intermediate Event is an Intermediate Event with a boolean condition as its trigger. The Event triggers further workflow execution when the condition evaluates to **true** and its outgoing Flow is taken.

The Event must define its boolean **Conditional**. When placed in the Process workflow, a Conditional Intermediate Event has one incoming Flow and one outgoing Flow and its execution starts when the incoming Flow transfers to the Event. When placed on an Activity boundary, the execution is triggered at the same time as the Activity execution. Note, that if the Event is non-interrupting, the Event triggers continuously while the condition is **true**.

Attributes

Condition

Boolean condition that must be evaluated to **true** for the execution to continue.

A.7.2.2.4. Compensation Intermediate Event

A compensation intermediate event is a boundary event attached to an activity in a transaction sub-process. It can finish with a compensation end event or a cancel end event. The compensation intermediate event must be associated with a flow, which is connected to the compensation activity.

The activity associated with the boundary compensation intermediate event is executed if the transaction sub-process finishes with the compensation end event. The execution continues with the respective flow.

A.7.2.2.5. Message Intermediate Event

A Message Intermediate Event is an Intermediate Event that allows you to manage a Message object.Use one of the following events:

- **Throwing Message Intermediate Event** produces a Message object based on the defined properties.
- **Catching Message Intermediate Event** listens for a Message object with the defined properties.

A.7.2.2.6. Message Intermediate Event types

A.7.2.2.6.1. Throwing Message Intermediate Event

When reached on execution, a Throwing Message Intermediate Event produces a Message and the execution continues to its outgoing Flow.

Attributes

MessageRef

ID of the produced Message object.

A.7.2.2.6.2. Catching Message Intermediate Event

When reached on execution, a Catching Message Intermediate Event awaits a Message defined in its properties. Once the Message is received, the Event triggers execution of its outgoing Flow.

MessageRef

ID of the expected Message object.

CancelActivity

If the event is placed on the boundary of an activity and **`Cancel Activity`** property is set to **`true`**, the activity execution is canceled when the event receives its Escalation object.

A.7.2.2.7. Escalation Intermediate Event

An Escalation Intermediate Event is an Intermediate Event that allows you to produce or consume an Escalation object. Depending on the action the event element is to perform, you need to use either of the following:

- **Throwing Escalation Intermediate Event** produces an Escalation object based on the defined properties.
- **Catching Escalation Intermediate Event** listens for an Escalation object with the defined properties.

A.7.2.2.8. Escalation Intermediate Event types

A.7.2.2.8.1. Throwing Escalation Intermediate Event

When reached on execution, a Throwing Escalation Intermediate Event produces an Escalation object and the execution continues to its outgoing Flow.

Attributes

EscalationCode

ID of the produced Escalation object.

A.7.2.2.8.2. Catching Escalation Intermediate Event

When reached on execution, a Catching Escalation Intermediate Event awaits an Escalation object defined in its properties. Once the object is received, the Event triggers execution of its outgoing Flow.

EscalationCode

Code of the expected Escalation object.

CancelActivity

If the event is placed on the boundary of an activity and **`Cancel Activity`** property is set to **`true`**, the activity execution is canceled when the event receives its Escalation object.

A.7.2.2.9. Error Intermediate Event

An Error Intermediate Event is an Intermediate Event that can be used only on an Activity

boundary. It allows the Process to react to an Error End Event in the respective Activity. The Activity must not be atomic. When the Activity finishes with an Error End Event that produces an Error with the respective ErrorCode, the Error Intermediate Event catches the Error object and execution continues to the outgoing Flow of the Error Intermediate Event.

Attributes

ErrorRef

reference number of the Error object the Event is listening for

A.7.2.2.10. Error Intermediate Event types

A.7.2.2.10.1. Throwing Error Intermediate Event

When reached on execution, a Throwing Error Intermediate Event produces an Error object and the execution continues to its outgoing Flow.

Attributes

ErrorRef

reference number of the produced Error object

A.7.2.2.10.2. Catching Error Intermediate Event

When reached on execution, a Catching Error Intermediate Event awaits an Error object defined in its properties. Once the object is received, the Event triggers execution of its outgoing Flow.

Attributes

ErrorRef

Reference number of the expected Error object.

CancelActivity

If the event is placed on the boundary of an activity and **`Cancel Activity`** property is set to **`true`**, the activity execution is canceled when the event receives its Escalation object.

A.7.2.2.11. Signal Intermediate Event

A Signal Intermediate Event is an Intermediate Event that allows you to produce or consume a Signal object. Depending on the action the event element is to perform, you need to use either of the following:

- **Throwing Signal Intermediate Event** produces a Signal object based on the defined properties.
- **Catching Signal Intermediate Event** listens for a Signal object with the defined properties.

A.7.2.2.12. Signal Intermediate Event types

A.7.2.2.12.1. Throwing Signal Intermediate Event

When reached on execution, a Throwing Signal Intermediate Event produces a Signal object and the execution continues to its outgoing Flow.

Attributes

SignalRef

The Signal code that is to be sent or consumed.

A.7.2.2.12.2. Catching Signal Intermediate Event

When reached on execution, a catching signal intermediate event awaits a signal object defined in its properties. Once the object is received, the event triggers execution of its outgoing flow.

Attributes

SignalRef

Reference code of the expected Signal object.

CancelActivity

If the event is placed on the boundary of an activity and **`Cancel Activity`** property is set to **`true`**, the activity execution is canceled when the event receives its Escalation object.

A.7.3. End Events

An End Event is a node that ends a particular workflow. It has one or more incoming Sequence Flows and no outgoing Flow.

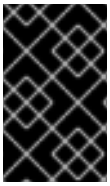
A Process must contain at least one End Event.

During runtime, an End Event finishes the Process workflow. It might finish only the workflow that reached the End Event or all workflows in the Process instance depending on its type.

A.7.3.1. End Event types

A.7.3.1.1. Simple End Event

The Simple End Event finishes the incoming workflow (consumes the incoming token). Any other running workflows in the Process or Sub-Process remain uninfluenced.



TERMINATE PROPERTY ON SIMPLE END EVENT

In Red Hat JBoss BPM Suite, the Simple End Event has the **`Terminate`** property in its Property tab. This is a boolean property, which turns a Simple End Event into a Terminate End Event when set to **`true`**.

A.7.3.1.2. Message End Event

When a Flow enters a Message End Event, the Flow finishes and the Event produces a Message as defined in its properties.

A.7.3.1.3. Escalation End Event

The Escalation End Event finishes the incoming workflow (consumes the incoming token) and produces an Escalation signal as defined in its properties, triggering the escalation process.

A.7.3.1.4. Terminate End Event

The Terminate End Event finishes all execution flows in the given process instance. Activities being executed are canceled. If a Terminate End Event is reached in a sub-process, the entire process instance is terminated.

A.7.3.1.5. Throwing Error End Event

The Throwing Error End Event finishes the incoming workflow (consumes the incoming token). Any other running workflows in the Process or Sub-Process remain uninfluenced.

Attributes

ErrorRef

reference code of the produced Error object

A.7.3.1.6. Cancel End Event

If a Process or Sub-Process finishes with a Cancel End Event, any compensations defined for the namespace are executed, and the Process or Sub-Process finishes as CANCELED.

A.7.3.1.7. Compensation End Event

A Compensation End Event is used to finish a transaction Sub-Process and trigger the compensation defined by the Compensation Intermediate Event attached to the boundary of the Sub-Process activities.

A.7.3.1.8. Signal End Event

A Throwing Signal End Event is used to finish a Process or Sub-Process flow. When the execution flow enters the element, the execution flow finishes and a Signal identified by its **`SignalRef`** property value.

A.7.4. Scope of Events

An event can send signals globally or be limited to a single process instance. You can use the scope attribute for events to define if a signal is to be considered internal (only for one process instance) or external (for all process instances that are waiting). The scope attribute called **`Signal Scope`** on the **Properties** panel of the process designer allows you to change the scope of the signal throw intermediate/end events.

The Scope data input is an optional property implemented to provide the following scope of throw events:

- *default*: The throw event is set to default scope when no scope is selected. In this case, the signal is given a ksession and it signals only the elements known to that ksession. The signal behavior depends on the strategy used:
    - Singleton: In this case, the event signals all instances available for this ksession.

    - Per request: In this case, the event signals only currently processed process instance and those with start signal events.
    - Per process instance: In this case, similar to the per request strategy, the event signals only currently processed process instance and those with start signal events.

- *processInstance*: This indicates that the scope of the signal remains within the same process instance that the signal is thrown from.
- *project/runtime manager*: This indicates that either the scope of the signal is bound to runtime manager or that it signals runtime manager that the instance is bound to.
- *external scope*: This indicates that the scope of the signal can be both project scope and cross deployments. For the signal to have a cross deployment scope, it requires to have a process variable called **`SignalDeploymentId`**. The **`SignalDeploymentId`** process variable provides information about which deployment/project must be the target of the signal.

A.8. GATEWAYS

A.8.1. Gateways

"Gateways are used to control how Sequence Flows interact as they converge and diverge within a Process.[4]"

Gateways are used to create or synchronize branches in the workflow using a set of conditions which is called the gating mechanism. Gateways are either converging (multiple Flows into one Flow) or diverging (One Flow into multiple Flows).

One Gateway cannot have multiple incoming *and* multiple outgoing Flows.

Depending on the gating mechanism you want to apply, you can use the following types of gateways:

- Parallel (AND): in a converging gateway, waits for all incoming Flows. In a diverging gateway, takes all outgoing Flows simultaneously.
- Inclusive (OR): in a converging gateway, waits for all incoming Flows whose condition evaluates to true. In a diverging gateway takes all outgoing Flows whose condition evaluates to true.
- Exclusive (XOR): in a converging gateway, only the first incoming Flow whose condition evaluates to true is chosen. In a diverging gateway only one outgoing Flow is chosen.
- Event-based: used only in diverging gateways for reacting to events. See Section A.8.2.1, "Event-based Gateway".
- Data-based Exclusive: used in both diverging and converging gateways to make decisions based on available data. See Section A.8.2.4, "Data-based Exclusive Gateway".

A.8.2. Gateway types

A.8.2.1. Event-based Gateway

“The Event-Based Gateway has pass-through semantics for a set of incoming branches (merging behavior). Exactly one of the outgoing branches is activated afterwards (branching behavior), depending on which of Events of the Gateway configuration is first triggered. [5]”

The Gateway is only diverging and allows you to react to possible Events as opposed to the Data-based Exclusive Gateway, which reacts to the process data. It is the Event that actually occurs that decides which outgoing Flow is taken. As it provides the mechanism to react to exactly one of the possible Events, it is exclusive, that is, only one outgoing Flow is taken.

The Gateway might act as a Start Event, where the process is instantiated only if one the Intermediate Events connected to the Event-Based Gateway occurs.

A.8.2.2. Parallel Gateway

“A Parallel Gateway is used to synchronize (combine) parallel flows and to create parallel flows.[6]”

Diverging

Once the incoming Flow is taken, all outgoing Flows are taken simultaneously.

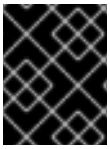
Converging

The Gateway waits until all incoming Flows have entered and only then triggers the outgoing Flow.

A.8.2.3. Inclusive Gateway

Diverging

Once the incoming Flow is taken, all outgoing Flows whose condition evaluates to true are taken. Connections with lower priority numbers are triggered before triggering higher priority ones; priorities are evaluated but the BPMN2 specification doesn’t guarantee this. So for portability reasons it is recommended that you do not depend on this.



IMPORTANT

Make sure that at least one of the outgoing Flow evaluates to true at runtime; otherwise, the process instance terminates with a runtime exception.

Converging

The Gateway merges all incoming Flows previously created by a diverging Inclusive Gateway; that is, it serves as a synchronizing entry point for the Inclusive Gateway branches.

Attributes

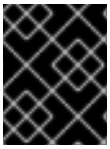
Default gate

The outgoing Flow taken by default if no other Flow can be taken

A.8.2.4. Data-based Exclusive Gateway

Diverging

The Gateway triggers exactly one outgoing Flow: the Flow with the constraint evaluated to true and the *lowest* Priority is taken. After evaluating the constraints that are linked to the outgoing Flows: the constraint with the lowest priority number that evaluates to true is selected.



POSSIBLE RUNTIME EXCEPTION

Make sure that at least one of the outgoing Flows evaluates to true at runtime: if no Flow can be taken, the execution returns a runtime exception.

Converging

The Gateway allows a workflow branch to continue to its outgoing Flow as soon as it reaches the Gateway; that is, whenever on of the incoming Flows triggers the Gateway, the workflow is sent to the outgoing Flow of the Gateway; if it is triggered from more than one incoming connection, it triggers the next node for each trigger.

Attributes

Default gate

The outgoing Flow taken by default if no other Flow can be taken

A.9. ACTIVITIES, TASKS AND SUB-PROCESSES

A.9.1. Activity

> "An Activity is work that is performed within a Business Process." [7]

This is opposed to the execution semantics of other elements that defined the Process logic.

An Activity can be further specified as a Sub-Process or a Task: while Task is atomic, that is represents a single piece of work, a Sub-Process is compound, that is it can be broken down into multiple Process elements.

An Activity in jBPM expects one incoming and one outgoing flow. If you want to design an activity with multiple incoming and multiple outgoing flows, set the value of the system property jbpm.enable.multi.con to true.

Activities have the basic properties just like any other Process element (ID and Name). Note that Activities (all of their subtypes, that is, Tasks, Sub-Process) have additional properties specific for the given Activity or Task type.

A.9.2. Activity mechanisms

A.9.2.1. Multiple instances

Activities can be run in multiple instances on execution. Individual instances are run in a sequential manner. The instances are run based on a collection of elements: for every element in the collection, a new Activity instance is created..

Every Activity has therefore the **`Collection Expression`** attribute that defines the collection with elements to iterate through.

A.9.2.2. Activity types

A.9.2.2.1. Call Activity

“A Call Activity identifies a point in the Process where a global Process or a Global Task is used. The Call Activity acts as a 'wrapper' for the invocation of a global Process or Global Task within the execution. The activation of a call Activity results in the transfer of control to the called global Process or Global Task. [8]”

A Call Activity, previously Reusable Sub-Process, represents an invocation of a Process from within a Process. The Activity must have one incoming and one outgoing Flow.

When the execution flow reaches the Activity, an instance of the Process with the ID defined by the Activity is created.

Attributes

Called Element

ID of the Process to be called and instantiated by the Activity

A.9.3. Tasks

A.9.3.1. Task types

A Task is the smallest unit of work in a Process flow and to help identify the various types of Tasks that can be performed, Red Hat JBoss BPM Suite uses the BPMN guidelines to separate them based on the types of inherent behavior that the Tasks might represent.

A Task that doesn’t serve a direct defined purpose is either called the **`None`** Task or the **`Abstract`** Task (deprecated).

The different types of tasks available in JBoss BPM Suite are listed here, except for the **`User`** Task.

We define the User Task in another section (see Section A.9.5, “User Task”).

A.9.3.2. Generic Task

> "Abstract Task: Upon activation, the Abstract Task completes. This is a conceptual model only; an Abstract Task is never actually executed by an IT system." [9]

A.9.3.3. Send Task

> "Send Task: Upon activation, the data in the associated Message is assigned from the data in the Data Input of the Send Task. The Message is sent and the Send Task completes." [10]

Attributes

MessageRef

the MessageRef ID of the generated Message

NOTE

In Red Hat JBoss BPM Suite 6.x, the Send task is not supported. A custom **`WorkItemHandler`** implementation is needed to use the Send task.

A.9.3.4. Receive Task

"Upon activation, the Receive Task begins waiting for the associated Message. When the Message arrives, the data in the Data Output of the Receive Task is assigned from the data in the Message, and Receive Task completes." [11]

Attributes

MessageRef

the associated Message

A.9.3.5. Manual Task

"Upon activation, the Manual Task is distributed to the assigned person or group of people. When the work has been done, the Manual Task completes. This is a conceptual model only; a Manual Task is never actually executed by an IT system." [12]

A.9.3.6. Service Task

A Service Task is used with the built-in **`ServiceTaskHandler`** to invoke Java methods or Web Services.

Implementation

The underlying technology that will be used to implement this task. You can use **`unspecified`** or **`WebService`** where **`WebService`** is the default value.

OperationRef

This attribute specifies the operation that is invoked by the Service Task. (typically method of Java class or method of WebService).

A.9.3.6.1. Using a Service Task to Call a WebService

Service Task can be used to invoke a web service using a BPMN2 specification.

First, the web service must be configured as a part of the process definition. This can be done using a few dedicated constructs:

1. In the process definition BPMN2 source, import the WSDL:

```
<import importType="http://schemas.xmlsoap.org/wsdl/"
        location="http://wsf.cdyne.com/WeatherWS/Weather.asmx?WSDL"
        namespace="http://ws.cdyne.com/WeatherWS/"/>
```

2. Define the message, interface, and operations:

```
<itemDefinition id="_2-2-4_InMessageType" />
<message id="_2-2-4_InMessage" itemRef="_2-2-4_InMessageType" />

<interface id="_2-2-4_ServiceInterface" name=""
implementationRef="Weather">
  <operation id="_2-2-4_ServiceOperation" name="hello"
implementationRef="GetCityWeatherByZIP">
    <inMessageRef>_2-2-4_InMessage</inMessageRef>
  </operation>
</interface>
```

IMPORTANT

Ensure that the **implementationRef** attribute for both the interface and operations points to a valid service and operations in WSDL.

3. Set the **implementation** attribute to a web service (the default value is used if no attribute is specified) and the **operationRef** attribute to the defined operation:

```
<serviceTask id="_2"
             name="Service Task"
             operationRef="_2-2-4_ServiceOperation"
             implementation="##WebService" >
  ...
</serviceTask>
```

To use a request or response object of the service as a variable, they must all implement the **java.io.Serializable** interface in order to be properly persisted. To do so, configure JAXB to add the interface while generating classes from WSDL:

1. Create an XML binding file with the following content.

```
<?xml version="1.0" encoding="UTF-8"?>
<bindings xmlns="http://java.sun.com/xml/ns/jaxb"
          xmlns:xsi="http://www.w3.org/2000/10/XMLSchema-instance"
          xmlns:xjc="http://java.sun.com/xml/ns/jaxb/xjc"
          xsi:schemaLocation="http://java.sun.com/xml/ns/jaxb
http://java.sun.com/xml/ns/jaxb/bindingschema_2_0.xsd"
          version="2.1">
  <globalBindings>
    <serializable uid="1" />
  </globalBindings>
</bindings>
```

2. Add the Apache CXF Maven plugin (**cxf-codegen-plugin**) to the project's **pom.xml** file:

```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.cxf</groupId>
      <artifactId>cxf-codegen-plugin</artifactId>
      <version>CXF_VERSION</version>
      ...
```

```
    </plugin>
  </plugins>
<build>
```

A.9.3.7. Business Rule Task

“A Business Rule Task provides a mechanism for the Process to provide input to a Business Rules Engine and to get the output of calculations that the Business Rules Engine might provide. [13]”

The Task defines a set of rules that need to be evaluated and fired on Task execution. Any rule defined as part of the ruleflow group in a rule resource is fired.

When a Rule Task is reached in the Process, the engine starts executing the rules with the defined ruleflow group. When there are no more active rules with the ruleflow group, the execution continues to the next Element. During the ruleflow group execution, new activations belonging to the active ruleflow group can be added to the Agenda as these are changed by the other rules. Note that the Process continues immediately to the next Element if there are no active rules of the ruleflow group.

If the ruleflow group was already active, the ruleflow group remains active and the execution continues if all active rules of the ruleflow group have been completed.

Attributes

RuleFlow Group

the name of the rule flow group that includes the set of rules to be evaluated by the Task

A.9.3.8. Script Task

A Script Task represents a script that should be executed during the Process execution.

The associated **`Script`** can access any variables and globals.

When using a Script Task follow these rules:

- Avoid low-level implementation details in the Process: A Script Task could be used to manipulate variables but other concepts like a Service Task should be your first choice when modeling more complex behavior in a higher-level manner.
- The script should be executed immediately; if there is the possibility that the execution could take some time, use an asynchronous Service Task.
- Avoid contacting external services through a Script Task: it would be interacting with external services without the knowledge of the engine, which can be problematic. Model communication with an external service using a Service Task.
- Scripts should not throw exceptions. Runtime exceptions should be caught and for example managed inside the script or transformed into signals or errors that can then be handled inside the process.

When a Script Task is reached during execution, the script is performer and the outgoing Flow is taken.

Script

script to be executed

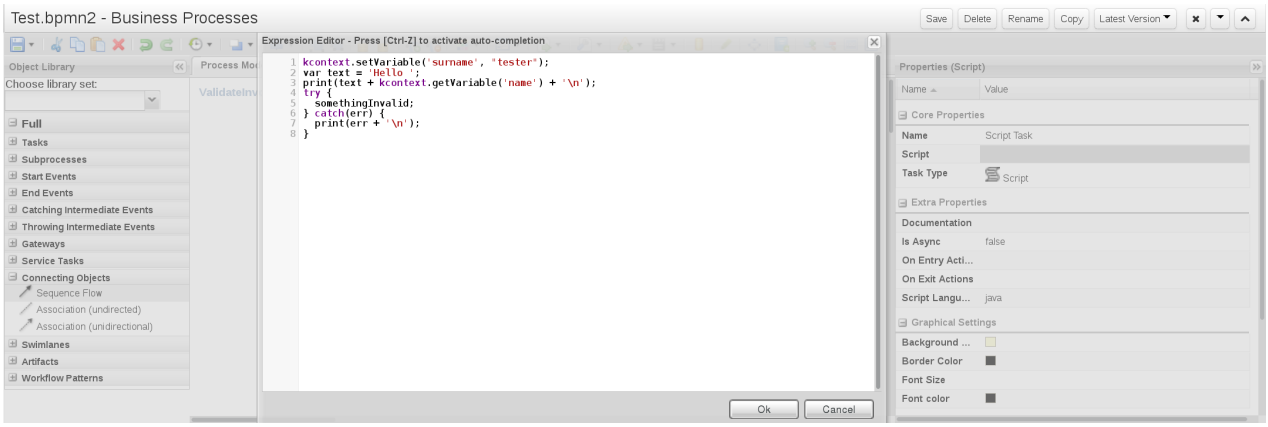
ScriptLanguage

language the script is defined in (currently supported languages are Java, JavaScript, and MVEL)

From JBoss BPM Suite 6.2 onwards, JavaScript is supported as dialect in Script Tasks. To define a Script Task in Business Central and JBoss Developer Studio using the process design tool:

1. Select a Script Task object from the Object Library menu on the left hand side and add it to the process design tool.
2. In the Properties panel on the right hand side, open the **Script** property.
3. Add the script to be executed to the Expression Editor and click **Ok**.

Example A.8. Script Task in Business Central using JavaScript



A.9.4. Sub-Process

“A Sub-Process is an Activity whose internal details have been modeled using Activities, Gateways, Events, and Sequence Flows. A Sub-Process is a graphical object within a Process, but it also can be 'opened up’to show a lower-level Process. [14]”

Therefore, a Sub-Process can be understood as a compound Activity or a *Process in a Process*. When reached during execution, the Element context is instantiated and the encapsulated process triggered. Note that, if you use a Terminating End Event inside a Sub-Process, the entire Process instance that contains the Sub-Process is terminated, not just the Sub-Process. A Sub-Process, just like a Process, ends when there are no more active Elements in it.

The following Sub-Process types are supported:

- Ad-Hoc Sub-Process: Sub-Process with no strict Element execution order
- Embedded Sub-Process: a "real" Sub-Process that is a part of the Parent Process execution and shares its data
- Reusable Sub-Process: a Sub-Process that is independent from its parent Process

- Event Sub-Process: a Sub-Process that is only triggered on a start event or a timer.

Note that any Sub-Process type can be also a Multi-Instance Sub-Process.

A.9.4.1. Embedded Sub-Process

An Embedded Sub-Process is a Sub-Process that encapsulates a part of the process.

It must contain a Start Event and at least one End Event. Note that the Element allows you to define local Sub-Process variables, that are accessible to all Elements inside this container.

A.9.4.2. AdHoc Sub-Process

“An Ad-Hoc Sub-Process is a specialized type of Sub-Process that is a group of Activities that have no REQUIRED sequence relationships. A set of Activities can be defined for the Process, but the sequence and number of performances for the Activities is determined by the performers of the Activities. [15]”

“An Ad-Hoc Sub-Process or Process contains a number of embedded inner Activities and is intended to be executed with a more flexible ordering compared to the typical routing of Processes. Unlike regular Processes, it does not contain a complete, structured BPMN diagram descriptionâi.e., from Start Event to End Event. Instead the Ad-Hoc Sub-Process contains only Activities, Sequence Flows, Gateways, and Intermediate Events. An Ad-Hoc Sub-Process MAY also contain Data Objects and Data Associations. The Activities within the Ad-Hoc Sub- Process are not REQUIRED to have incoming and outgoing Sequence Flows. However, it is possible to specify Sequence Flows between some of the contained Activities. When used, Sequence Flows will provide the same ordering constraints as in a regular Process. To have any meaning, Intermediate Events will have outgoing Sequence Flows and they can be triggered multiple times while the Ad-Hoc Sub-Process is active.[16]”

The Elements of an AdHoc Sub-Process are executed in parallel.

AdHocCompletionCondition

the condition that once met the execution is considered successful and finishes

AdHocCancelRemainingInstances

if set to **`true`**, once the AdHocCompletionCondition is met, execution of any Elements is immediately canceled.

A.9.4.3. Multi-instance Sub-Process

A Multiple Instance Sub-Process is a Sub-Process that is instantiated/run multiple times when its execution is triggered. The instances are created in a sequential manner: a new Sub-Process instance is created only after the previous instance has finished.

A Multiple instance Sub-Process has one incoming Connection and one outgoing Connection.

Collection expression

Variable that represents the collection of elements that are to be iterated over (The variable must be an array or be of the java.util.Collection type.)
If the collection expression evaluates to null or an empty collection, the Multi-Instances Sub-Process is completed immediately and the outgoing flow is taken.

Variable Name

Variable that will store the collection element used in the currently running iteration

A.9.4.4. Event Sub-Process

An Event Sub-Process becomes active when its start event gets triggered. It can interrupt the parent process context or run in parallel to it.

With no outgoing or incoming connections, only an event or a timer can trigger these Sub-Processes. These Sub-Processes are not part of the regular control flow. Although self-contained, they are executed in the context of the bounding Sub-Process.

You would use these Sub-Processes within a process flow to handle events that happen external to the main process flow. For example, while booking a flight, two events may occur: (interrupting) cancel booking, or (non-interrupting) check booking status. Both these events can be modeled using the Event Sub-Process.

A.9.5. User Task

> "A User Task is a typical 'workflow' Task where a human performer performs the Task with the assistance of a software application and is scheduled through a task list manager of some sort." [17]

The User Task cannot be performed automatically by the system and therefore requires an intervention of a human user, the Actor. Also, it is relatively atomic as opposed to such non-atomic Elements as Sub-Processes.

On execution, the User Task element is instantiated as a User Task that appears in the list of Tasks of one or multiple Actors.

If a User Task element defines a **`GroupID`**, it is displayed in Task lists of all users that are members of the group: any of the users can claim the Task. Once claimed, the Task disappears from the Task lists of the other users.

Note that User Task is implemented as a domain-specific Tasks and serve as base for your custom Task (see Section 4.15.1, “Work Item Definition”).

Actors

comma-separated list of users who are entitled to perform the generated User Task

Comment

A comment associated with this User Task. The JBoss BPM Suite Engine does not use this field but business users can enter extra information about this task.

Content

The data associated with this task. This attribute does not affect TaskService’s behavior.

CreatedBy

name of the user or ID of the Process that created the task

GroupID

comma-separated list of groups who are entitled to perform the generated User Task

Locale

locale the Element is defined for. This was intended to support internationalization (i18n), but this property is not used by the JBoss BPM Suite engine at the moment.

Notifications

Definition of notification applied on the Human Task (see Section A.9.5.3, “Notification”)

Priority

Integer value defining the User Task priority (the value influences the User Task ordering in the user Task list and the simulation outcome)

Reassignment

Definition of escalation applied on the Human Task (see Section A.9.5.2, “Reassignment”)

ScriptLanguage

One of **Java** or **MVEL**.

Skippable

Boolean value that defines if the User Task can be skipped (if **true**, the actor of the User Task can decide not to complete it and the User Task is never executed)

Task Name

Name of the User Task generated on runtime (displayed in the Task List of Business Central)

Note that any other displayed attributes are used by features not restricted to the User Task element and are described in the chapters dealing with the particular mechanism.

A.9.5.1. User Task lifecycle

When a User Task element is encountered during Process execution, a User Task instance is created. The User Task instance execution is preformed by the User Task service of the Task Execution Engine (see the *Red Hat JBoss BPM Suite Administration and Configuration Guide*). The Process instance leaves the User Task element and continues the execution only when the associated User Task has been completed or aborted.

When the Process instance enters the User Task element, the User Task is the **Created** stage. This is usually a transient state and the User Task enters the **Ready** state immediately: the User Task appears in the Task Lists of all actors that are allowed to execute the task. As soon as one of the actors claims the User Task to indicate they are executing it, the User Task becomes **Reserved**. If a User Task has only one potential actor, it is automatically assigned to that actor upon creation. When the user who has claimed the User Task starts the execution, the User Task status changes to **InProgress**. On completion, the status changes to **Completed** or **Failed** depending on the execution outcome.

Note that the User Task lifecycle can include other statuses if the User Task is reassigned (delegated or escalated), revoked, suspended, stopped, or skipped. For further details, on the User Task lifecycle see the Web Services Human Task specification.

A.9.5.2. Reassignment

The reassignment mechanism is the mechanism implementing the escalation and delegation capabilities for User Tasks, that is, automatic reassignment of a User Task to another actor or group after a User Task has remained inactive for a certain amount of time.

Reassignment can be defined to take place either if the given User Task is for a given time in either of the following states:

- not started: **READY** or **RESERVED**
- not completed: **IN_PROGRESS**

When the conditions defined in the reassignment are met, the User Task is reassigned to the users or groups defined in the reassignment. If the actual owner is included in the new users or groups definition, the User Task is reset and reset to the READY state.

Reassignment is defined in the Reassignment property of User Task elements. The property can take an arbitrary number of reassignment definitions with the following parameters:

- **Users**: comma-separated list of user IDs that are reassigned to the task on escalation (Strings or expressions #{user-id})
- **Groups**: comma separated list of group IDs that are reassigned to the task on escalation (Strings or expressions #{group-id})
- **Expires At**: time definition when escalation is triggered (String values and expressions #{expiresAt}; for information on time format, see Section A.5, “Timing”)
- **Type**: state the task needs to be in at the given Expires At time so that the escalation is triggered.

A.9.5.3. Notification

The Notification mechanism provides the capability to send an e-mail notification if a User Task is at the given time in one of the following states:

- not started: **READY** or **RESERVED**
- not completed: **IN_PROGRESS**

Notification is defined in the Notification property of User Task elements. The property can take an arbitrary number of notification definitions with the following parameters:

- **Type**: state the User Task needs to be in at the given Expires At time so that the notification is triggered
- **Expires At**: time definition when notification is triggered (String values and expressions #{expiresAt}; for information on time format, see Section A.5, “Timing”)
- **From**: user or group ID of users used in the From field of the email notification message (Strings or expressions)
- **To Users**: comma-separated list of user IDs the notification is to be sent to (Strings or expressions #{user-id})
- **To Groups**: comma separated list of group IDs the notification is to be sent to (Strings or expressions #{group-id})
- **Reply To**: user or group ID that receives any replies to the notification (Strings or expressions #{group-id})
- **Subject**: subject of the email notification (Strings or expressions)

- **Body**: body of the email notification (Strings and expression)

Available variables
Notification can reference Process variables (**#{processVariable}**) and Task variables (**${taskVariable}**).

In addition to custom Task variables, the notification mechanism can make use of the following local Task variables:

- **taskId**: internal ID of the User Task instance
- **processInstanceId**: internal ID of Task's parent Process instance
- **workItemId**: internal ID of a work item that created the User Task
- **processSessionId**: knowledge session ID of the parent Process instance
- **owners**: list of users and groups that are potential owners of the User Task
- **doc**: map that contains regular task variables

Example A.9. Body of notification with variables

```
<html>
 <body>
  <b>${owners[0].id} you have been assigned to a task (task-id
${taskId})</b><br>
  You can access it in your task
  <a href="http://localhost:8080/jbpm-
console/app.html#errai_ToolSet_Tasks;Group_Tasks.3">inbox</a><br/>
  Important technical information that can be of use when working on
it<br/>
  - process instance id - ${processInstanceId}<br/>
  - work item id - ${workItemId}<br/>

  <hr/>

  Here are some task variables available
  <ul>
   <li>ActorId = ${doc['ActorId']}</li>
   <li>GroupId = ${doc['GroupId']}</li>
   <li>Comment = ${doc['Comment']}</li>
  </ul>
  <hr/>
  Here are all potential owners for this task
  <ul>
  $foreach{orgEntity : owners}
   <li>Potential owner = ${orgEntity.id}</li>
  $end{}
  </ul>

  <i>Regards from jBPM team</i>
 </body>
</html>
```

A.10. CONNECTING OBJECTS

A.10.1. Connecting Objects

Connecting object connect two elements. There are two main types of Connecting object:

- Sequence Flow, which connect Flow elements of a Process and define the flow of the execution (transport the token from one element to another)
- Association Flow, which connect any Process elements but have no execution semantics

A.10.2. Connecting Objects types

A.10.2.1. Sequence Flow

A Sequence Flow represents the transition between two Flow elements: it establishes an oriented relationship between Activities, Events, and Gateways and defines their execution order.

Condition Expression

A condition that needs to be true to allow the workflow to take the Sequence Flow
If a Sequence Flow has a Gateway element as its source, you need to define a Conditional Expression, which is evaluated before the Sequence Flow is taken. If false, the workflow attempts to switch to another Sequence Flow. If true, the Sequence Flow is taken.

When defining the condition in Java, make sure to return a boolean value:

```
return <expression resolving to boolean>;
```

Condition Expression Language

You can use either Java or Drools to define the Condition Expression.

AVAILABLE VARIABLES

When defining a Condition Expression, make sure to call process and global variables. You can also call the **`kcontext`** variable, which holds the Process instance information.

A.11. SWIMLANES

Swimlanes are a process element to visually group tasks related to one group or user. For example, you can create a Marketing task to group all User Tasks related to marketing activities into one Lane.

A.11.1. Lanes

> "A Lane is a sub-partition within a Process (often within a Pool)... " [18]

A Lane allows you to group some of the Process elements and define their common parameters. Note that a Lane may contain another Lane.

To add a new Lane, open up the **Swimlanes** menu item in the Object Library to show the Lane artifact. Drag and drop the Lane artifact to your Process Model. This artifact is a box in which you can add your User Tasks.

Lanes should be given distinguishing names and background colors to fully separate them into functional groups. You can do so by selecting a lane and opening up the Properties panel.

At runtime, Lanes auto-claim/assign task to user who has done another task of that Lane within the same process instance. This user must be eligible for claiming a task, that is, this user must be a potential owner. If a User Task doesn't have an actor or group assigned it marks the task as having no potential owners (and therefore, at runtime, the process will just stop).

For example, let's say there are two User Tasks (UT1 and UT2) located in the same Lane. UT1 and UT2 have group field set to the **`analyst`** value. When the Process is started, and UT1 is claimed/started/completed by an **`analyst`** user, UT2 gets claimed and assigned to the user who completed UT1. On the other hand, if only UT1 had the **`analyst`** group assigned, and UT2 had no user or group assignments, the process would stop after UT1 had been completed.

A.12. ARTIFACTS

A.12.1. Artifacts

Artifacts are considered any object depicted in the BPMN diagram that are not part of the Process workflow: they have no incoming or outgoing Flow objects.

The purpose of Artifacts is to provide additional information needed to understand the diagram.

A.12.2. Data Objects

Data Objects are visualizations of Process or Sub-Process variables. Note that not every Process or Sub-Process variable must be depicted as a Data Object in the BPMN diagram.

Also note, that Data Objects have the visualization properties and the variable properties.

---

[1] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[2] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[3] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[4] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[5] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[6] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[7] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[8] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[9] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[10] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[11] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[12] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[13] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[14] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[15] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[16] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[17] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

[18] Business Process Model and Notation (BPMN). Version 2.0, OMG Document Number: formal/2011-01-03 http://www.omg.org/spec/BPMN/2.0

APPENDIX B. SERVICE TASKS

Service task is a task that uses a service, such as a mail service, web service, or another service.

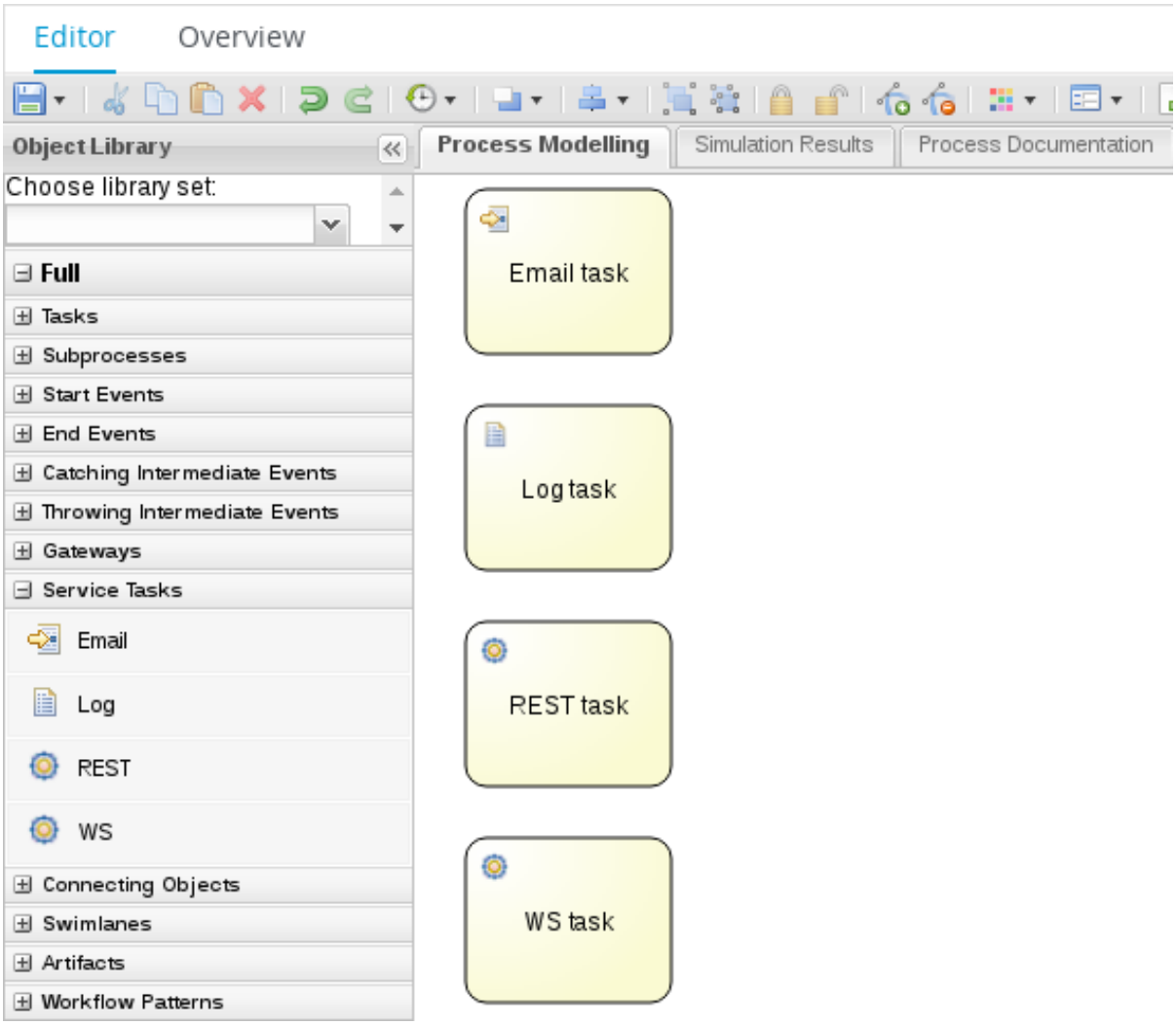
For the convenience of the user, Red Hat JBoss BPM Suite comes with predefined types of service tasks:

- Web Service task for invoking a web service from a BPMN2 process.
- Email task for sending emails through the setup mail server.
- Log task that calls the **`SystemOutWorkItemHandler`** class.
- REST task for sending REST calls.

Note that since the tasks extend the service task, their attributes are implemented as **`Assignments`**, **`DataInputSet`**, and **`DataOutputSet`**, not as separate properties.

In Process Designer, you can find these service tasks in the expanded **Object Library** on the left.

1. In Business Central, go to **Authoring** → **Project Authoring**.
2. In Project Explorer, locate the project and the respective process under **BUSINESS PROCESSES**.
3. Open the process in Process Designer and expand the **Object Library**.
4. Drag and drop the selected service task to the required position on the canvas.



For more information about service task attributes, see Section B.1, “WS Task Attributes”, Section B.2, “Email Task Attributes”, and Section B.3, “REST Task Attributes”.

If you require other task types, implement your task as instructed in Section 4.15, “Domain-Specific Tasks”.

B.1. WS TASK ATTRIBUTES

The Web Service task implements the **`WebServiceWorkItemHandler`** class. The Web Service task serves as a web service client with the web service response stored as **`String`**. To invoke a Web Service task from a BPMN process, the correct task type must be used.

B.1.1. Use Cases

B.1.1.1. Multiple Parameters

The Web Service task can be used to invoke a web service method with multiple parameters. To do so, the following changes must be made to the BPMN2 process definitions:

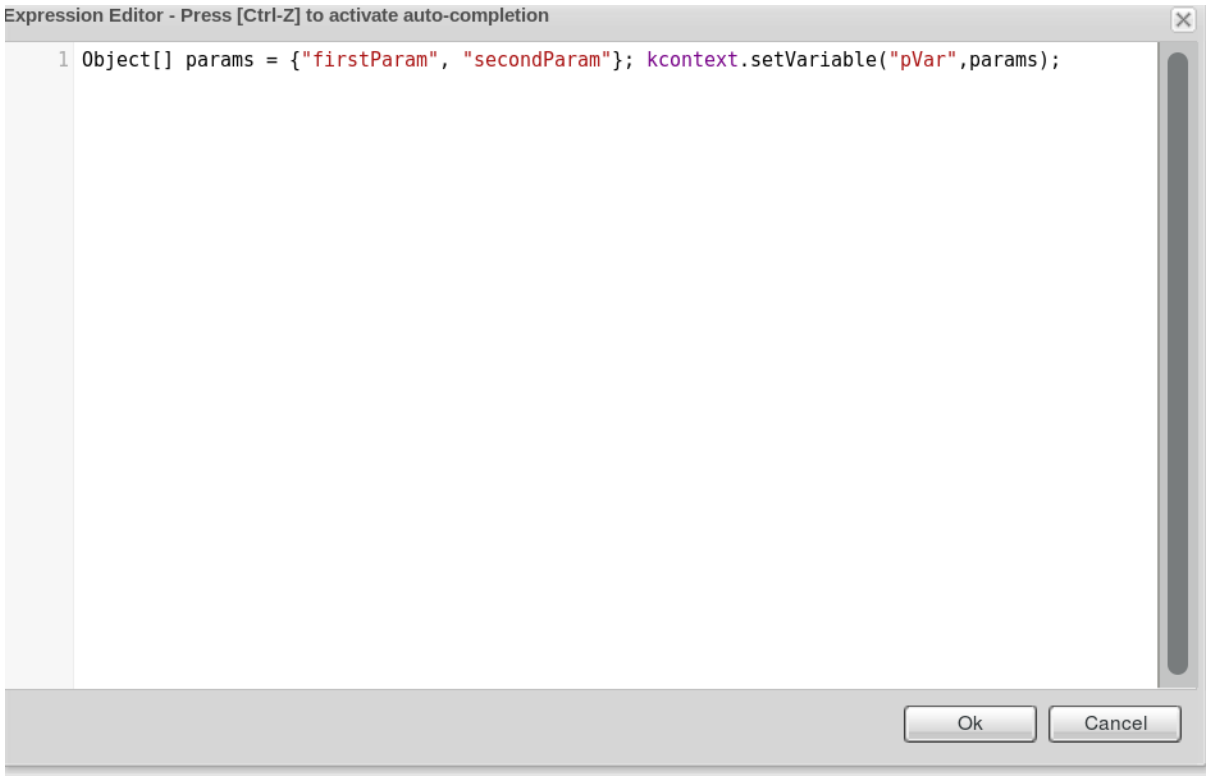
1. In the Process Designer, open the **Properties** panel and select the **Variable Definitions** property to create a process variable called **pVar** with the type **`Object[]`**.

2. Select the **WS** task in the Process Designer and modify the **WS Data I/O** table by editing **Assignments** in the **Properties** panel, or by clicking the **Edit Data I/O** icon. Change **WS Data Input** from **`Parameter:String`** to **`Parameter:Object[]`**.

3. Change the **Data Inputs and Assignments** section to map **`pVar`** to **`Parameter`**.

4. Define an array of **WS** parameters by selecting the **WS** task in the Process Designer and adding the following to the **On Entry Actions** in the **Properties** panel:

```
Object[] params = {"firstParam", "secondParam"};
kcontext.setVariable("pVar",params);
```



This will allow the web service to be invoked with two parameters.

B.1.1.2. Custom Objects

In addition to primitive object types, the WebService task can use more complex objects, such as **`Person`** or **`Employee`**.

To use custom objects, the following steps are required:

1. Create Custom Model objects using either the Data Modeler in Business Central or using an external tool, like Red Hat JBoss Developer Studio.

2. Use this Custom Model class in one of the WebService tasks.

3. Generate WSDL for this web service.

4. Use Red Hat JBoss Developer Studio to generate Java classes from the WSDL.

5. Create a **`.jar`** file that includes the model class generated from the WSDL file. Add **`kmodule.xml`** under the **`META-INF`** of the **`.jar`**.

6. Upload the **.jar** to the Artifact Repository. In Business Central, add it to the list of project's dependencies that includes the configured Web Service task. This Web Service task must have new classes generated, and cannot rely on the original ones.

7. Modify the project configuration using the **Deployment descriptor** as follows:

```
<kmodule xmlns="http://jboss.org/kie/6.0.0/kmodule"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <kbase name="defaultKieBase" default="true"
eventProcessingMode="stream" equalsBehavior="identity" packages="*">
    <ksession name="defaultKieSession" type="stateful"
default="true" clockType="realtime">
      <workItemHandlers>
        <workItemHandler type="new
org.jbpm.process.workitem.webservice.WebServiceWorkItemHandler(ksess
ion, runtimeManager.getEnvironment().getClassLoader())"
name="WebService"/>
      </workItemHandlers>
    </ksession>
  </kbase>
</kmodule>
```

The above configuration utilizes the **WebServiceWorkItemHandler**.

B.1.1.3. Web Service Task Example

This example demonstrates a process that obtains a weather forecast for given ZIP codes. The process looks as follows:

1. In the first human task, the process asks for ZIP codes.

2. Next, the result of the first human task is transformed into a collection that is used as an input for the service task with multiple instances.

3. Based on the input collection, the process creates several service task instances for querying the weather forecast service.

4. Once all the service task instances are completed, the result is logged to the console.

5. Another human task then shows the weather forecast for the chosen ZIP codes.

After the process instance is started, the user is prompted to select the mode of the service task: synchronous or asynchronous. Note that the difference between the two can be noticeable depending on the particular service.

Input Attributes

Endpoint

The endpoint location of the web service you want to invoke.

Parameter

The object or array to be sent for the operation.

Mode

Can be **SYNC**, **ASYNC**, or **ONEWAY**.

Interface

The name of a service, for example **Weather**.

Namespace

The namespace of the web service, such as **http://ws.cdyne.com/WeatherWS/**.

URL

The web service URL, such as http://wsf.cdyne.com/WeatherWS/Weather.asmx?WSDL.

Operation

The method name to call.

Output Attributes

Result

An object with the result.

B.2. EMAIL TASK ATTRIBUTES

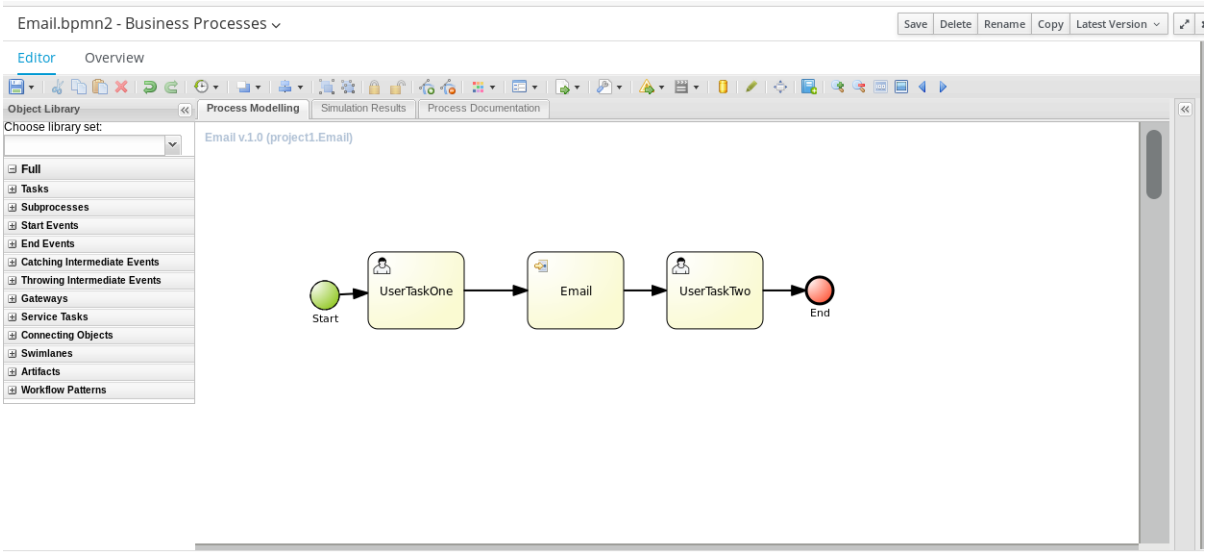
The Email task sends an email based on the task properties.

Registering Email Task in Business Central

Email task is not registered by default in Business Central, and therefore must be registered by the user.

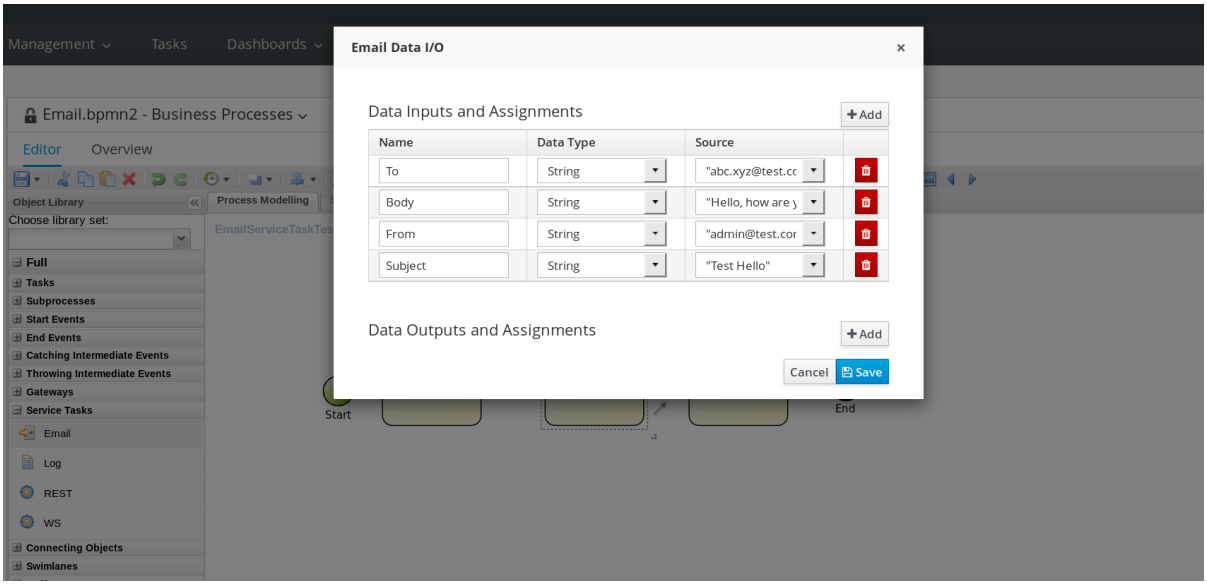
Follow the procedure below to configure Business Central to use Email service task.

1. Design a BPMN2 process definition in the Process Designer of Business Central. Add an Email Service task to the workflow.



2. Select the **Email Service Task** and open the **Properties** panel.

3. Open **Assignments** and fill in the **To**, **From**, **Subject**, and **Body** properties, and any other relevant input attributes.



Alternatively, values can be mapped to properties using Process Variable to Task Variable mapping assignments.

From the Process Designer, open the **Properties** panel and select the **`Variable Definitions`** property to map variables.

Registering EmailWorkItemHandler

`EmailWorkItemHandler` is the work item handler implementation of the Email Service task. The Email work item is included in the work item definition file by default, however **`EmailWorkItemHandler`** is not a part of the default **`kie-deployment-descriptor.xml`** file, and therefore must be explicitly registered by the user.

To register **`EmailWorkItemHandler`**:

1. Open the Project Editor and click **Project Settings: Project General Settings → Deployment descriptor** from the menu.

2. Scroll down to the Work Item handlers list and click **Add** to add the **`EmailWorkItemHandler`** to the list. For example:

```
new
org.jbpm.process.workitem.email.EmailWorkItemHandler("localhost","25
","me@localhost","password")
```

Alternatively, email server parameters can be supplied using a constructor in the **`ProcessMain.java`** file:

```
EmailWorkItemHandler emailWorkItemHandler = new
EmailWorkItemHandler("localhost", "1125", "", "",true);
ksession.getWorkItemManager().registerWorkItemHandler("Email",
emailWorkItemHandler );
```

Configuring Deadline

The Deadline email feature can be configured in two ways:

1. **Mail Session on Container Level**

With this method, the Deadline email feature uses **EmailSessionProducer** to look up the **mail/jbpmMailSession** via JNDI. The following example is for Red Hat JBoss EAP 6 **standalone.xml**:

```
<system-properties>
...
  <property name="org.kie.mail.session"
value="java:jboss/mail/mail/jbpmMailSession"/>
...
</system-properties>
...
<subsystem xmlns="urn:jboss:domain:mail:1.2">
  <mail-session name="default" jndi-name="mail/jbpmMailSession" >
    <smtp-server outbound-socket-binding-ref="mail-smtp" tls="true">
      <login name="email@gmail.com" password="___"/>
    </smtp-server>
  </mail-session>
</subsystem>
...
<outbound-socket-binding name="mail-smtp">
  <remote-destination host="smtp.gmail.com" port="587"/>
</outbound-socket-binding>
```

1. **Using email.properties**
   If the **mail/jbpmMailSession** is not found, Red Hat JBoss BPM Suite searches for **/email.properties** in the class path. The content should look like the following:

```
mail.smtp.host=localhost
mail.smtp.port=25
mail.from=xxx@xxx.com
mail.replyto=xxx@xxx.com
```

Input Attributes

The following parameters are required by default:

To

The email address of the email recipient. Separate multiple addresses by a semicolon (**;**).

From

The email address of the sender of the email.

Subject

The subject of the email.

Body

The HTML body of the email.

The following parameters are optional, and can be configured by mapping values assigned to these properties using **Process Variable to Task Variable** mapping in **Assignments**:

Reply-To

Sets the reply recipient address to the **From** address of the received message. Separate multiple addresses by a semicolon (**;**).

Cc

The email address of the carbon copy recipient. Separate multiple addresses by a semicolon (`;`).

Bcc

The email address of the blind carbon copy recipient. Separate multiple addresses by a semicolon (`;`).

Attachments

The URL of the files you want to attach to the email. Multiple attachments can be added to the email using a comma (`,`) to separate each URL in the list.

Debug

A boolean value related to the execution of the Email work item. For example,

```
"Success" = true
```

The Email task is completed immediately and cannot be aborted.

B.3. REST TASK ATTRIBUTES

The REST task performs REST calls and outputs the response as an object.

`RestWorkItemHandler` is capable of interacting with the REST service, and supports both types of services:

- *Secured*: requires authentication.
- *Open*: does not require authentication.

Authentication methods currently supported are:

- **`BASIC`**
- **`FORM_BASED`**

Authentication information can be given on handler initialization and can be overridden using work item parameters. All other configuration options must be given in the work item parameters map:

Input Attributes

Url

Target URL to be invoked. This attribute is mandatory.
It is often necessary to configure the URL attribute with an expression. This gives you the ability to change the URL dynamically throughout the runtime. For example:

```
http://DOMAIN:PORT/restService/getCars?brand=CAR_BRAND
```

In the provided example, **`CAR_BRAND`** will be replaced by the value of the**`carBrand`** variable.

Method

The method of the request, such as GET, POST, or similar. The default method is GET.

ContentType

The data type in case of sending data. This attribute is mandatory for POST and PUT requests.

Content

The data you want to send. This attribute is mandatory for POST and PUT requests.

ConnectTimeout

The connection timeout. The default value is 60 seconds.

ReadTimeout

The timeout on response. The default value is 60 seconds.
It is possible to modify the default timeout configuration shown below:

```
Integer connectTimeout = getParamAsInt(params.get("ConnectTimeout"));
if (connectTimeout==null) connectTimeout = 60000;
Integer readTimeout = getParamAsInt(params.get("ReadTimeout"));
if (readTimeout==null) readTimeout = 60000;
```

Username

The user name for authentication. This attribute overrides the handler initialization user name.

Password

The password for authentication. This attribute overrides the handler initialization password.

User name and password for basic authentication can be passed at construction time using the following:

```
RESTWorkItemHandler(String username, String password);
```

AuthUrl

The URL that is handling authentication when using the **`AuthenticationType.FORM_BASED`** authentication method.
Use the following constructor for **`FORM_BASED`** authentication:

```
public RESTWorkItemHandler(String username, String password, String
authUrl, ClassLoader classLoader) {
  this();
  this.username = username;
  this.password = password;
  this.type = AuthenticationType.FORM_BASED;
  this.authUrl = authUrl;
  this.classLoader = classLoader;
}
```

The following is an example of how the constructor must be used in **Deployment descriptor**:

```
new
org.jbpm.process.workitem.rest.RESTWorkItemHandler("username","password","
http://mydomain.com/my-j-security-check-url",classLoader)
```

IMPORTANT

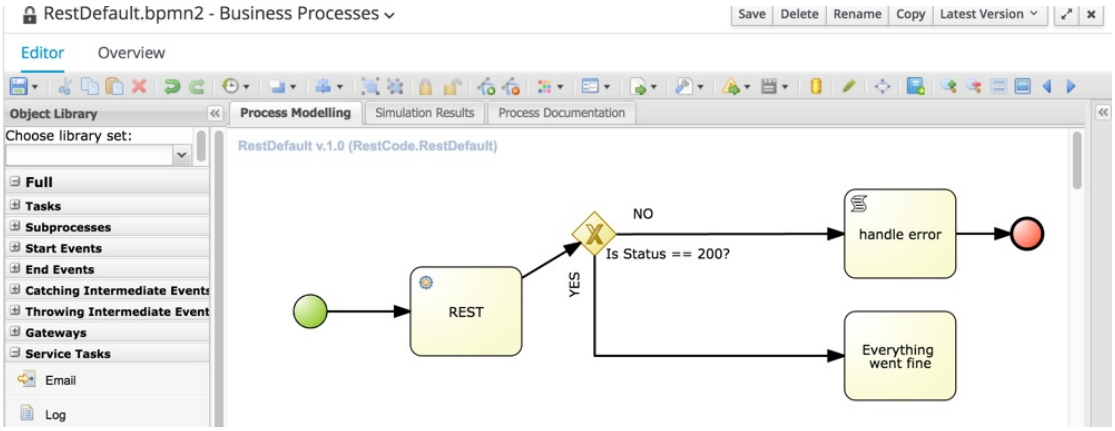
AuthUrl configuration requires the typical implementation for**FORM_BASED** authentication in Java EE servers, and therefore should point to the **j_security_check** URL. Similarly, inputs for user name and password must be bound to **j_username** and **j_password** when using **FORM_BASED** authentication, otherwise authentication may fail.

HandleResponseErrors

An optional parameter that instructs the handler to throw errors in case of unsuccessful response codes.
HandleResponseErrors can be handled in two ways:

1. **Process Definition**

   a. **Status**: When **RESTWorkItemHandler** produces a **Status** output variable that includes an HTTP response code. This can be mapped to a process variable and used in a XOR gateway to determine the service outcome.



   b. **StatusMsg**: The output variable**StatusMsg** includes additional messages sent by the server, and is filled only when the HTTP Code is not between 200 and 300.

2. **Using a Boundary Event**
   This can be enabled by setting the REST work item input variable **HandleResponseErrors** to the value**true**.

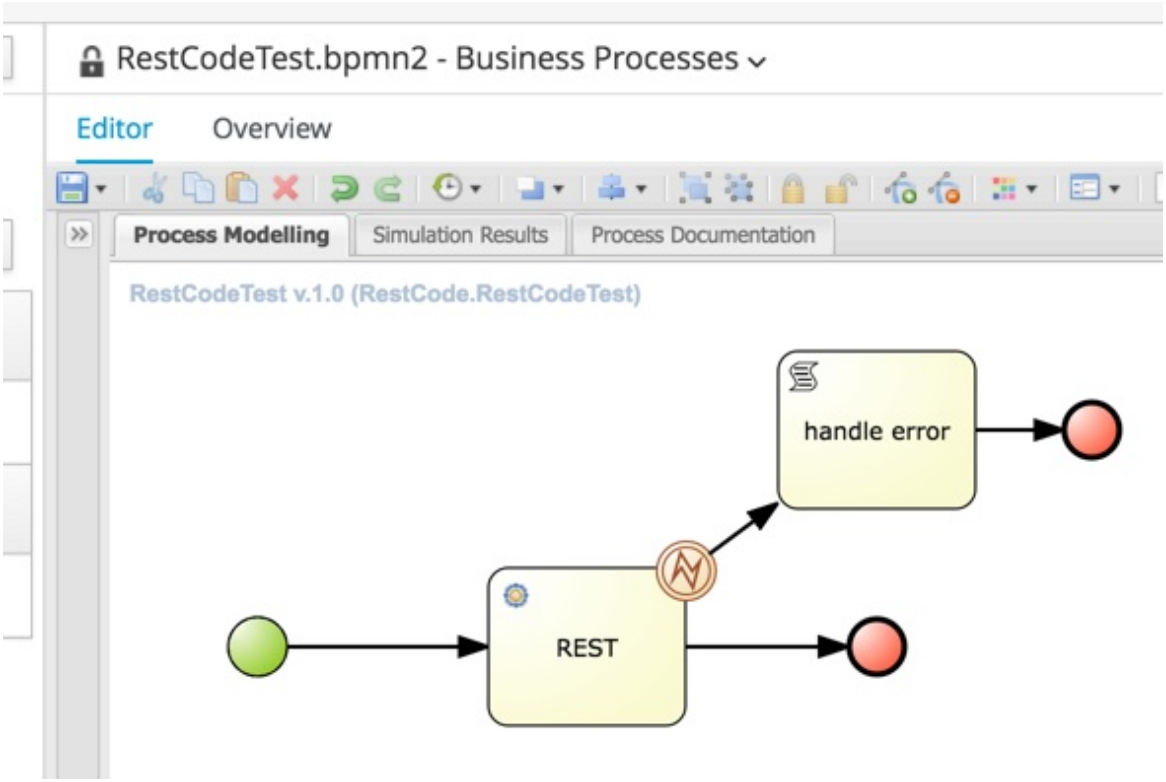
IMPORTANT

The **HandleResponse** must have a valid boolean expression or be left empty (which will default to **false**), otherwise it will fail.

When the REST work item input variable **HandleResponseErrors** is set to**true**, the **RESTWorkItemHandler** will, upon receiving an HTTP response code outside of the 200-300 interval, throw the following Java exception:

```
public RESTServiceException(int status, String response, String
endpoint) {
  super("Unsuccessful response from REST server (status " + status
+", endpoint " + endoint +", response " + response +"");
```

With the option enabled, this error can be caught using a boundary event:



The provided example includes:

- A **`WorkItemHandlerRuntimeException restError`** process variable.
- A **`WorkItemHandlerRuntimeException BoundaryError`** event-defined output variable that has been mapped to the **`restError`** process variable.
- A Script task that includes the following code:

  ```
  org.jbpm.process.workitem.rest.RESTServiceException x =
  (org.jbpm.process.workitem.rest.RESTServiceException)
  restError.getCause().getCause();
  ```

  This code allows **`RestServiceException`** to be extracted from **`WorkItemHandlerRuntimeException`**. Using **`RestServiceException`** provides access to the following methods:

- **`getResponse`**
- **`getStatus`**
- **`getEndpoint`**
  The next line in the Script task is:

  ```
  System.out.println("response:"+x.getResponse());
  ```

  This provides the full error message as returned by the server.

Output Attributes

Result

The result returned by the REST service.

ResultClass

This attribute determines the class to which the value from the **`Result`** attribute will be converted. The default value is **`String`**.

Status

This variable will always be filled. It will either have a value from interval 200 to 300 if successful, or an error response code if the request was unsuccessful.

StatusMsg

If the service returned an error response, this will contain the error response message.

All output attributes are **`String`** by default.

APPENDIX C. SIMULATION DATA

C.1. PROCESS

Simulation attributes

Base currency

currency to be used for simulation

Base time unit

time unit to apply to all time definitions in the Process

C.2. START EVENT

Simulation attributes

Wait time

time to wait as to simulate the execution of the Start Event

Time unit

time unit to be used for the wait time property

C.3. CATCHING INTERMEDIATE EVENTS

Simulation attributes

Wait time

time to wait as to simulate the execution of the Catching Intermediate Event

Time unit

time unit to be used for the wait time property

C.4. SEQUENCE FLOW

Simulation attributes

Probability

probability the Flow is taken in percent
The probability value is applied only if the Flow’s source element is a Gateway and there are multiple Flow elements leaving the Gateway. When defining Flow probabilities, make sure their sum is 100.

C.5. THROWING INTERMEDIATE EVENTS

Simulation attributes

Distribution type

distribution type to be applied to the processing time values

C.6. HUMAN TASKS

Simulation attributes

Cost per time unit

costs for every time unit lapsed during simulation

Currency

currency of the cost per unit property

Staff availability

number of actors available to work on the given Task

> **Example C.1. Staff availability impact**
>
> Let's assume a simulation of 3 instances of a Process. A new instance is created every 30 minutes. The Process contains a None Start Event, a Human Task, and a Simple End Event. The Human Task takes 3 hours to complete, the **`Working hours`** is set to **`3`**. We have only one person to work on the Human Tasks, that is **`Staff availability`** is set to **`1`**. That results in the following:
>
> - The Human Task generated by the first Process instance will be executed after 3 hours;
> - The Human Task generated by the second Process instance will be executed in approx. 6 hours (the second Process instance is created after 30 minutes; however, the actor is busy with the first Task; he becomes available only after another 2.5 hours, and takes 3 hours to execute the second Task).
> - The Human Task generated by the third Process instance will be executed in approx. 9 hours (the second Human Task instance is finished after 3 hours; the actor needs another 3 hours to complete the third Human Task).

Working hours

time period for simulation of the Task execution

C.7. SERVICE TASKS

Simulation attributes

Cost per time unit

costs for every time unit lapsed during simulation

Currency

currency of the cost per unit property

Distribution type

distribution type to be applied to the element execution time

C.8. END EVENTS

Simulation attributes

Distribution Type

distribution type to be applied to the element execution time

C.9. DISTRIBUTION TYPES

The **`Distribution type`** property defines the distribution of possible time values (scores) of Process elements.

The elements might use one of the following score distribution types on simulation:

- **`normal`**: bell-shaped, symmetrical
- **`uniform`**: rectangular distribution; every score (time period) is applied the same number of times
- **`Poisson`**: negatively skewed normal distribution

C.9.1. Normal

The element values are picked based on the normal distribution type, which is bell-shaped and symmetrical.

Normal distribution type

Processing time (mean)

mean processing time the element needs to be processed (in time unit defined in the time units property)

Standard deviation

standard deviation of the processing time (in time unit defined in the time units property)

Time unit

time unit to be used for the mean processing time and standard deviation values

C.9.2. Uniform

The Uniform distribution or rectangular distribution returns the possible values with the same levels of probability.

Uniform distribution type

Processing time (max)

maximum processing time of the element

Processing time (min)

minimum processing time of the element

Time unit

time unit to be used for the processing time values

C.9.3. Poisson

The Poisson distribution returns the possible values similarly as normal distribution; however, the distribution is negatively skewed, not symmetrical.The mean and the variant

are equal.

Poisson distribution type

Processing time (mean)

mean time for the element processing (in time unit defined in the time units property)

Time unit

time unit to be used for the mean processing time

APPENDIX D. REVISION HISTORY

Note that revision numbers relate to the edition of this manual, not to version numbers of Red Hat JBoss BRMSBPM Suite.

Revision 6.3.0-17 **Mon Mar 20 2017** **Tomas Radej**
Rebuilt.

Revision 6.3.0-16 **Wed Feb 22 2017** **Tomas Radej**
Rebuilt.

Revision 6.3.0-15 **Fri Dec 23 2016** **Tomas Radej**
Rebuilt.

Revision 6.3.0-14 **Mon Nov 28 2016** **Tomas Radej**
Rebuilt.

Revision 6.3.0-13 **Wed Oct 5 2016** **Tomas Radej**
Rebuilt.

Revision 6.3.0-12 **Mon Oct 3 2016** **Tomas Radej**
Built for release 6.3.3.

Revision 6.3.0-11 **Thu Sep 15 2016** **Tomas Radej**
Rebuilt.

Revision 6.3.0-10 **Thu Sep 15 2016** **Tomas Radej**
Published the AsciiDoc version of the docs.

Revision 6.3.0-7 **Thu Jul 14 2016** **Tomas Radej**
Updated documentation with release 6.3.1.

Revision 6.3.0-6 **Thu Jun 2 2016** **Marek Czernek**
Releasing the newest documentation.

Revision 6.3.0-5 **Thu May 5 2016** **Tomas Radej**
Built with live links.

Revision 6.3.0-4 **Thu May 5 2016** **Tomas Radej**
Bootstrapping links.

Revision 6.3.0-3 **Thu Apr 28 2016** **Tomas Radej**
All books rebuilt.

Revision 6.3.0-2 **Thu Apr 28 2016** **Tomas Radej**
All books rebuilt.

Revision 6.3.0-1 **Thu Apr 28 2016** **Tomas Radej**
Initial build for release 6.3.0 of JBoss BPM SuiteJBoss BRMS.