



Red Hat JBoss BPM Suite 6.4

User Guide

The User Guide for Red Hat JBoss BPM Suite

Red Hat JBoss BPM Suite 6.4 User Guide

The User Guide for Red Hat JBoss BPM Suite

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

Emily Murphy

Gemma Sheldon

Michele Haglund

Mikhail Ramendik

Stetson Robinson

Vidya Iyengar

Legal Notice

Copyright © 2019 Red Hat, Inc.

The text of and illustrations in this document are licensed by Red Hat under a Creative Commons Attribution–Share Alike 3.0 Unported license ("CC-BY-SA"). An explanation of CC-BY-SA is available at

<http://creativecommons.org/licenses/by-sa/3.0/>

. In accordance with CC-BY-SA, if you distribute this document or an adaptation of it, you must provide the URL for the original version.

Red Hat, as the licensor of this document, waives the right to enforce, and agrees not to assert, Section 4d of CC-BY-SA to the fullest extent permitted by applicable law.

Red Hat, Red Hat Enterprise Linux, the Shadowman logo, the Red Hat logo, JBoss, OpenShift, Fedora, the Infinity logo, and RHCE are trademarks of Red Hat, Inc., registered in the United States and other countries.

Linux[®] is the registered trademark of Linus Torvalds in the United States and other countries.

Java[®] is a registered trademark of Oracle and/or its affiliates.

XFS[®] is a trademark of Silicon Graphics International Corp. or its subsidiaries in the United States and/or other countries.

MySQL[®] is a registered trademark of MySQL AB in the United States, the European Union and other countries.

Node.js[®] is an official trademark of Joyent. Red Hat is not formally related to or endorsed by the official Joyent Node.js open source or commercial project.

The OpenStack[®] Word Mark and OpenStack logo are either registered trademarks/service marks or trademarks/service marks of the OpenStack Foundation, in the United States and other countries and are used with the OpenStack Foundation's permission. We are not affiliated with, endorsed or sponsored by the OpenStack Foundation, or the OpenStack community.

All other trademarks are the property of their respective owners.

Abstract

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

Table of Contents

CHAPTER 1. INTRODUCTION	10
1.1. USE CASE: PROCESS-BASED SOLUTIONS IN THE LOAN INDUSTRY	10
1.2. COMPONENTS	11
1.3. RED HAT JBOSS BPM SUITE AND BRMS	11
1.4. BUSINESS CENTRAL	11
1.4.1. Business Central Environment	12
1.4.2. Perspectives	13
1.4.3. Embedding Business Central	14
CHAPTER 2. BASIC CONCEPTS	16
PART I. MODELING	18
CHAPTER 3. PROJECT	19
3.1. CREATING AN ORGANIZATIONAL UNIT	19
Creating an Organizational Unit in Business Central	19
Creating an Organizational Unit Using the kie-config-cli Tool	20
Creating an Organizational Unit Using the REST API	20
3.2. CREATING A REPOSITORY	20
Creating a Repository in Business Central	20
Creating a Repository Using the kie-config-cli Tool	21
Creating a Repository Using the REST API	21
3.3. CLONING A REPOSITORY	22
Cloning a Repository in Business Central	22
Cloning a Repository Using the REST API	24
3.4. CREATING A PROJECT	24
Creating a Project in Business Central	24
Creating a Project Using the REST API	26
3.5. ADDING DEPENDENCIES	26
3.6. DEFINING KIE BASES AND SESSIONS	27
Defining KIE Bases and Sessions in the Project Editor	27
Defining KIE Bases and Sessions in kmodule.xml	28
3.7. CREATING A RESOURCE	28
3.8. ASSET METADATA AND VERSIONING	29
Metadata Management	29
Version Management	30
3.9. FILTERING ASSETS BY TAG	30
3.10. ASSET LOCKING SUPPORT	31
3.11. PROCESS DEFINITION	32
3.11.1. Creating a Process Definition	32
3.11.2. Importing a Process Definition	33
3.11.3. Importing jPDL 3.2 to BPMN2	34
CHAPTER 4. PROCESS DESIGNER	36
4.1. CONFIGURING AUTOMATIC SAVING	37
4.2. DEFINING PROCESS PROPERTIES	37
4.3. DESIGNING PROCESS	38
4.3.1. Copying Elements	39
4.3.2. Aligning Elements	40
4.3.3. Changing Element Layering	40
4.3.4. Bending Connection Elements	40
4.3.5. Resizing Elements	41

4.3.6. Grouping Elements	41
4.3.7. Locking Elements	41
4.3.8. Changing Color Scheme	41
4.3.9. Recording local history	42
4.3.10. Enlarging and shrinking canvas	42
4.3.11. Validating a Process	42
4.3.12. Correcting Invalid Processes	43
4.4. EXPORTING PROCESS	43
4.5. PROCESS ELEMENTS	45
4.5.1. Generic Properties of Visualized Process Elements	45
4.5.2. Defining Process Element Properties	45
4.6. BUSINESS PROCESS SAVE POINTS	46
4.7. FORMS	47
4.7.1. Defining Process form	47
4.7.2. Defining Task form	48
4.7.3. Defining form fields	48
4.8. FORM MODELER	48
4.8.1. Creating a Form in Form Modeler	49
4.8.2. Opening an Existing Form in Form Modeler	50
4.8.3. Setting Properties of a Form Field in Form Modeler	50
4.8.4. Configuring a Process in Form Modeler	51
4.8.5. Generating Forms from Task Definitions	51
4.8.6. Editing Forms	52
4.8.7. Moving a Field in Form Modeler	52
4.8.8. Adding New Fields to a Form	53
4.8.9. Configuring Fields of a Form	57
4.8.10. Creating Subforms with Simple and Complex Field Types	57
4.8.11. Enabling Document Attachments in a Form or Process	60
4.8.11.1. Using a Custom Document Marshalling Strategy for a Content Management System (CMS)	63
4.8.12. Rendering Forms for External Use	68
4.8.12.1. JavaScript Library for Form Reuse	68
Blueprint for using the JavaScript Library	68
Full list of available methods in the JavaScript Library	69
4.9. VARIABLES	71
4.9.1. Global Variables	71
4.9.1.1. Creating Global Variables	72
4.9.1.2. Process variables	73
4.9.2. Local Variables	74
4.9.2.1. Accessing Local Variables	74
4.9.3. Setting Process Variables From Business Rule Task	75
4.9.3.1. Mapping Process Variables through Business Rule Task Assignments field	75
4.9.3.2. Mapping Process Variables through WorkflowProcessInstance	75
4.10. ACTION SCRIPTS	76
4.11. INTERCEPTOR ACTIONS	77
4.12. ASSIGNMENT	77
4.12.1. Data I/O Editor	77
4.12.2. Data I/O Editor Example	78
4.13. CONSTRAINTS	79
4.14. DOMAIN-SPECIFIC TASKS	81
4.14.1. Work Item Definition	82
4.14.2. Creating Custom Work Item Definition	83
JBoss Developer Studio Process Designer	83
Web Process Designer	83

4.14.3. Work Item Handler	84
4.14.4. Registering Work Item handler in Business Central	86
4.14.5. Registering Work Item Handler Outside of Business Central	87
4.15. SERVICE REPOSITORY	88
4.15.1. Installing Services from Service Repository	89
Installing Services in Process Designer	89
Installing Services During Business Central Startup	90
4.15.2. Setting up Service Repository	90
Repository Configuration File	90
Work Item Configuration File	91
4.15.3. Retrieving Service Repository Information	92
4.16. ACTOR ASSIGNMENT CALLS	92
4.17. LDAP CONNECTION	93
4.17.1. Connecting to LDAP	94
4.18. EXCEPTION MANAGEMENT	94
Business exceptions	95
Technical exceptions	95
CHAPTER 5. DATA MODELS	96
5.1. DATA MODELER	96
5.2. AVAILABLE FIELD TYPES	97
5.3. ANNOTATIONS IN DATA MODELER	97
5.4. CREATING A DATA OBJECT	98
5.5. PERSISTABLE DATA OBJECTS	98
5.6. DATA OBJECT DOMAIN SCREENS	99
Drools & jBPM	99
Persistence	100
Advanced	102
5.7. CONFIGURING RELATIONSHIPS BETWEEN DATA OBJECTS	104
5.8. PERSISTENCE DESCRIPTOR	104
5.9. DEPLOYMENT DESCRIPTOR	105
CHAPTER 6. ADVANCED PROCESS MODELING	107
6.1. PROCESS MODELING OPTIONS	107
6.2. WORKFLOW PATTERNS	107
6.2.1. Defining workflow patterns	107
6.2.2. Changing workflow patterns for an existing project	108
CHAPTER 7. SOCIAL EVENTS	109
Follow User	109
Activity Timeline	109
PART II. SIMULATION AND TESTING	110
CHAPTER 8. PROCESS SIMULATION	111
8.1. PATH FINDER	111
8.2. SIMULATING PROCESSES	112
8.2.1. Defining Simulation Data on Elements	112
8.2.2. Running Process Simulations	112
8.2.3. Examining Simulation Results	113
8.2.3.1. Switching Between Graph Types	114
8.2.3.2. Filtering in Graphs	116
8.2.3.3. Viewing Graph Timeline	117
CHAPTER 9. TESTING	119

9.1. TEST SCENARIOS	119
9.2. CREATING A TEST SCENARIO	119
9.3. ADDITIONAL TEST SCENARIO FEATURES	123
PART III. PLUG-IN	127
CHAPTER 10. CREATING BPM PROJECT	128
CHAPTER 11. CREATING PROCESS	129
CHAPTER 12. CHECKING SESSION LOGS	130
PART IV. DEPLOYMENT AND RUNTIME MANAGEMENT	131
CHAPTER 13. DEPLOYING AND MANAGING PROJECTS	132
13.1. DEPLOYING A PROJECT	132
13.1.1. Duplicate GAV Detection	132
13.2. PROCESS MANAGEMENT	133
13.2.1. Process Definitions	134
13.2.2. Process Instances	134
13.2.2.1. Searching Process Instances by Partial Correlation Key	135
13.2.2.2. Searching Process Instances Based on Business Data	136
13.2.3. Creating a New Process Instance List	136
13.2.4. Aborting a Process instance	137
Aborting a Process instance using API	137
Aborting a Process instance from the Business Central	137
13.3. SIGNALING PROCESS INSTANCE	137
Signaling Process Instance Using API	137
Signaling Process Instance from Business Central	138
13.4. TASK MANAGEMENT	138
13.4.1. Tasks List	139
Task Client	139
13.4.2. Creating Custom Tasks Filters	140
13.4.3. Creating a User Task	141
13.4.4. Task Variables as Expressions	142
CHAPTER 14. LOGGING	144
CHAPTER 15. EXAMPLES	145
PART V. BAM	146
CHAPTER 16. RED HAT JBOSS DASHBOARD BUILDER	147
What is Business Activity Monitoring?	147
16.1. BASIC CONCEPTS	147
16.2. ACCESSING DASHBOARD BUILDER	147
16.3. PROCESS & TASK DASHBOARD	148
Tasks Dashboard	150
16.4. DATA SOURCES	150
16.4.1. Connecting to Data Sources	150
16.4.2. Security Considerations	151
16.4.3. Building a Dashboard for Large Volumes of Data	151
16.4.4. Data Providers	153
16.4.4.1. Creating Data Providers	153
16.4.5. Workspace	154
16.4.5.1. Creating Workspace	154

16.4.5.2. Configuring a default workspace	155
16.4.5.3. Pages	155
16.4.5.3.1. Creating Pages	155
16.4.5.3.2. Defining Page Permissions	156
16.4.5.4. Panels	156
16.4.5.4.1. Adding Panels	157
16.5. IMPORT AND EXPORT	158
16.5.1. Importing and Exporting Workspaces	158
16.5.2. Importing and Exporting KPIs	159
16.5.3. Importing Data Sources	161
16.6. DASHBOARD BUILDER DATA MODEL	163
CHAPTER 17. DATA SETS	167
17.1. MANAGING DATA SETS	167
17.2. CACHING	168
Client Cache	168
Backend Cache	168
17.3. DATA REFRESH	168
CHAPTER 18. MANAGEMENT CONSOLE	169
CHAPTER 19. GRAPHIC RESOURCES	170
Graphic Resources Definitions	170
19.1. WORKING WITH GRAPHIC RESOURCES	170
APPENDIX A. PROCESS ELEMENTS	171
CHAPTER 20. PROCESS	172
Runtime	172
CHAPTER 21. EVENTS MECHANISM	175
CHAPTER 22. COLLABORATION MECHANISMS	176
22.1. SIGNALS	176
22.1.1. Triggering Signals	176
Signalling External Deployments	177
22.1.2. Catching and Processing Signals	180
22.1.3. Triggering Signals Using API	180
22.2. MESSAGES	181
22.2.1. Sending Messages	181
22.2.2. Catching Messages	182
22.2.3. Sending Messages Using API	182
22.3. ESCALATION	183
Attributes	183
CHAPTER 23. TRANSACTION MECHANISMS	184
23.1. ERRORS	184
Attributes	184
23.2. COMPENSATION	184
CHAPTER 24. TIMING	186
CHAPTER 25. EVENT TYPES	188
25.1. START EVENT	188
25.1.1. Start Event types	188
25.1.1.1. None Start Event	188

25.1.1.2. Message Start Event	188
Attributes	189
25.1.1.3. Timer Start Event	189
Attributes	189
25.1.1.4. Escalation Start Event	189
Attributes	189
25.1.1.5. Conditional Start Event	190
Attributes	190
25.1.1.6. Error Start Event	190
Attributes	190
25.1.1.7. Compensation Start Event	190
25.1.1.8. Signal Start Event	190
Attributes	190
25.2. INTERMEDIATE EVENTS	190
25.2.1. Intermediate Events	190
25.2.2. Intermediate Event types	192
25.2.2.1. Timer Intermediate Event	192
Attributes	192
25.2.2.2. Conditional Intermediate Event	192
Attributes	192
25.2.2.3. Compensation Intermediate Event	193
25.2.2.4. Message Intermediate Event	193
Throwing Message Intermediate Event	193
Attributes	193
Catching Message Intermediate Event	193
Attributes	193
25.2.2.5. Escalation Intermediate Event	193
Throwing Escalation Intermediate Event	194
Attributes	194
Catching Escalation Intermediate Event	194
Attributes	194
25.2.2.6. Error Intermediate Event	194
25.2.2.6.1. Catching Error Intermediate Event	194
Attributes	194
25.2.2.7. Signal Intermediate Event	194
Throwing Signal Intermediate Event	194
Attributes	195
25.2.2.7.1. Catching Signal Intermediate Event	195
Attributes	195
25.3. END EVENTS	195
25.3.1. End Event types	196
25.3.1.1. Simple End Event	196
25.3.1.2. Message End Event	196
25.3.1.3. Escalation End Event	196
25.3.1.4. Terminate End Event	196
25.3.1.5. Throwing Error End Event	196
Attributes	196
25.3.1.6. Cancel End Event	196
25.3.1.7. Compensation End Event	196
25.3.1.8. Signal End Event	196
25.4. SCOPE OF EVENTS	197
CHAPTER 26. GATEWAYS	198

26.1. GATEWAYS	198
26.2. GATEWAY TYPES	198
26.2.1. Event-based Gateway	198
26.2.2. Parallel Gateway	199
26.2.3. Inclusive Gateway	199
Attributes	199
26.2.4. Data-based Exclusive Gateway	200
Attributes	200
CHAPTER 27. ACTIVITIES, TASKS AND SUB-PROCESSES	201
27.1. ACTIVITY	201
27.2. ACTIVITY MECHANISMS	201
27.2.1. Multiple Instances	201
27.2.2. Activity Types	201
27.2.2.1. Call Activity	201
Attributes	201
27.3. TASKS	202
27.3.1. None Task	202
27.3.2. Send Task	202
Attributes	202
27.3.3. Receive Task	202
Attributes	202
27.3.4. Manual Task	202
27.3.5. Service Task	202
27.3.5.1. Using Service Task to Invoke Web Service	203
27.3.5.2. Using Service Task to Invoke Java Method	205
27.3.6. Business Rule Task	208
Attributes	208
27.3.7. Script Task	209
Attributes	209
27.4. SUB-PROCESS	210
27.4.1. Embedded Sub-Process	210
27.4.2. AdHoc Sub-Process	211
Attributes	211
27.4.3. Multi-instance Sub-Process	211
Attributes	211
27.4.4. Event Sub-Process	212
27.5. USER TASK	212
Attributes	212
27.5.1. User Task lifecycle	213
27.5.2. Reassignment	213
27.5.3. Notification	214
Available variables	215
CHAPTER 28. CONNECTING OBJECTS	217
28.1. CONNECTING OBJECTS	217
28.2. CONNECTING OBJECTS TYPES	217
28.2.1. Sequence Flow	217
CHAPTER 29. SWIMLANES	218
29.1. LANES	218
CHAPTER 30. ARTIFACTS	219
30.1. ARTIFACTS	219

30.2. DATA OBJECTS	219
APPENDIX B. SERVICE TASKS: WS TASK, EMAIL TASK, REST TASK	220
CHAPTER 31. WS TASK	222
31.1. MULTIPLE PARAMETERS	222
31.2. CUSTOM OBJECTS	222
31.3. WEB SERVICE TASK EXAMPLE	223
Input Attributes	223
Output Attributes	224
CHAPTER 32. EMAIL TASK	225
Registering Email Task in Business Central	225
Registering EmailWorkItemHandler	225
Configuring Deadline	226
Input Attributes	226
CHAPTER 33. REST TASK	228
Input Attributes	228
Output Attributes	229
Handling REST Response Error	230
APPENDIX C. SIMULATION DATA	232
CHAPTER 34. PROCESS	233
Simulation Attributes	233
CHAPTER 35. ACTIVITIES	234
Simulation Attributes	234
CHAPTER 36. START EVENT	235
Simulation Attributes	235
CHAPTER 37. CATCHING INTERMEDIATE EVENTS	236
Simulation Attributes	236
CHAPTER 38. SEQUENCE FLOW	237
Simulation Attributes	237
CHAPTER 39. THROWING INTERMEDIATE EVENTS	238
Simulation Attributes	238
CHAPTER 40. HUMAN TASKS	239
Simulation Attributes	239
CHAPTER 41. END EVENTS	240
Simulation Attributes	240
CHAPTER 42. DISTRIBUTION TYPES	241
42.1. NORMAL	241
Normal Distribution Attributes	241
42.2. UNIFORM	241
Uniform Distribution Attributes	241
42.3. POISSON	241
42.3.1. Poisson Distribution Attributes	241
APPENDIX D. VERSIONING INFORMATION	242

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.

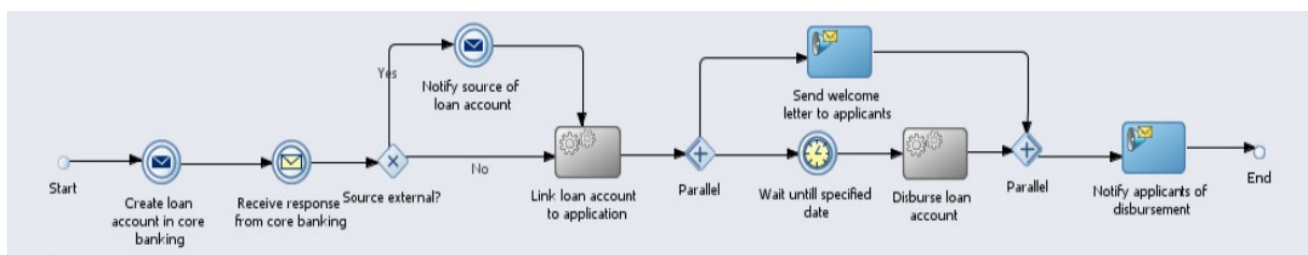
1.1. USE CASE: PROCESS-BASED SOLUTIONS IN THE LOAN INDUSTRY

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:

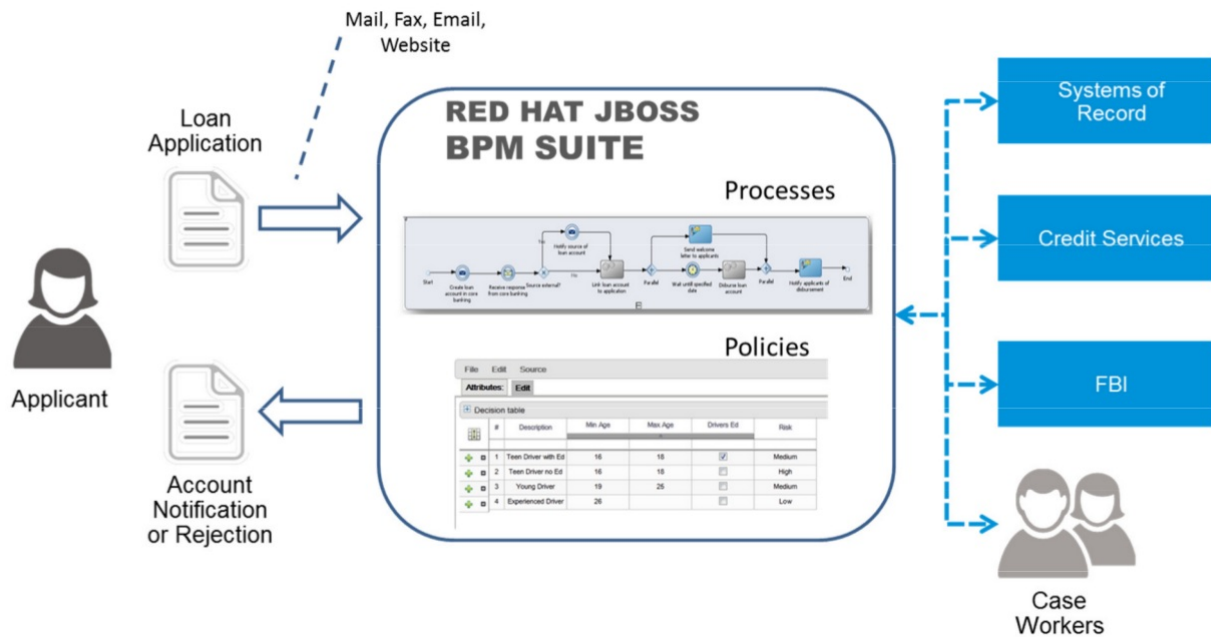
Figure 1.1. High-level loan application process flow



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 27.3.6, "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 (see [Section 13.4, "Task Management"](#)).
- *Process Manager* for managing process instances (see [Section 13.2.2, "Process Instances"](#)).
- *Dashboard Builder*, the BAM component, for monitoring and reporting (see [Chapter 16, 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 kjar) 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.

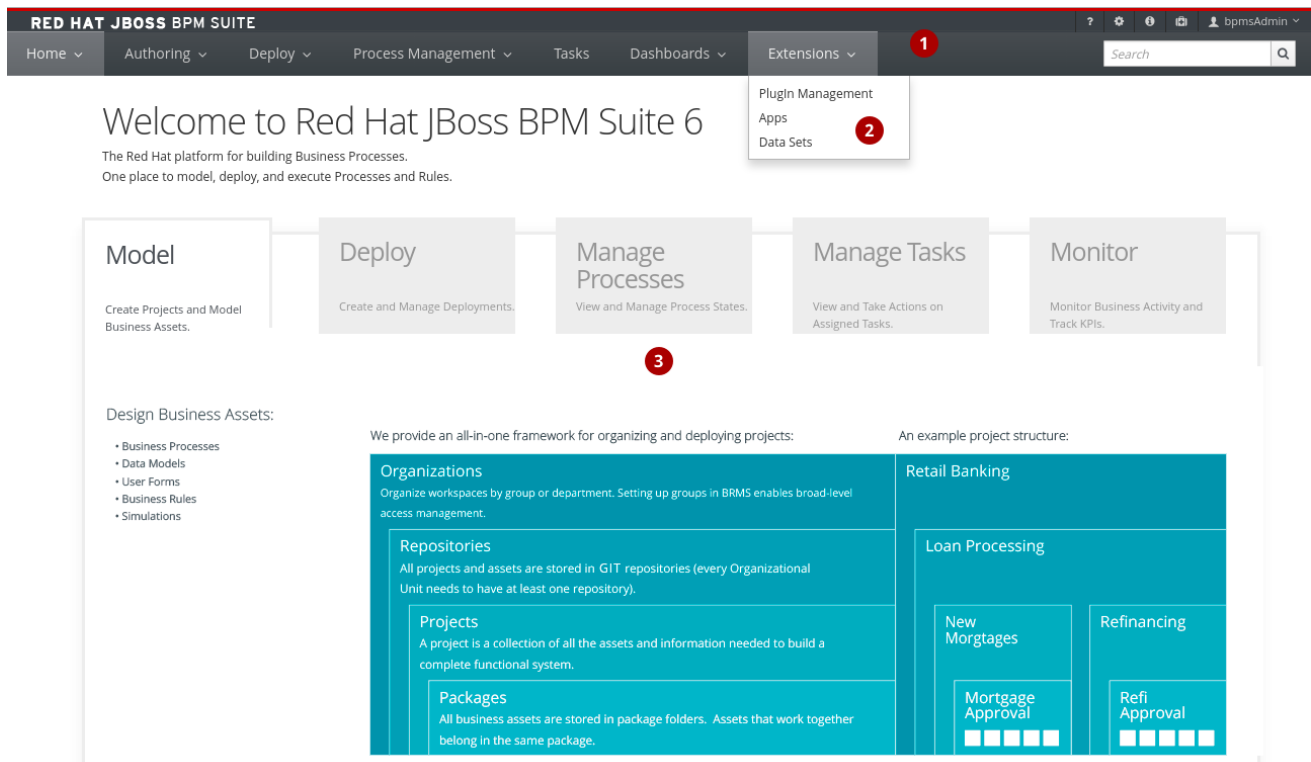
Business Central can be accessed from your web browser on **[https://\\$HOSTNAME/business-central](https://$HOSTNAME/business-central)** (for instances running on localhost **<https://localhost:8080/business-central>**).

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 with the overview of available repository structure, and information on available resources, such as, business process definitions, form definitions, and others.
 - The editor area on the right of the **Project Explorer** view, where the respective editor appears when a resource is opened.
 - The **Messages** view with validation messages.
 - **Contributors** perspective enables you to view the number of commits sorted by the organizational unit, repository, author, and other criteria.
 - **Artifact Repository** perspective contains a list of jars which can be added as dependencies. The available operations in this perspective are upload/download artifact and open (view) the **pom.xml** file. The view is available for users with the **admin** role only.

- **Administration** perspective contains:
 - The **File Explorer** view with available asset repositories
 - The editor area on the right of the **File Explorer** view, where the respective editor appears when a resource is opened.

The **Administration** perspective allows an administrator to connect a Knowledge Store to a repository with assets and to create a new repository. For more information, see the *Red Hat JBoss BPM Suite Administration and Configuration Guide* .

The view is available for users with the **admin** role only.
- **Deploy** group:
 - **Process Deployments** perspective contains a list of the deployed resources and allows you to build, deploy, and undeploy new units.
 - **Execution Servers** perspective contains a list of the deployed Intelligent Process Server templates and containers associated with the templates.
 - **Jobs** perspective allows you to monitor and trigger asynchronous jobs scheduled for the Executor Service.
- **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.
- **Extensions** group:
 - **Plugin Management** perspective enables you to customize and create new Business Central perspectives and plugins.
 - **Apps** perspective enables you to browse, categorize and open custom perspective plugins.
 - **Data Sets** perspective enables you to define and connect to external data sets.

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)
path	Path to the asset to be edited. Note that asset should already exist.	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.

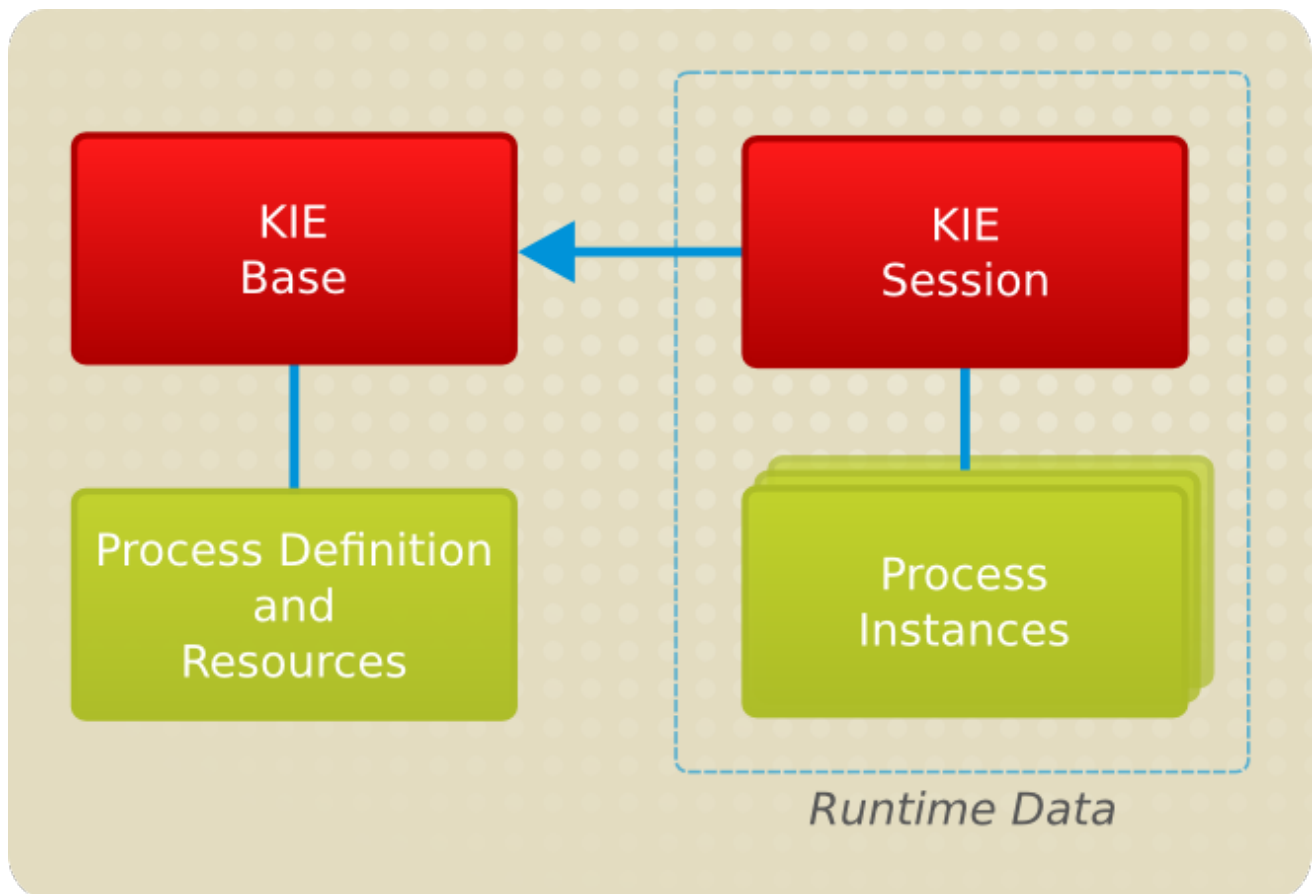
You can define a Kie Base and its Kie Session in the Project Editor of the GUI application (see [Section 3.6, “Defining KIE Bases and Sessions”](#)).

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

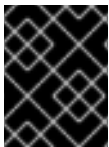
A project is a container for asset packages (business processes, rules, work definitions, decision tables, fact models, data models, and DSLs) that lives in the Knowledge Repository. It is this container that defines the properties of the KIE Base and KIE Session that are applied to its content. In the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

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

Creating an Organizational Unit in Business Central



IMPORTANT

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

Procedure: Using Business Central to Create an Organizational Unit

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

Figure 3.1. Add New Organizational Unit Dialog Window

Add New Organizational Unit
✕

Organizational Unit Information

Name *	<input style="width: 90%;" type="text" value="Organizational Unit name..."/>
Default Group ID * i	<input style="width: 90%;" type="text" value="Default Group Id..."/>
Owner	<input style="width: 90%;" type="text" value="Organizational Unit owner..."/>

+ Ok
Cancel

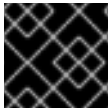
4. Enter the two mandatory parameters (**name** and **default group ID**) and click **Ok**.

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

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

For more information about the `kie-config-cli` tool, see *Red Hat JBoss BPM Suite Administration and Configuration Guide*, chapter *Command Line Configuration*.

Creating an Organizational Unit Using the REST API



IMPORTANT

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

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

Input parameter of the call is an **OrganizationalUnit** instance. The call returns a **CreateOrganizationalUnitRequest** instance.

Example 3.1. Creating an Organizational Unit Using the Curl Utility

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

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

Execute the following command:

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

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

3.2. CREATING A REPOSITORY

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

Creating a Repository in Business Central



IMPORTANT

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

Procedure: Using Business Central to Create a Repository

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

Figure 3.2. *New Repository* Dialog Window

3. Specify the two mandatory parameters:
 - repository name



NOTE

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

- organizational unit: specifies the location of the newly created repository.
4. Click **Finish**.

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

Creating a Repository Using the `kie-config-cli` Tool

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

For more information about the **kie-config-cli** tool, see *Red Hat JBoss BPM Suite Administration and Configuration Guide* .

Creating a Repository Using the REST API



IMPORTANT

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

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

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

Example 3.2. Creating a Repository Using the Curl Utility

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

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

Execute the following command:

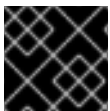
```
curl -X POST 'localhost:8080/business-central/rest/repositories/' -u USERNAME:PASSWORD -H
'Accept: application/json' -H 'Content-Type: application/json' -d
'{"name":"newRepository","description":null,"requestType":"new","gitURL":null,"organizationalUnitName":"helloWorldUnit"}'
```

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

3.3. CLONING A REPOSITORY

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

Cloning a Repository in Business Central



IMPORTANT

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

Procedure: Using Business Central to Clone a Repository

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

Figure 3.3. *Clone Repository* Dialog Window

3. In the **Clone Repository** dialog window, enter the repository details:
 - a. Enter the **Repository Name** to be used as the repository identifier in the Asset repository and select the **Organizational Unit** it should be added to.
 - b. Enter the URL of the Git repository:
 - for a local repository, use **file:///PATH_TO_REPOSITORY/REPOSITORY_NAME;**

**NOTE**

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

- for a remote or preexisting repository, use **<https://github.com/>USERNAME/REPOSITORY_NAME.git** or **git://HOST_NAME/REPOSITORY_NAME.**

**IMPORTANT**

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

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

4. Click **Clone**.

A confirmation prompt with the notification that the repository was created successfully is displayed.

5. Click **Ok**.

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

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

Cloning a Repository Using the REST API



IMPORTANT

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

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

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

Example 3.3. Cloning a Repository Using the Curl Utility

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

```
{
  "name"           : "clonedRepository",
  "description"    : null,
  "requestType"    : "clone",
  "gitURL"         : "git://localhost:9418/newRepository",
  "organizationalUnitName" : "helloWorldUnit"
}
```

Execute the following command:

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

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

3.4. 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.

New Project

New Project
Wizard

Project General Settings

Project Name

Project Description

Group artifact version

Group ID Example: com.myorganization.myprojects

Artifact ID Example: MyProject

Version Example: 1.0.0

< Previous
Next >
Cancel
✓ Finish

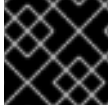
4. Define the **Project General Settings** and **Group artifact version** details of the new project. These parameters are stored in the **pom.xml** Maven configuration file. See the detailed description of the parameters:

- **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.common**).
- **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 3.4. Creating a Project Using the Curl Utility

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

```
{
  "name"      : "MortgageProject",
  "description" : null,
  "groupld"   : "org.mycompany.common",
  "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,"groupld":"org.mycompany.common","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*.

3.5. 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.

3.6. 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.

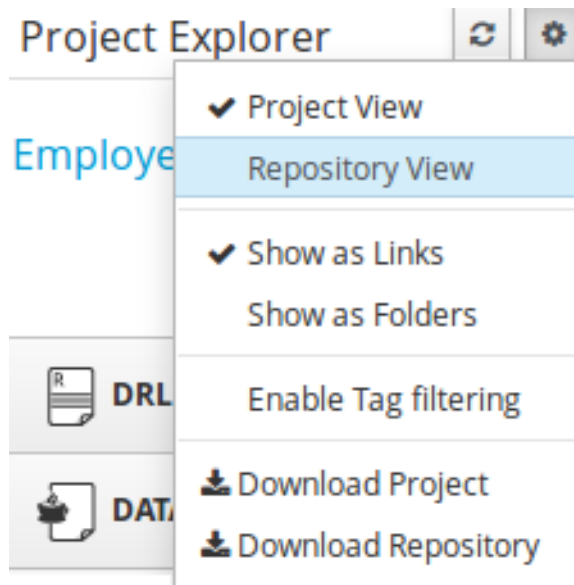
6. Mark it **Default** and select appropriate state. For Red Hat JBoss BPM Suite, use **stateful** sessions.
7. Click **Save** in the top right corner once you are done.

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 3.4. 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.

3.7. 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:

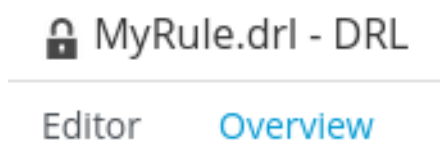
1. In the **Repository** view of the Project Explorer, navigate to the **REPOSITORY/PROJECT/src/main/resources/** directory.
2. Go to **New Item → Package**.
3. In the **New resource** dialog, define the package name and check the location of the package in the repository.

3.8. 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.



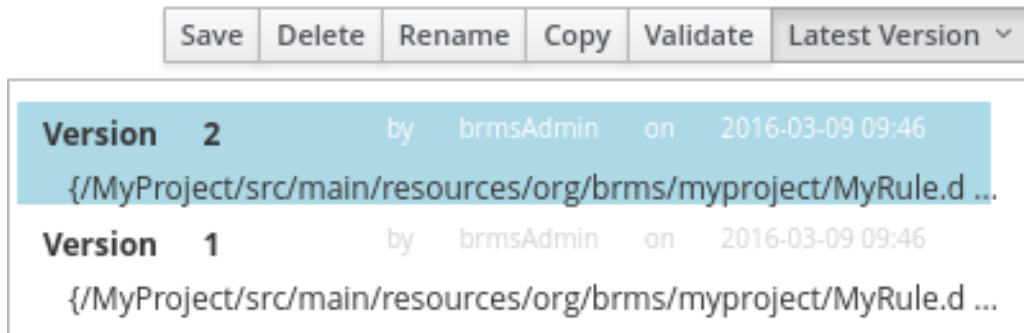
IMPORTANT

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.

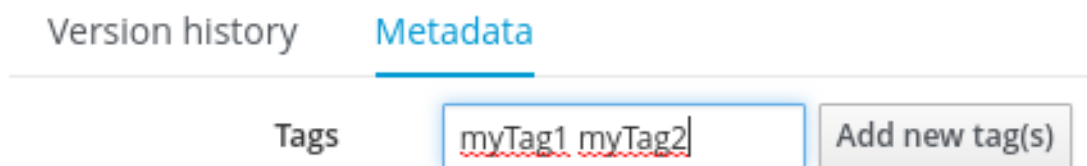
3.9. FILTERING ASSETS BY TAG

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

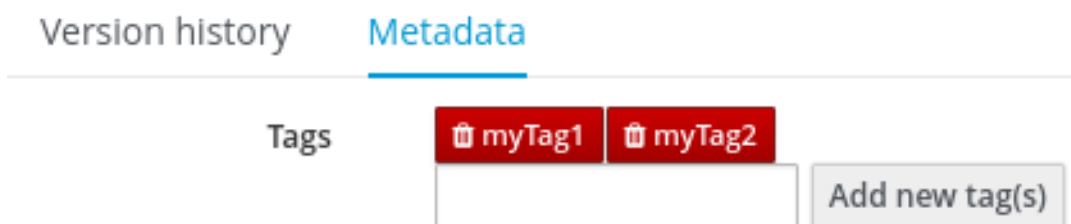
1. Open the **Overview** tab of an asset and click the **Metadata** screen.
2. In the **Tags** field, enter a name of your new tag and click **Add a new tag(s)** button. You can assign multiple tags to an asset at once by separating tag names by space.

Figure 3.5. Creating Tags



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

Figure 3.6. Tags in Metadata View



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


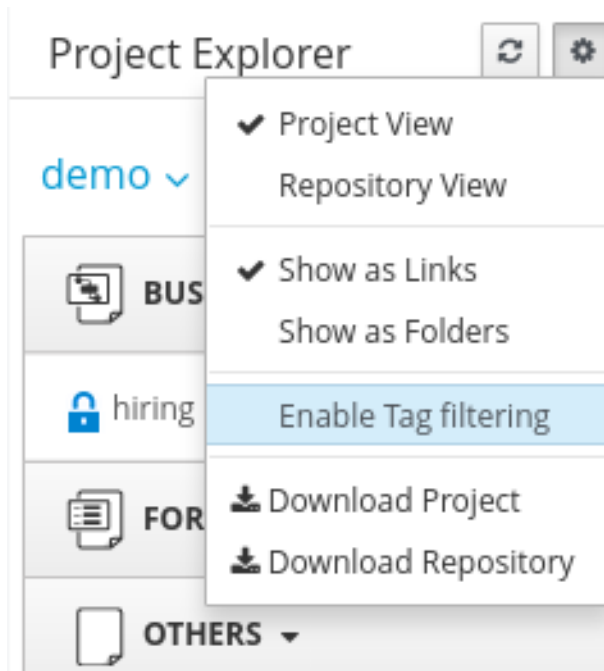
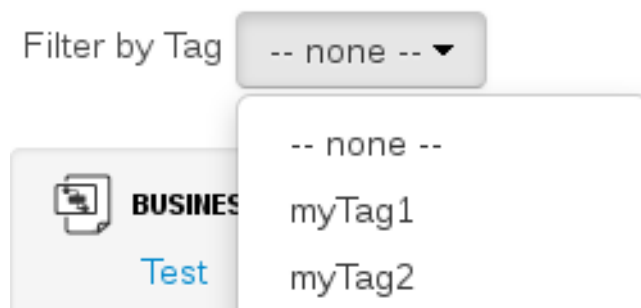
3. Click **Save** button to save your metadata changes.
4. Once you are done assigning tags to your assets, click the  (Customize View) button in the Project Explorer and select the **Enable Tag filtering** option:

Figure 3.7. Enable Tag Filtering



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

Figure 3.8. Filter by Tag



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

3.10. 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 unlock 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.

3.11. 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 3.5. 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>
```

3.11.1. 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.

3.11.2. Importing a Process Definition

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


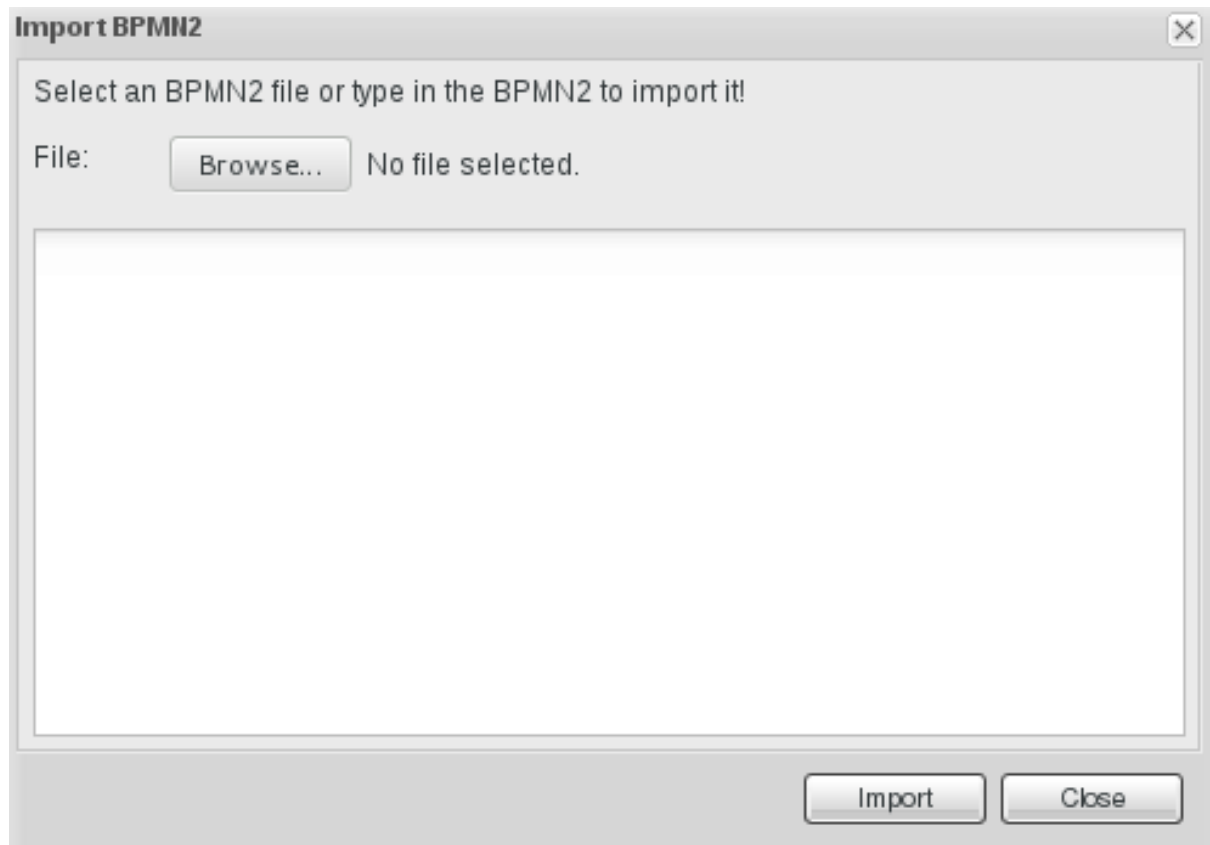
1. In the **Project Explorer**, select a Project and the respective package to which you want to import the Process definition.
2. Create a new Business Process to work in by going to **New Item** → **Business Process**.
3. In the Process Designer toolbar, click the **Import**  icon in the editor toolbar 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 3.9. 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.

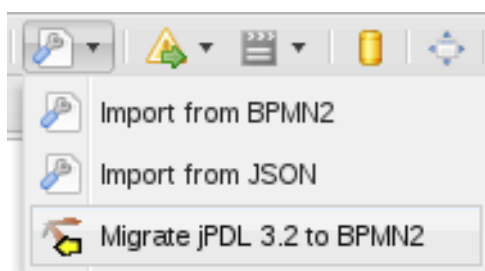
A process can also be imported to the git repository in the filesystem by cloning the repository, adding the process files, and pushing the changes back to git. In addition to alternative import methods, you can copy and paste a process or just open a file in the import dialog.

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.

3.11.3. Importing jPDL 3.2 to BPMN2

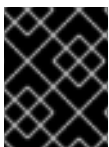
To migrate and import a jPDL definition to BPMN2, in the Process Designer, click on the import button then scroll down and select **Migrate jPDL 3.2 to BPMN2**

Figure 3.10. 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 3.11. 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

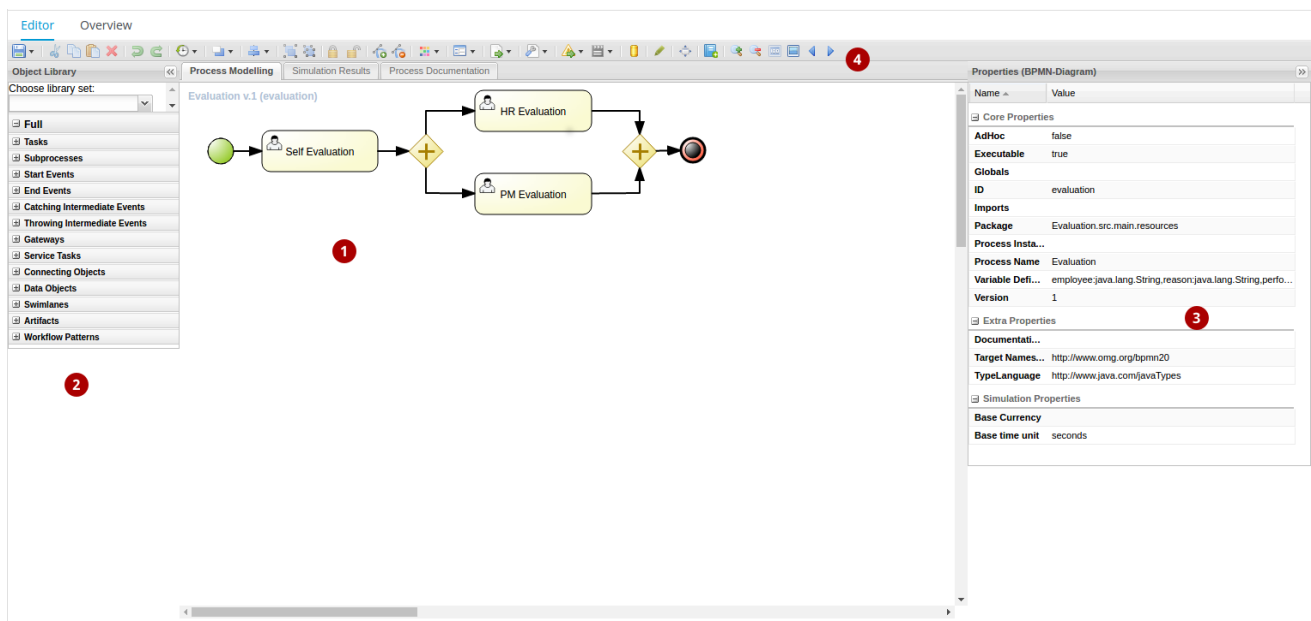
Thick-client version of the Process Designer integrated in the JBoss Developer Studio plug-in

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



1. 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).
2. 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](#).
3. The Properties panel displays the properties of the selected element. If no element is selected, the panel contains process properties.
4. The editor toolbar enables you, for example, to select an operation to be applied to the Elements on the canvas. It also contains tools for validation, simulation, saving, and others.



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


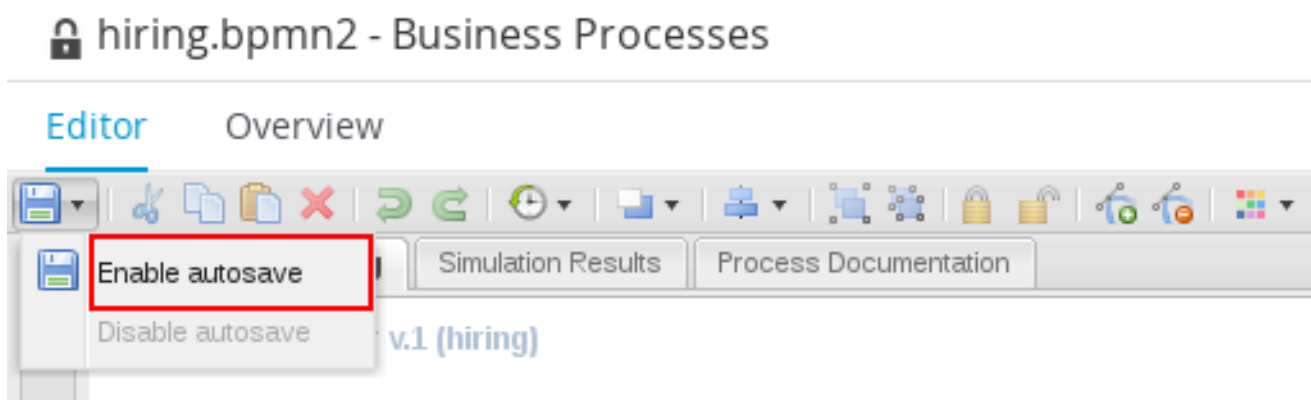
The automatic saving feature periodically commits every change in Process Designer into a Git repository. To set an automatic saving, click the  button in Process Designer and choose **Enable autosave**.

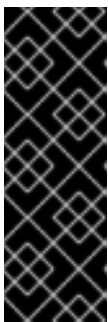
Figure 4.2. Enable Autosave Option in Process Designer



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

Properties (BPMN-Diagram)	
Name ▲	Value
☰ Core Properties	
AdHoc	false
Executable	true
Globals	
ID	myProject.myProc
Imports	
Package	org.jbpm
Process Name	myProc
Variable Defi...	<input type="text"/> ▼
Version	1.0

- Define the process properties on the tab by clicking individual entries. For entries that require other input than 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.
- To save your changes, click **Save** in the top right corner.

4.3. DESIGNING PROCESS

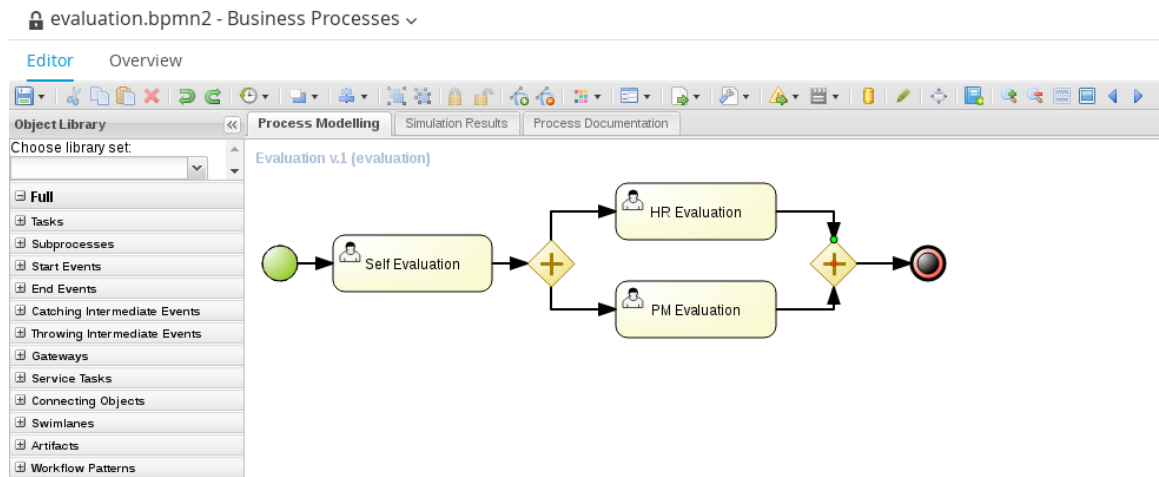
To model a process, do the following:

- 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.

- Add and edit the required shapes to the process diagram on the canvas.
 - 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 menu displays elements that you can connect 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. For multiline names, define 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. Copying Elements

You can copy individual elements and finished business processes. To copy your selection into a different package:

1. On the canvas, click and drag the cursor to select the elements you want to copy.
2. Click  to copy your selection.
3. Switch into the second business process where you want to your add the copied elements.
4. In the second business process, create process variables that are used in the business process you want to copy. Variable **Name** and **Type** parameters must be identical in order to preserve variable mapping.
5. Click  to paste your selection.
6. Click **Save**.

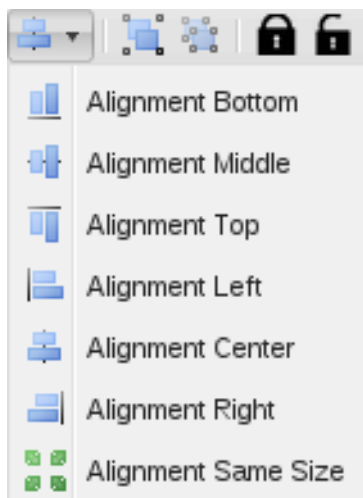
To copy a business process into the same package:

1. Click **Copy**.
2. The **Copy this item** dialogue window appears. Name your copy.

3. Click **Create copy**.

4.3.2. 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.

4.3.3. Changing Element Layering

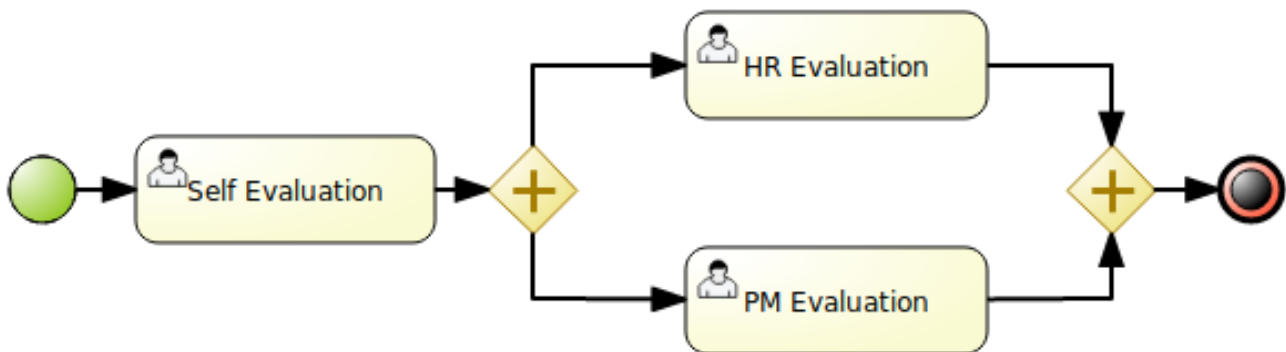
To change the element layering, select the required element or a group of elements and click the  button in the Process Designer toolbar. Choose one of the following options:

-  **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.


4.3.4. Bending Connection Elements

You can bend the connection elements and create angles in your business process. To do so, click and drag the connection element in the desired angle and direction. You can also straighten a bent connection in the same manner, that is clicking on the bent angle and dragging it back to make a straight line.



4.3.5. 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.



4.3.6. 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.

4.3.7. 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** ().


4.3.8. 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.

4.3.9. 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.

4.3.10. Enlarging and shrinking canvas


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

4.3.11. 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

The screenshot shows the Process Designer interface with a process diagram titled "Evaluation v.1 (evaluation)". The diagram consists of a start event (green circle) leading to a "Self Evaluation" task, followed by a join gateway (yellow diamond with a plus sign). This gateway splits into two parallel paths: "HR Evaluation" and "PM Evaluation", both tasks. These paths merge at an end gateway (yellow diamond with a plus sign). A context menu is open over the end gateway, showing options: "Start validating", "Stop validating", and "View all issues".

Below the diagram is a "Messages" section with a table:

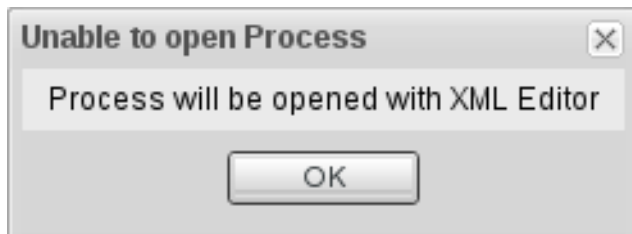
Level	Text	File
ⓘ	Build of project 'evaluation' (requested by bpmAdmin) completed. Build: FAILURE	-
⊗	Process 'Evaluation' [evaluation]: Node '' [6] join has no outgoing connection.	evaluation.bpmn2
⊗	Process 'Evaluation' [evaluation]: Process has no end node.	evaluation.bpmn2

4.3.12. 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**.

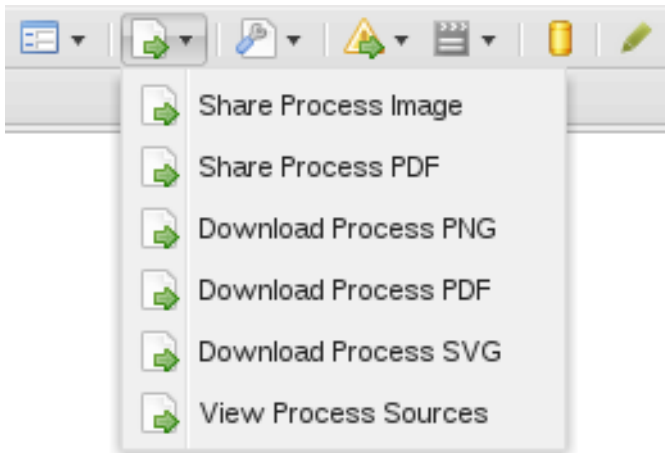
You can now open the valid process and view it as a diagram on the canvas.

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 enables 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).


```

1 <?xml version="1.0" encoding="UTF-8"?>
2 <bpmn2:definitions xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xmlns="http://www.omg.org/bpmn20"
  xmlns:bpmn2="http://www.omg.org/spec/BPMN/20100524/MODEL"
  xmlns:bpmndi="http://www.omg.org/spec/BPMN/20100524/DI"
  xmlns:color="http://www.omg.org/spec/BPMN/non-normative/color"
  xmlns:dc="http://www.omg.org/spec/DD/20100524/DC"
  xmlns:di="http://www.omg.org/spec/DD/20100524/DI"
  xmlns:drools="http://www.jboss.org/drools" id="K_0d0SJSEeaiTZHCI6pl5g"
  xsi:schemaLocation="http://www.omg.org/spec/BPMN/20100524/MODEL BPMN20.xsd
  http://www.jboss.org/drools drools.xsd http://www.bpsim.org/schemas/1.0
  bpsim.xsd" exporter="jBPM Designer" exporterVersion="6.2.0"
  expressionLanguage="http://www.mvel.org/2.0"
  targetNamespace="http://www.omg.org/bpmn20"
  typeLanguage="http://www.java.com/javaTypes">
3 <bpmn2:itemDefinition id="original documentItem"
  structureRef="org.jbpm.document.Document"/>
4 <bpmn2:itemDefinition id="translated documentItem"
  structureRef="org.jbpm.document.Document"/>
5 <bpmn2:itemDefinition id="_statusItem" structureRef="String"/>
6 <bpmn2:itemDefinition id="_commentsItem" structureRef="String"/>
7 <bpmn2:itemDefinition id="_uploader_mailItem" structureRef="String"/>
8 <bpmn2:itemDefinition id="_uploader_nameItem" structureRef="String"/>
9 <bpmn2:itemDefinition id="_initiatorItem" structureRef="String"/>
10 <bpmn2:itemDefinition id="_8B76BE7F-694F-4D8F-9AF7-
  AFB3119E11D1_TaskNameInputXItem" structureRef="String"/>
11 <bpmn2:itemDefinition id="_8B76BE7F-694F-4D8F-9AF7-
  AFB3119E11D1_in_docInputXItem" structureRef="org.jbpm.document.Document"/>
12 <bpmn2:itemDefinition id="_8B76BE7F-694F-4D8F-9AF7-
  AFB3119E11D1_GroupIdInputXItem" structureRef="Object"/>
13 <bpmn2:itemDefinition id="_8B76BE7F-694F-4D8F-9AF7-
  AFB3119E11D1_BusinessAdministratorIdInputXItem" structureRef="Object"/>
14 <bpmn2:itemDefinition id="_8B76BE7F-694F-4D8F-9AF7-
  AFB3119E11D1_out_commentsOutputXItem" structureRef="String"/>

```

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 [Appendix A, 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. BUSINESS PROCESS SAVE POINTS

To ensure the engine will save the state of the process, a save point is created before the following nodes:

- Catch event
- Human tasks
- Every node marked **Is Async**

Asynchronous continuation allows process designers to decide what activities should be executed asynchronously without any additional work. To mark a node as 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**.

The **Is Async** feature is available for all task types (Service, Send, Receive, Business Rule, Script, and User Tasks), subprocesses (embedded and reusable), and multi-instance task and subprocesses. When marked **Is Async**, the node execution is started in a separate thread.

When the engine encounters one of the save point nodes, the transaction is committed into the database before continuing with the execution. This ensures that the state of the process is saved.



NOTE

Asynchronous processing relies on Executor Service component, which must be configured and running. If you are using Red Hat JBoss BPM Suite in the embedded mode, additional steps will be required depending on how you utilize the Red Hat JBoss BPM Suite API.

For fully asynchronous workflow execution, use the Intelligent Process Server configured with JMS Queues.

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:

- Process instantiation
- Task instantiation

The form is then respectively called a *process form* or a *task form*. Forms acquire data from a human user for both the process instance execution, or the task instance execution:

- A process form can take as its input and output process variables.
- A task form can take as its input Data Input Assignment variables with assignment defined, and as its output Data Output Assignments with assignment defined.

For example:


- With a process form, a user can provide the input parameters needed for process instantiation.
- With a task form, you can use a Human Task to provide input for further process execution.

The input is then mapped to the task using the data input assignment, which you can then use inside of a task. When the task is completed, the data is mapped as a data output assignment to provide the data to the parent process instance. For further information, 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 Task 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.

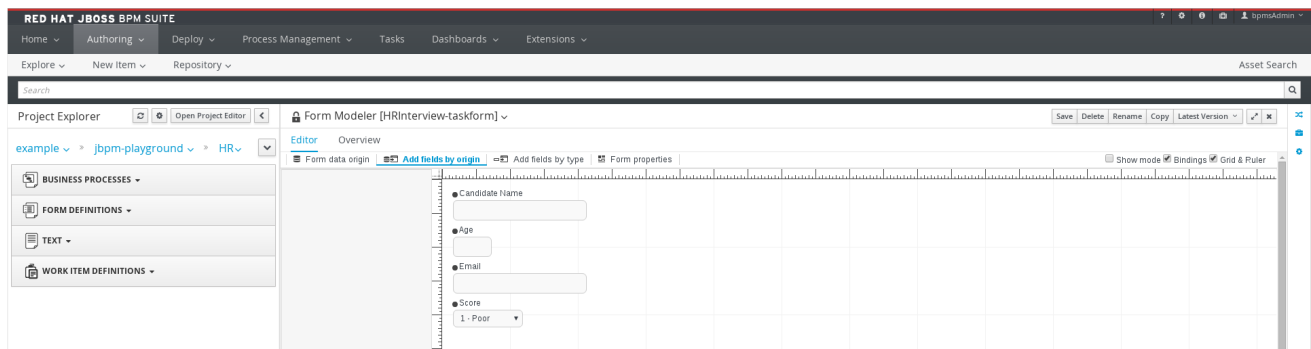
Form Modeler includes the following key features:

- 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 enables 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 Basic type**, **From Data Model**, and **From Java Class**

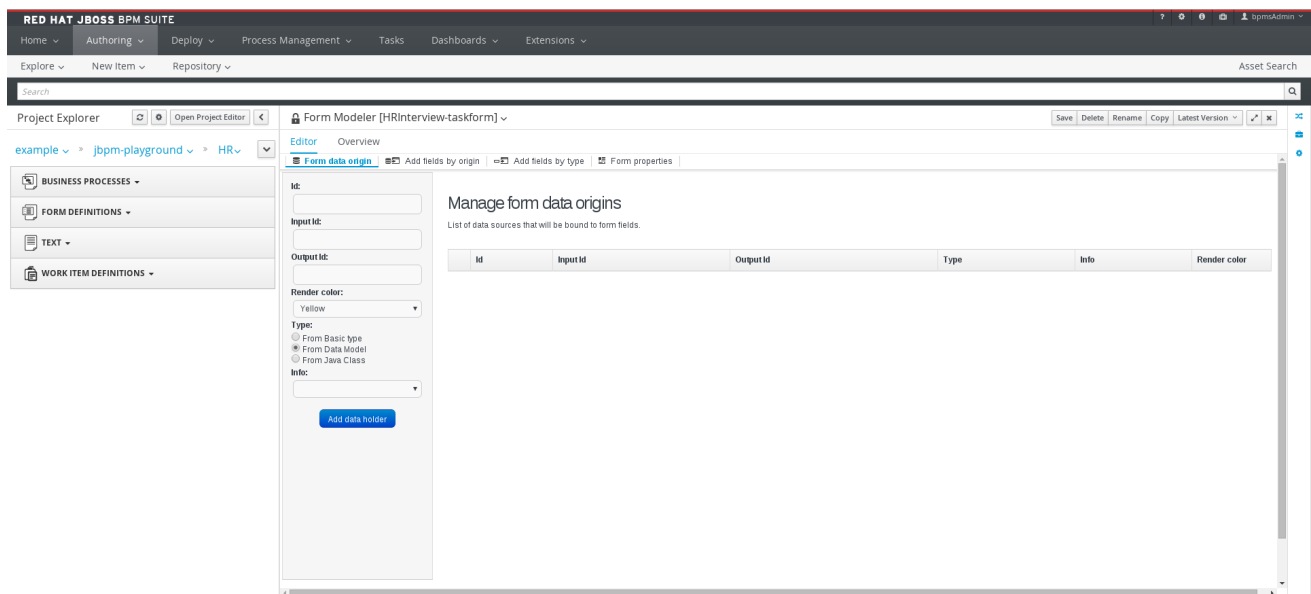
Use the **Add fields by origin** tab visible in the following figure to select fields based on their source.

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:

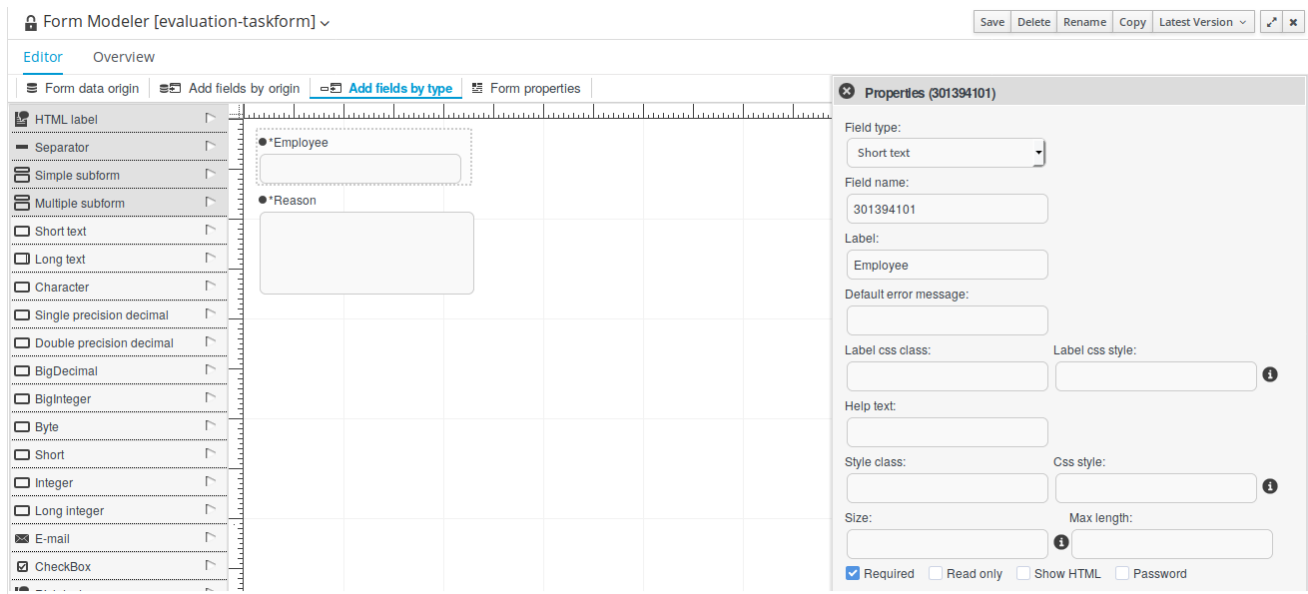
1. In Business Central, go to **Authoring** → **Project Authoring**.
2. On the perspective menu, select **New Item** → **Form**.
3. In the **Create New Form** dialog window, enter the name of your form in **Resource Name**, select the package, and click **OK**.

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.

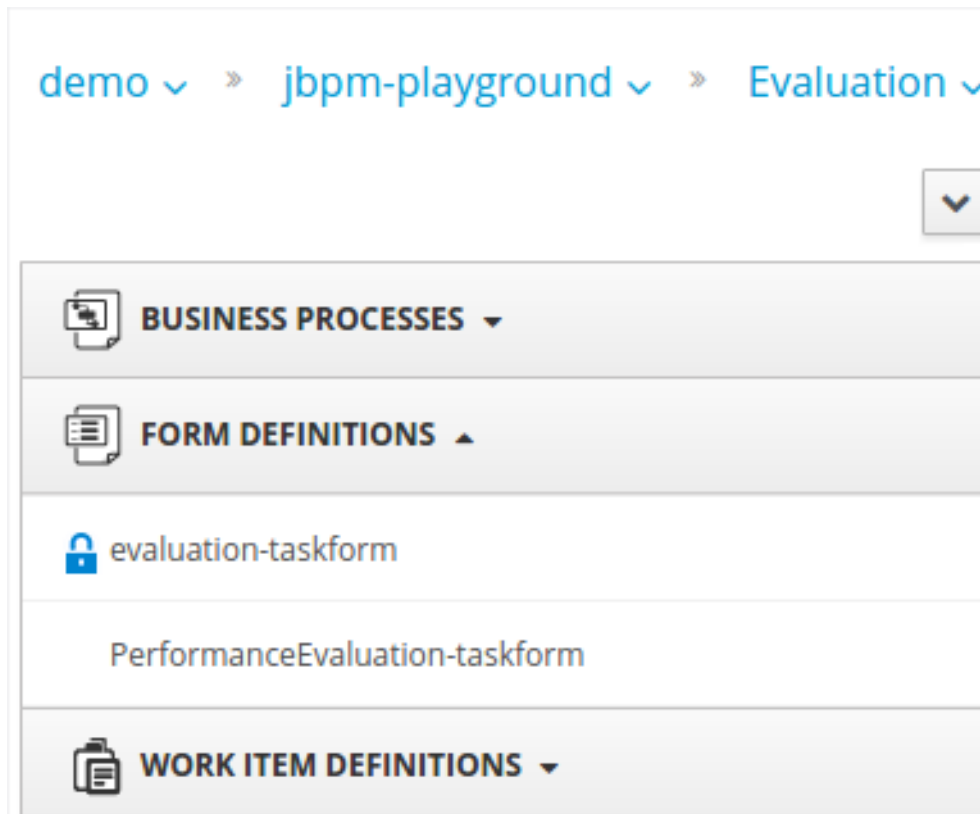
Figure 4.8. New form



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:

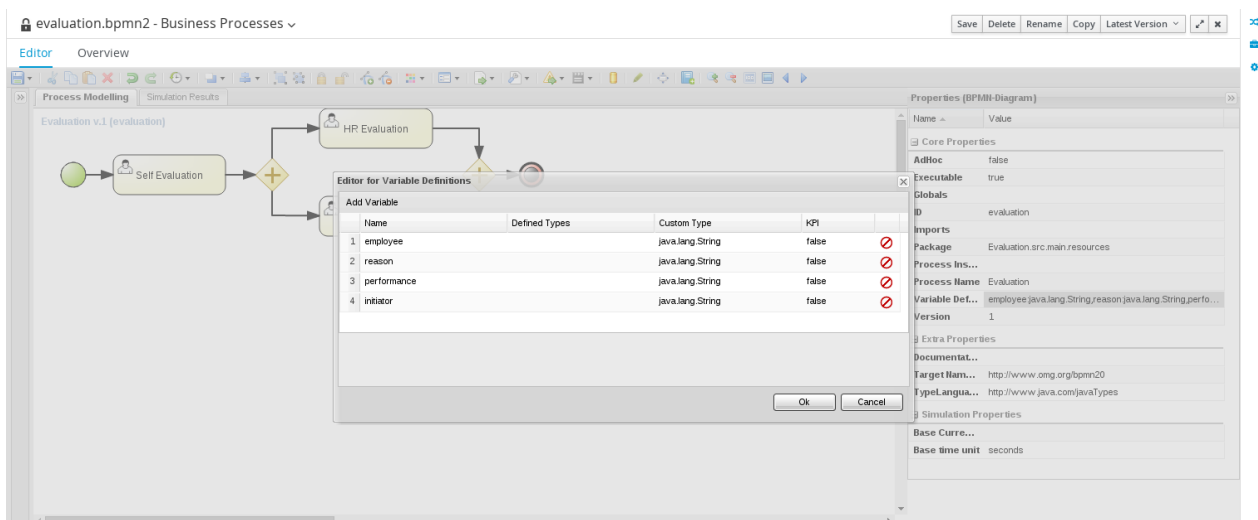
1. In Form Modeler, select the **Add fields by type** tab and click the arrow  button to the right of a field type. The field type is added to the canvas.
2. On the canvas, place the cursor on the field and click the edit  icon.
3. In the **Properties** dialog window that opens on the right, set the form field properties and click **Apply** at the bottom of the dialog window 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 store the form input. Variables can be of a simple type, like **String**, or a complex type. 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 **Editor for Variable Property** window of the **variables definition** property of the business process.
3. Determine which variables you want to set as input parameters for the task, which will receive response from the form. After you create the variables, map the variables to inputs by setting **Data Input Assignments** and **Data Output Assignments** for a Human Task. To do so, use the **Data I/O** form of the **Assignments** property.

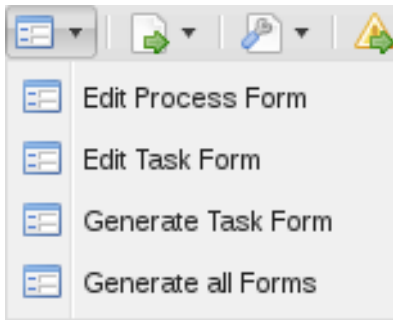
Example 4.1. Defining a Variable using Data Modeler



4.8.5. Generating Forms from Task Definitions

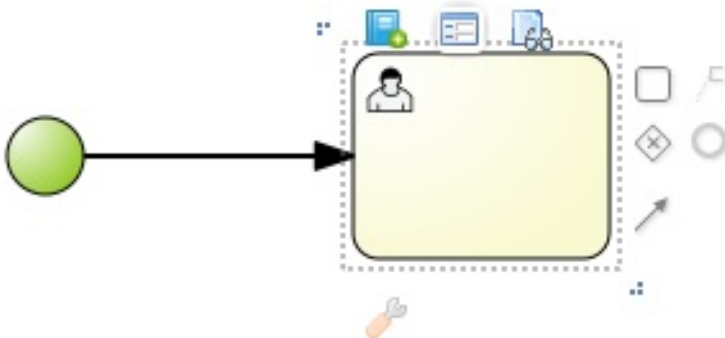
In the Process Designer module, you can generate forms automatically from task and variable definitions, and easily open concrete forms from Form Modeler by using the following menu option:

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 **TASK_NAME-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 **PROCESS_ID-task**, the application will use it as the initial form when starting the process.

4.8.6. Editing Forms

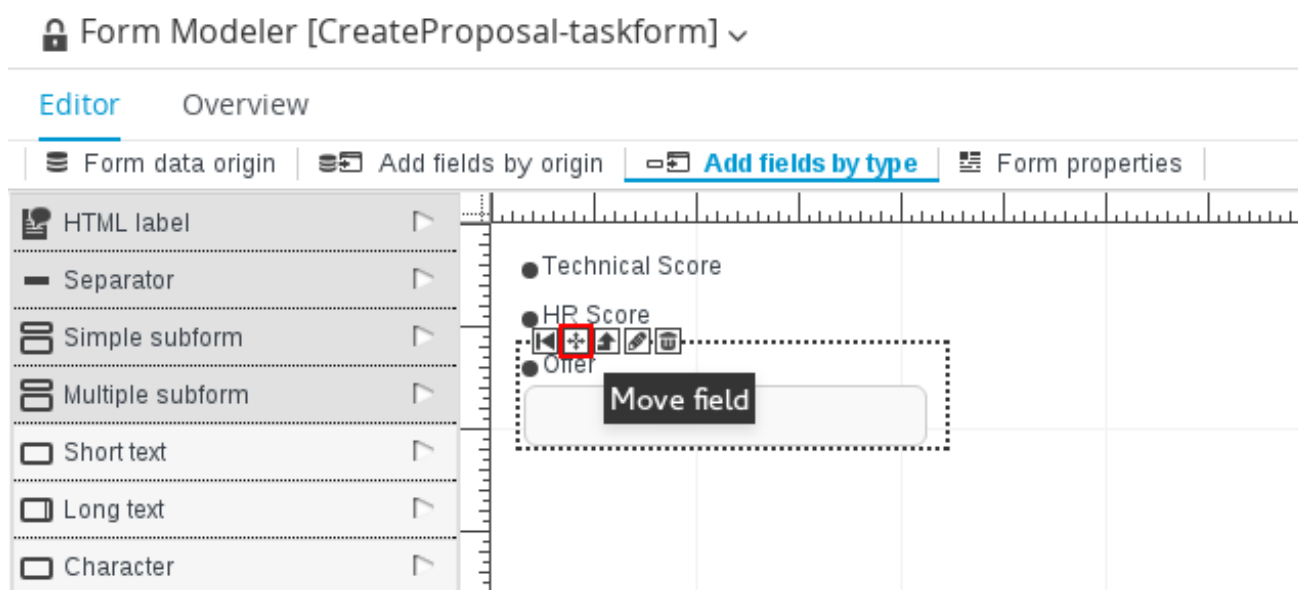
After you generate a form, you can start editing it. If the form has been generated automatically, the **Form data origin** tab contains the process variables as the origin of the data, which enables you to bind form fields with them and create data bindings. Data bindings determine the way task input 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

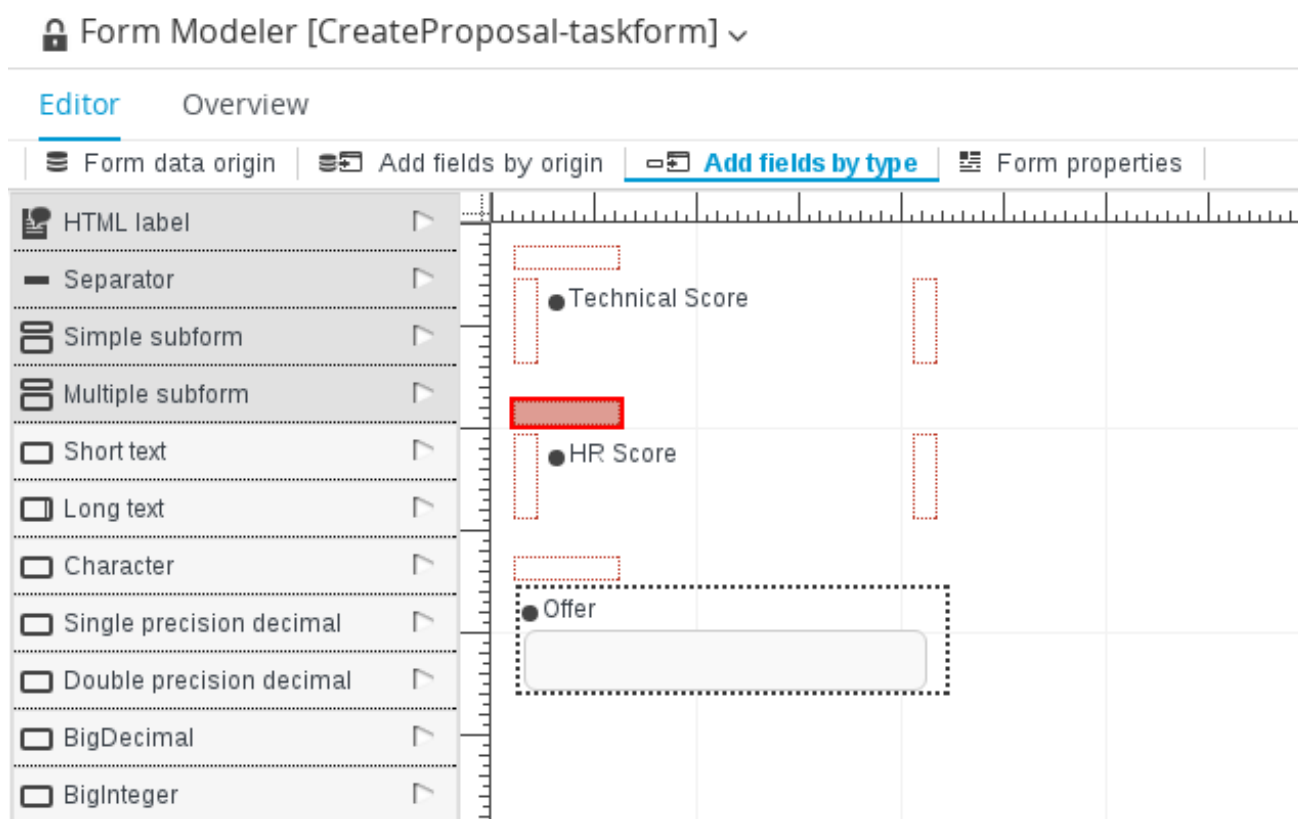
You can place fields in different areas of the form. To move a field, access the field's contextual menu and select the **Move field** option shown on the following screenshot. This option displays the different regions of the form where you can place the field.

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 enables you to add fields to the form based on defined data origins.

Figure 4.14. Adding Fields by Origin

Form Modeler [CreateOrder-taskform] ▾

Editor Overview

Form data origin **Add fields by origin** Add fields by type Form properties

po

requiresCFOAp...

Please, enter all the required information. The instructions to perform this task can be found [here](#)

● Purchase Order Header

*Creation date 04-04-16

*Customer

*Project

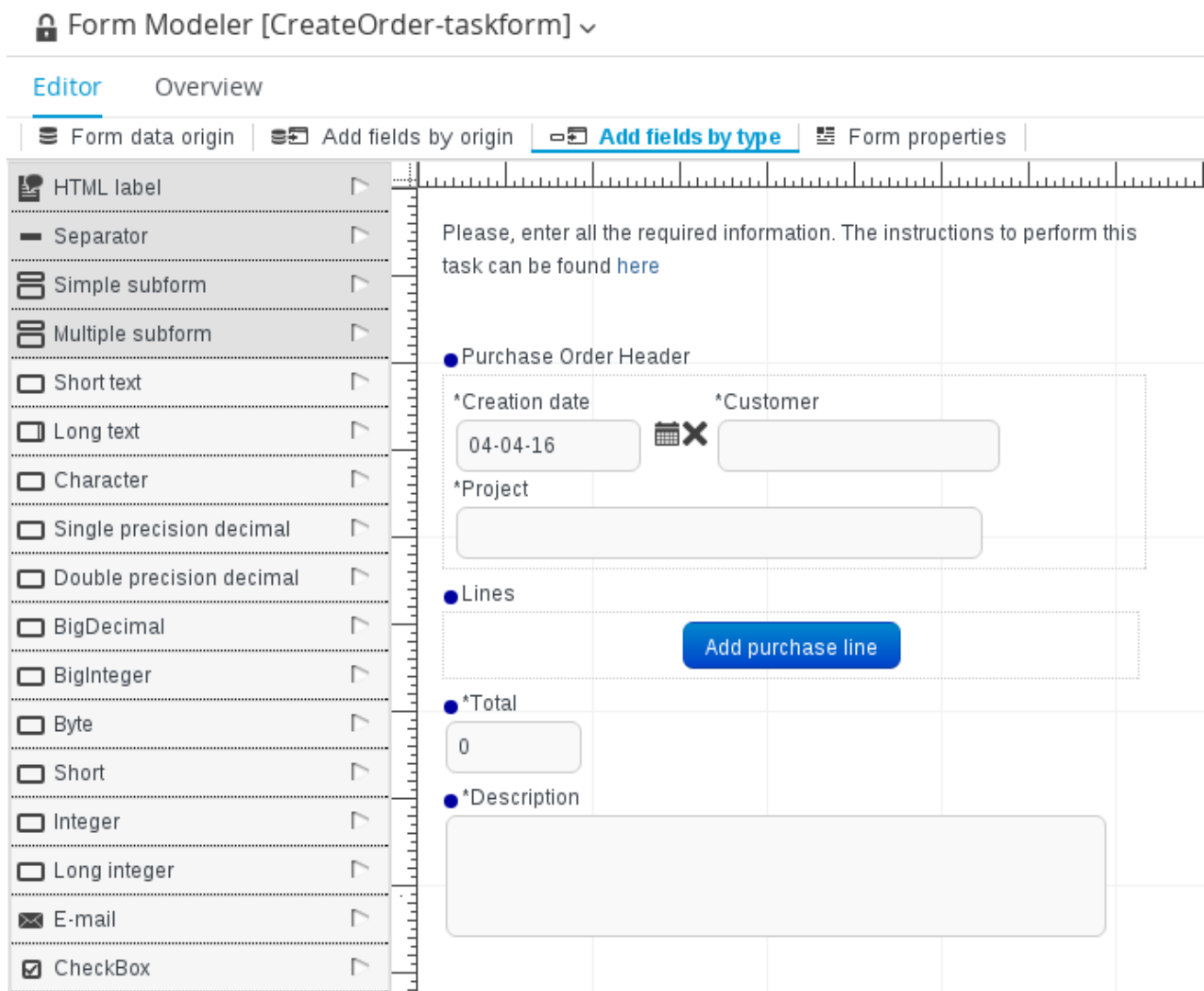
● Lines

● *Total

● *Description

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 enables 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



NOTE

The **Document** form field requires additional setup to be accessed from the relevant forms and processes. For information about enabling document attachments, see [Section 4.8.11, "Enabling Document Attachments in a Form or Process"](#).

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** is the text that is displayed as a field label.
- **Error Message** is a message displayed when there is a problem with a field, for example in validation.
- **Label CSS Class** enables you to enter a class css to apply in label visualization.
- **Label CSS Style** enables you to enter the style to be applied to the label.
- **Help Text** is the text displayed as an alternative attribute to help the user in data introduction.
- **Style Class** enables you to enter a class CSS to be applied in field visualization.
- **CSS Style** enables you to directly enter the style to be applied to the label.
- **Read Only** allows reading only, provides no write access to such field.
- **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**.

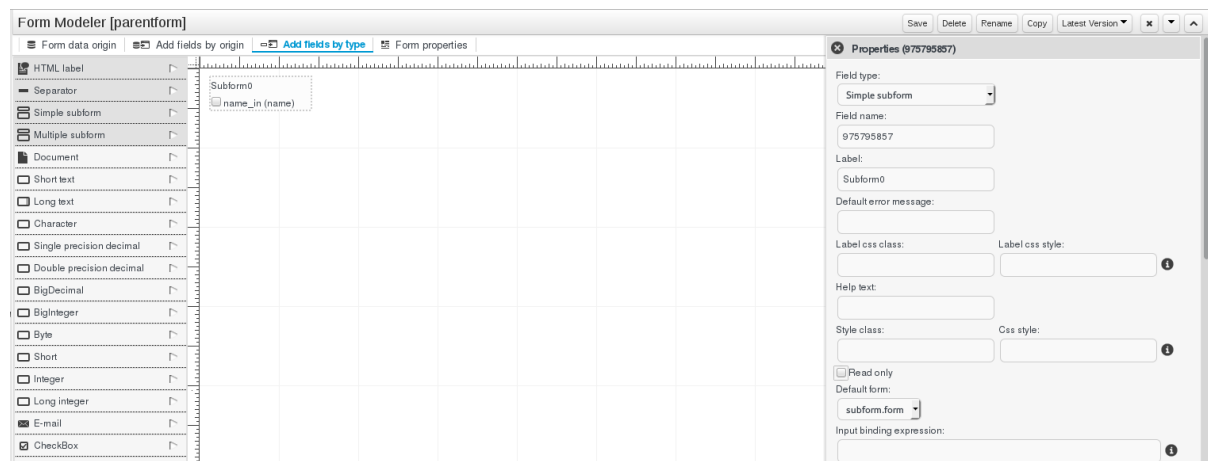
Figure 4.16. Create Subform

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

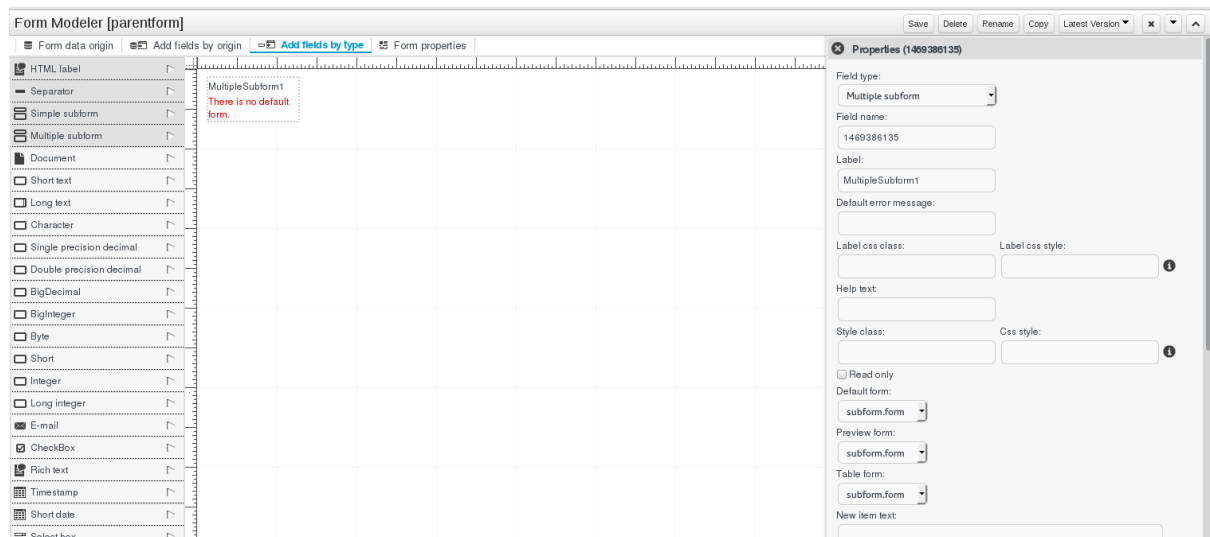


10. Click **Save** to save the parent form.
This inserts your subform containing a single Java object inside the parent form.

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. Enabling Document Attachments in a Form or Process

Red Hat JBoss BPM Suite supports document attachments in forms using the **Document** form field. With the **Document** form field, you can upload documents that are required as part of a form or process. For information about adding fields to forms, see [Section 4.8.8, "Adding New Fields to a Form"](#).

To enable document attachments in forms and processes, follow these steps:

- Set the document marshalling strategy.
- Create a document variable in the process.
- Map the task inputs and outputs to the variable.

Set the Document Marshalling Strategy

The document marshalling strategy for your project determines where documents are stored for use with forms and processes. The default document marshalling strategy in Red Hat JBoss BPM Suite is **org.jbpm.document.marshalling.DocumentMarshallingStrategy**. This strategy uses a **DocumentStorageServiceImpl** class that stores documents locally in your **PROJECT_HOME/docs** folder. You can set this document marshalling strategy or a custom document marshalling strategy for your project in Business Central or in the **kie-wb-deployment-descriptor.xml** file directly.

1. In Business Central, click **Authoring** → **Project Authoring** and navigate to your project.
2. Click **Open Project Editor** and then click **Project Settings: Project General Settings** → **Deployment descriptor**.
3. Under **Marshalling strategies**, click **Add**.
4. In the **Identifier** value field, click **Enter Value** and enter **org.jbpm.document.marshalling.DocumentMarshallingStrategy** to use the default document marshalling strategy or enter the identifier of a custom document marshalling strategy.

For more information about custom document marshalling strategies, see [Section 4.8.11.1, "Using a Custom Document Marshalling Strategy for a Content Management System \(CMS\)"](#).



5. Set **Resolver type** to **reflection**.
6. Click **Save** and **Validate** to ensure correctness of your deployment descriptor file. Alternatively, you can navigate to `~/META_INF/kie-wb-deployment-descriptor.xml` in your project and edit the deployment descriptor file directly with the required `<marshalling-strategies>` elements.

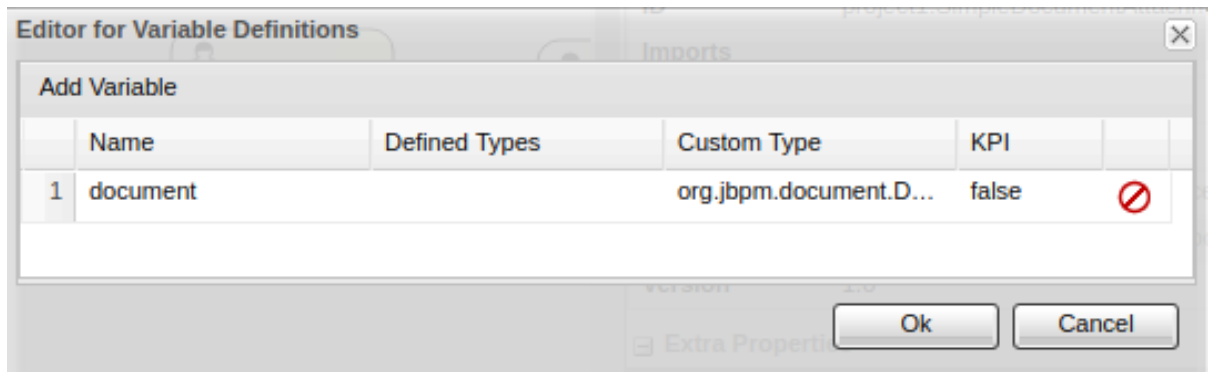
Example `kie-wb-deployment-descriptor.xml` file with default document marshalling strategy:

```
<deployment-descriptor
  xsi:schemaLocation="http://www.jboss.org/jbpm deployment-descriptor.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <persistence-unit>org.jbpm.domain</persistence-unit>
  <audit-persistence-unit>org.jbpm.domain</audit-persistence-unit>
  <audit-mode>JPA</audit-mode>
  <persistence-mode>JPA</persistence-mode>
  <runtime-strategy>SINGLETON</runtime-strategy>
  <marshalling-strategies>
  <marshalling-strategy>
  <resolver>reflection</resolver>
  <identifier>
  org.jbpm.document.marshalling.DocumentMarshallingStrategy
  </identifier>
  </marshalling-strategy>
  </marshalling-strategies>
```

Create a Document Variable in the Process

After you set the document marshalling strategy, create a document variable in the related process. This variable is required for the document to be visible in the **Documents** tab of the **Process Management** → **Process Instances** view in Business Central.

1. In Business Central, navigate to your business process and open it in the Business Process Designer.
2. Click on the canvas and click  on the right side of the window to open the **Properties** tab.
3. Next to **Variable Definition**, click the empty space and click . The **Editor for Variable Definitions** dialog opens.
4. Click **Add Variable** and enter the following values:
 - Name: **document**
 - Custom Type: **org.jbpm.document.Document**
5. Click **Ok**.



Map Task Inputs and Outputs to the Document Variable

If you want to view or modify the attachments inside of the task forms, create assignments inside of the task inputs and outputs.

1. In Business Central, navigate to your business process and open it in the Business Process Designer.
2. Click on a User Task and click on the right side of the window to open the **Properties** tab.
3. Next to **Assignments**, click the empty space and click . The **Data I/O** dialog window opens.
4. Next to **Data Inputs and Assignments**, click **Add** and enter the following values:
 - Name: **taskdoc_in**
 - Data Type: **Object**
 - Source: **document**
5. Next to **Data Outputs and Assignments**, click **Add** and enter the following values:
 - Name: **taskdoc_out**
 - Data Type: **Object**
 - Target: **document**

Note that the **Source** and **Target** fields contain the name of the process variable you created earlier.

6. Click **Save**.
7. In the Process Designer, click and select **Generate all Forms**.
8. Click **Save** to save the process.

Now, when you build and deploy your project, you can see any configured **Document** attachments in the **Documents** tab of the **Process Management** → **Process Instances** view.

4 - SimpleDocumentAttachmentProcess

Options ▾



Instance Details

Process Variables

Documents

Logs

Name	Last Modification	Size	Actions
15036183_179...	Sat Dec 03 09:...	43.579 KB	Download

4.8.11.1. Using a Custom Document Marshalling Strategy for a Content Management System (CMS)

The document marshalling strategy for your project determines where documents are stored for use with forms and processes. The default document marshalling strategy in Red Hat JBoss BPM Suite is **org.jbpm.document.marshalling.DocumentMarshallingStrategy**. This strategy uses a **DocumentStorageServiceImpl** class that stores documents locally in your **PROJECT_HOME/docs** folder. If you want to store form and process documents in a custom location, such as in a centralized content management system (CMS), add a custom document marshalling strategy to your project. You can set this document marshalling strategy in Business Central or in the **kie-deployment-descriptor.xml** file directly.

1. Create a custom marshalling strategy **.java** file that includes an implementation of the **org.kie.api.marshalling.ObjectMarshallingStrategy** interface. This interface enables you to implement the variable persistence required for your custom document marshalling strategy. The following methods in this interface help you create your strategy:
 - **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 backward compatibility.
 - **Object read(ObjectInputStream os)**: Same as **unmarshal**, provided for backward compatibility.

Example **ObjectMarshallingStrategy** implementation for storing and retrieving data from a Content Management Interoperability Services (CMIS) system:

```
package org.jbpm.integration.cmis.impl;

import java.io.ByteArrayInputStream;
import java.io.ByteArrayOutputStream;
import java.io.IOException;
import java.io.ObjectInputStream;
import java.io.ObjectOutputStream;
```

```
import java.util.HashMap;

import org.apache.chemistry.opencmis.client.api.Folder;
import org.apache.chemistry.opencmis.client.api.Session;
import org.apache.chemistry.opencmis.commons.data.ContentStream;
import org.apache.commons.io.IOUtils;
import org.drools.core.common.DroolsObjectInputStream;
import org.jbpm.document.Document;
import org.jbpm.integration.cmis.UpdateMode;

import org.kie.api.marshalling.ObjectMarshallingStrategy;

public class OpenCMISPlaceholderResolverStrategy extends OpenCMISSupport implements
ObjectMarshallingStrategy {

    private String user;
    private String password;
    private String url;
    private String repository;
    private String contentUrl;
    private UpdateMode mode = UpdateMode.OVERRIDE;

    public OpenCMISPlaceholderResolverStrategy(String user, String password, String url,
String repository) {
        this.user = user;
        this.password = password;
        this.url = url;
        this.repository = repository;
    }

    public OpenCMISPlaceholderResolverStrategy(String user, String password, String url,
String repository, UpdateMode mode) {
        this.user = user;
        this.password = password;
        this.url = url;
        this.repository = repository;
        this.mode = mode;
    }

    public OpenCMISPlaceholderResolverStrategy(String user, String password, String url,
String repository, String contentUrl) {
        this.user = user;
        this.password = password;
        this.url = url;
        this.repository = repository;
        this.contentUrl = contentUrl;
    }

    public OpenCMISPlaceholderResolverStrategy(String user, String password, String url,
String repository, String contentUrl, UpdateMode mode) {
        this.user = user;
        this.password = password;
        this.url = url;
        this.repository = repository;
        this.contentUrl = contentUrl;
        this.mode = mode;
    }
}
```

```

    }

    public boolean accept(Object object) {
        if (object instanceof Document) {
            return true;
        }
        return false;
    }

    public byte[] marshal(Context context, ObjectOutputStream os, Object object) throws
    IOException {
        Document document = (Document) object;
        Session session = getRepositorySession(user, password, url, repository);
        try {
            if (document.getContent() != null) {
                String type = getType(document);
                if (document.getIdentifier() == null || document.getIdentifier().isEmpty()) {
                    String location = getLocation(document);

                    Folder parent = findFolderForPath(session, location);
                    if (parent == null) {
                        parent = createFolder(session, null, location);
                    }
                    org.apache.chemistry.opencmis.client.api.Document doc = createDocument(session,
                    parent, document.getName(), type, document.getContent());
                    document.setIdentifier(doc.getId());
                    document.addAttribute("updated", "true");
                } else {
                    if (document.getContent() != null && "true".equals(document.getAttribute("updated"))) {
                        org.apache.chemistry.opencmis.client.api.Document doc = updateDocument(session,
                        document.getIdentifier(), type, document.getContent(), mode);

                        document.setIdentifier(doc.getId());
                        document.addAttribute("updated", "false");
                    }
                }
            }
        } finally {
            session.clear();
        }
    }

    public Object unmarshal(Context context, ObjectInputStream ois, byte[] object, ClassLoader
    classloader) throws IOException, ClassNotFoundException {
        DroolsObjectInputStream is = new DroolsObjectInputStream( new ByteArrayInputStream(
        object ), classloader );
        String objectId = is.readUTF();
        String canonicalName = is.readUTF();
        Session session = getRepositorySession(user, password, url, repository);
        try {

```

```

    org.apache.chemistry.opencmis.client.api.Document doc =
(org.apache.chemistry.opencmis.client.api.Document) findObjectForId(session, objectId);
    Document document = (Document) Class.forName(canonicalName).newInstance();
    document.setAttributes(new HashMap<String, String>());

    document.setIdentifier(objectId);
    document.setName(doc.getName());
    document.setLastModified(doc.getLastModificationDate().getTime());
    document.setSize(doc.getContentStreamLength());
    document.addAttribute("location", getFolderName(doc.getParents()) +
getPathAsString(doc.getPaths()));
    if (doc.getContentStream() != null && contentUrl == null) {
        ContentStream stream = doc.getContentStream();
        document.setContent(IOUtils.toByteArray(stream.getStream()));
        document.addAttribute("updated", "false");
        document.addAttribute("type", stream.getMimeType());
    } else {
        document.setLink(contentUrl + document.getIdentifier());
    }
    return document;
} catch (Exception e) {
    throw new RuntimeException("Cannot read document from CMIS", e);
} finally {
    is.close();
    session.clear();
}
}

public Context createContext() {
    return null;
}

// For backward compatibility with previous serialization mechanism
public void write(ObjectOutputStream os, Object object) throws IOException {
    Document document = (Document) object;
    Session session = getRepositorySession(user, password, url, repository);
    try {
        if (document.getContent() != null) {
            String type = document.getAttribute("type");
            if (document.getIdentifier() == null) {
                String location = document.getAttribute("location");

                Folder parent = findFolderForPath(session, location);
                if (parent == null) {
                    parent = createFolder(session, null, location);
                }
                org.apache.chemistry.opencmis.client.api.Document doc = createDocument(session,
parent, document.getName(), type, document.getContent());
                document.setIdentifier(doc.getId());
                document.addAttribute("updated", "false");
            } else {
                if (document.getContent() != null && "true".equals(document.getAttribute("updated"))) {
                    org.apache.chemistry.opencmis.client.api.Document doc = updateDocument(session,
document.getIdentifier(), type, document.getContent(), mode);

                    document.setIdentifier(doc.getId());
                }
            }
        }
    }
}

```

```

        document.addAttribute("updated", "false");
    }
}
}
ByteArrayOutputStream buff = new ByteArrayOutputStream();
ObjectOutputStream oos = new ObjectOutputStream( buff );
oos.writeUTF(document.getIdentifier());
oos.writeUTF(object.getClass().getCanonicalName());
oos.close();
} finally {
    session.clear();
}
}

public Object read(ObjectInputStream os) throws IOException, ClassNotFoundException {
    String objectId = os.readUTF();
    String canonicalName = os.readUTF();
    Session session = getRepositorySession(user, password, url, repository);
    try {
        org.apache.chemistry.opencmis.client.api.Document doc =
(org.apache.chemistry.opencmis.client.api.Document) findObjectForId(session, objectId);
        Document document = (Document) Class.forName(canonicalName).newInstance();

        document.setIdentifier(objectId);
        document.setName(doc.getName());
        document.addAttribute("location", getFolderName(doc.getParents()) +
getPathAsString(doc.getPaths()));
        if (doc.getContentStream() != null) {
            ContentStream stream = doc.getContentStream();
            document.setContent(IOUtils.toByteArray(stream.getStream()));
            document.addAttribute("updated", "false");
            document.addAttribute("type", stream.getMimeType());
        }
        return document;
    } catch(Exception e) {
        throw new RuntimeException("Cannot read document from CMIS", e);
    } finally {
        session.clear();
    }
}
}
}

```

2. In Business Central, click **Authoring** → **Project Authoring** and navigate to your project.
3. Click **Open Project Editor** and then click **Project Settings: Project General Settings** → **Deployment descriptor**.
4. Under **Marshalling strategies**, click **Add**.
5. In the **Identifier** value field, click **Enter Value** and enter the identifier of the custom document marshalling strategy that you created (for example, **org.jbpm.integration.cmis.impl.OpenCMISPlaceholderResolverStrategy**).
6. Set **Resolver type** to **reflection**.

- Click **Save** and **Validate** to ensure correctness of your deployment descriptor file. Alternatively, you can navigate to `~/META_INF/kie-wb-deployment-descriptor.xml` in your project and edit the deployment descriptor file directly with the required `<marshalling-strategies>` elements.

Example `kie-deployment-descriptor.xml` file with custom document marshalling strategy:

```
<deployment-descriptor
  xsi:schemaLocation="http://www.jboss.org/jbpm deployment-descriptor.xsd"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <persistence-unit>org.jbpm.domain</persistence-unit>
  <audit-persistence-unit>org.jbpm.domain</audit-persistence-unit>
  <audit-mode>JPA</audit-mode>
  <persistence-mode>JPA</persistence-mode>
  <runtime-strategy>SINGLETON</runtime-strategy>
  <marshalling-strategies>
  <marshalling-strategy>
  <resolver>reflection</resolver>
  <identifier>
  org.jbpm.integration.cmis.impl.OpenCMISPlaceholderResolverStrategy
  </identifier>
  </marshalling-strategy>
  </marshalling-strategies>
```

- To enable documents stored in a custom location to be attached to forms and processes, create a document variable in the relevant processes and map task inputs and outputs to that document variable in Business Central. For instructions, see [Section 4.8.11, "Enabling Document Attachments in a Form or Process"](#).

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:

- 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 jbpRestAPI = 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:

```
jbpRestAPI.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 element 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:

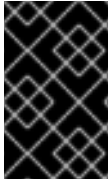
- **Session context**: *Global variables* are visible to all process instances and assets in the given session and are intended to be used primarily by business rules and by constraints. These are created dynamically by the rules or constraints.
- **Process context**: *Process variables* are defined 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 enables 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:


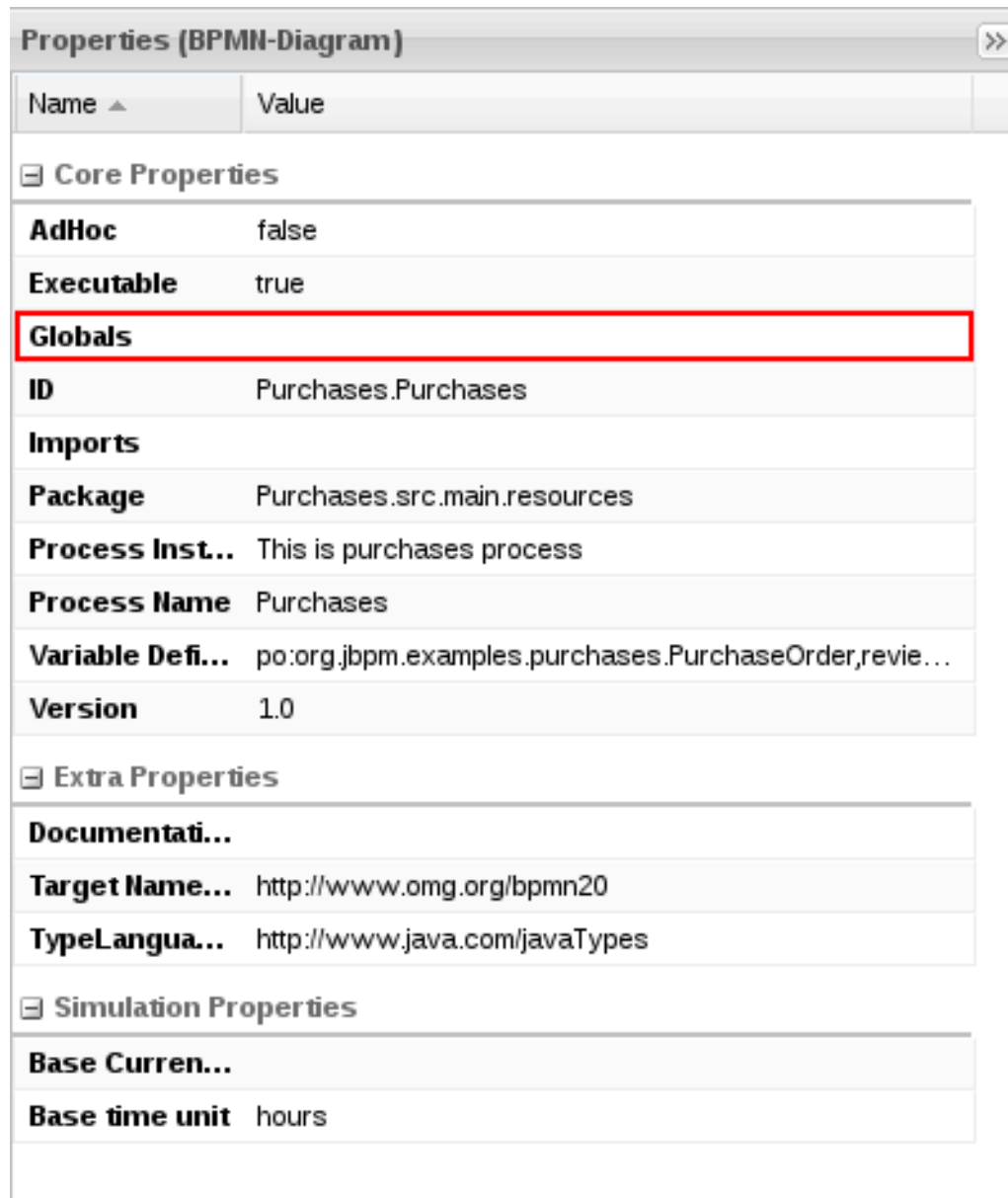
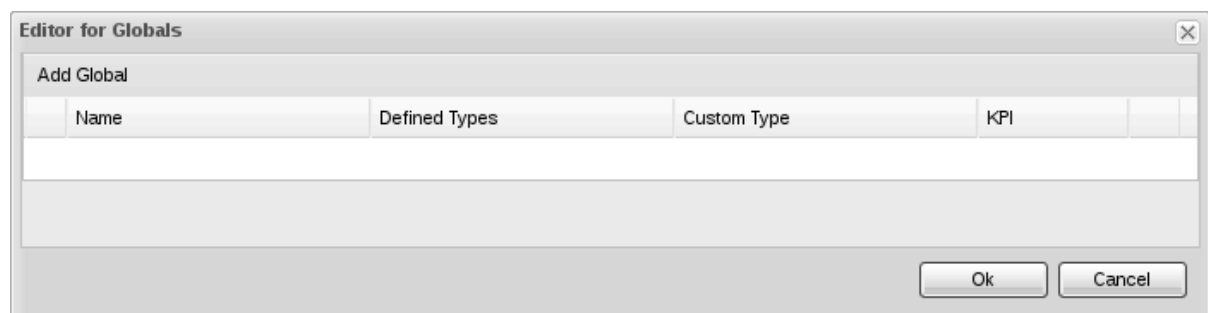
1. In Business Central, go to **Authoring** → **Project Authoring**.
2. Open the respective process in **Process Designer**.
3. Click  in the right hand corner of the **Process Designer** and in the **Properties (BPMN-Diagram)** panel that opens, locate the **Globals** property.

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.

4.9.1.2. 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 a 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.

Procedure: Defining Process Variables

1. In Business Central, click **Authoring** → **Project Authoring**.
2. Open the respective process in **Process Designer**.
3. Click on an empty space in the canvas and click .
4. Click on the text field next to **Variable Definitions** and click .
5. Define your variables in the **Editor for Variable Definitions** window.
6. Click **Ok** and **Save** to save your process.

Note that process variables should be mapped to local variables. See [Section 4.9.2, “Local Variables”](#) for more information.

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 **Data Input Assignments** and **Data Output Assignments** in the **Assignments** property. Data Input Assignment defines variables that enter the Task and therefore provide the entry data needed for the task execution. The Data Output Assignments can refer to the context of the Task after execution to acquire output data.

User Tasks present data related to the actor that is executing the User Task. Additionally, User Tasks also request the actor to provide result data related to the execution.

To request and provide the data, use task forms and map the data in the Data Input Assignment parameter to a variable. Map the data provided by the user in the Data Output Assignment parameter if you want to preserve the data as output. For further information, 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**.

4.9.3. Setting Process Variables From Business Rule Task

Process variables and rule facts do not share the same context. If a rule has to manipulate a process variable, you must explicitly map process variable to rule fact. You can access and set process variables from a business rule task using the following approaches:

- Mapping process Variables through Business Rule Task **Assignments** field
- Mapping process Variables through **WorkflowProcessInstance**

4.9.3.1. Mapping Process Variables through Business Rule Task Assignments field

The following example of a domain class called **ValidationError** containing a boolean attribute **isValid** illustrates mapping through the **Assignments** field:

1. Set a process variable called **validationError** of type **ValidationError**.
2. Instantiate the **ValidationError** object in the **ON ENTRY ACTION** field or in the **Script Tasks** placed before the **Business Rule Task**:

```
//Instantiate the object and set the flag to false
demo1.hello1.ValidationError validationError1 = new demo1.hello1.ValidationError();
validationError1.setIsValid(false);

//Assign the object to the process variable
kcontext.setVariable("validationError",validationError1);
```

1. In the Business Rule Task, click **Assignments** field and map the task variable in **DataInput** and **DataOutput**:
 - **Name:** **myvar**
 - **Data type:** **demo1.hello1.ValidationError**
 - **Source:** **validationError**
2. Edit the rules belonging to the **ruleflow-group** and assign it to the Business Rule Task:

```
rule "HelloAll"
dialect "mvel"
ruleflow-group "validate"
no-loop
when
  _myvar: ValidationError()
then
  _myvar.setIsValid( true );
  update( _myvar );
  System.out.println("The value returned is: " + _myvar.getIsValid());
end
```

Here, the rule is inserting the fact in the Business Rule Task through **DataInput** and binding it to **_myvar**. You can modify the **THEN** part of the rule and use it in your process as it is now mapped to **validationError** variable in **DataOutput**.

4.9.3.2. Mapping Process Variables through WorkflowProcessInstance

The following example of setting a process variable, which is used for group attribute in a Human Task, illustrates how you can map process variables through **WorkflowProcessInstance**:

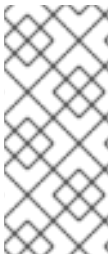
1. Create a process variable called **dynamicGroupId** with type **String**.
2. In the Human Task, set **Groups** attribute as **#{dynamicGroupId}**.
3. Put the Business Rule Task ahead of the Human Task and set the ruleflow group value to **dynamic-group**.
4. Create a rule under this ruleflow group. This rule sets the process variable **dynamicGroupId** dynamically based on its conditions. For example:

```
import org.kie.api.runtime.process.WorkflowProcessInstance;

rule "sampleRule"
  no-loop true
  ruleflow-group "dynamic-group"
  when
    $process : WorkflowProcessInstance( )
  then
    WorkflowProcessInstance $p =
    (WorkflowProcessInstance)kcontext.getKieRuntime().getProcessInstance($process.getId()); //casting
    to WorkflowProcessInstance is essential
    $p.setVariable( "dynamicGroupId","analyst" );
    retract($process);
```

The **WorkflowProcessInstance** object is not inserted into the ksession by default. You can insert it using the following:

```
kcontext.getKieRuntime().insert(kcontext.getProcessInstance());
```



NOTE

When a process instance is inserted into ksession as a fact, it can only be used to read values from it. This is because when using persistence, a process instance is considered read-only after a transaction is completed. You must reload the process instance before you attempt to modify it and once the work is done, retract it before the process is completed.

4.10. ACTION SCRIPTS

Action scripts are pieces of code that define the **Script** property of a Script Task 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 4.2. 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, Javascript, Drools, or MVEL, and set the language in the **Script Language** property.

4.12. ASSIGNMENT

The assignment mechanism enables 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 Data Input Assignments. Assignments that retrieve data from Activities are executed after the Activity has executed. They map from local data items in activities, known as Data Output Assignments, to business process variables.

4.12.1. Data I/O Editor


The Data I/O Editor is the dialog window 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.

PM Evaluation Data I/O
✕

Data Inputs and Assignments
+ Add

Name	Data Type	Source	
<input type="text" value="reason"/>	<input style="border: 1px solid #add8e6;" type="text" value="Object"/>	<input type="text" value="reason"/>	
<input type="text" value="performance"/>	<input type="text" value="Object"/>	<input type="text" value="performance"/>	

Data Outputs and Assignments
+ Add

Name	Data Type	Target	
<input type="text"/>	<input type="text"/>	<input type="text"/>	

Cancel Save

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**

Obtain Customer Info Data I/O



Data Inputs and Assignments

+ Add

Name	Data Type	Source	
invoice	Invoice[org.kie.te ▼	invoice ▼	
reason	String ▼	reason ▼	
maxamount	Float ▼	10000.00 ▼	
myvar	com.test.MyType ▼	myvar ▼	

Data Outputs and Assignments

+ Add

Name	Data Type	Target	
invoice	Invoice[org.kie.te ▼	invoice ▼	
reason	String ▼	reason ▼	
myvar	com.test.MyType ▼	myvar ▼	

Cancel Save

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 window.

The Data I/O Editor enables 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 window 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, Javascript, Drools, or MVEL, and have access to the data in the working memory, including the global and process variables.

Example 4.3. Java Code Constraint

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

Example 4.4. MVEL Code Constraint

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

Example 4.5. Javascript Code Constraint

```
kcontext.setVariable('surname', "tester");
var text = 'Hello ';
print(text + kcontext.getVariable('name') + '\n');
```

- **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.

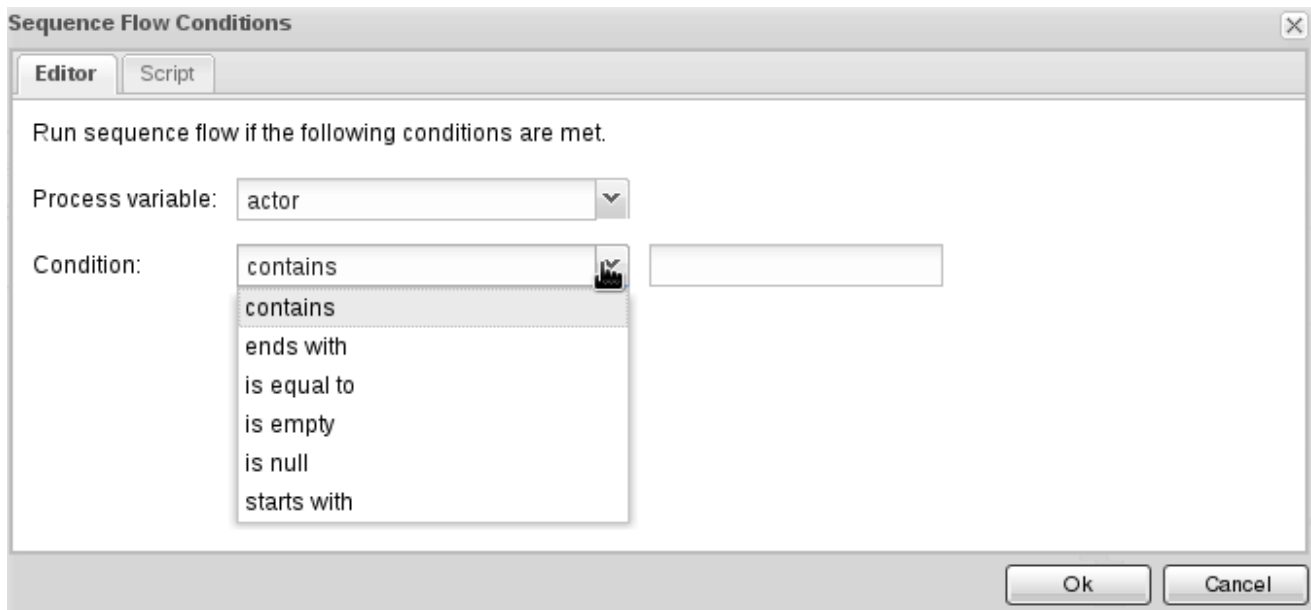
Example 4.6. Rule Constraint with Process Variable Assignment

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

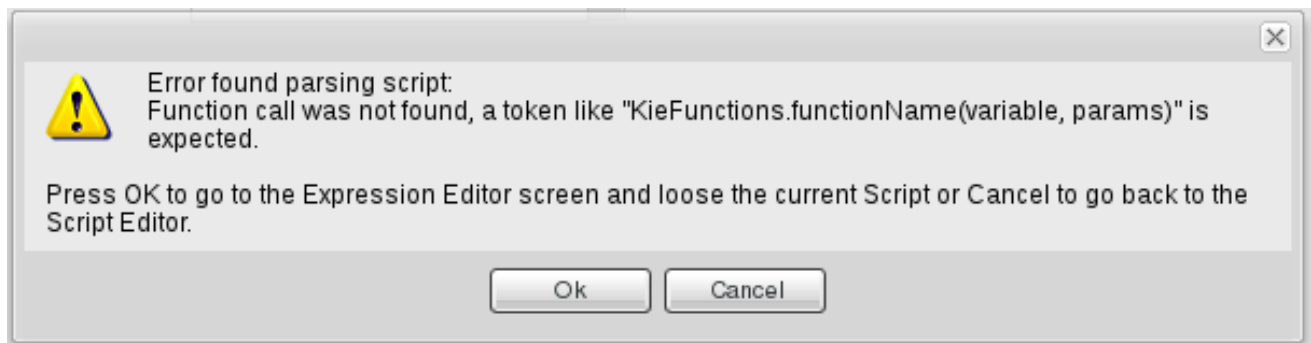
This rule constraint retrieves the process variable **name**.

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. 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).

4.14.1. 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 name of a service in the given work item set.

description

The description of a service.

version

A version number.

parameters

Defines service data inputs by specifying a name and a type. To define service data outputs, you can add a new property **results** that follows the same structure.

displayName

The name displayed in a palette.

icon

Refers to a file with the specified name that must be located in the same directory as the work item configuration file to be used by the import wizard. Icons are used in process diagrams. Icon is a GIF or PNG file with a size of 16x16 px.

category

Defines a category under which a service is placed when browsing the repository. If the defined category does not exist, a new category is created.

defaultHandler

Defines the default handler implementation, for example a Java class that implements the **WorkItemHandler** interface and can be used to execute the service. The class can be automatically registered as a handler when importing the service from a repository.

It is also possible to use MVEL to resolve the expression. MVEL provides the additional benefit of resolving handler's parameters. For example:

```
"defaultHandler" : "mvel: new org.jbpm.process.workitem.twitter.TwitterHandler(ksession)"
```

Available parameters are for example: **ksession**, **taskService**, **runtimeManager**, **classLoader**, and **entityManagerFactory**.

documentation

Refers to an HTML file with the specified name that must be located in the same directory as the work item configuration file. The file contains a description of the service.

dependencies

The dependencies for the **defaultHandler** class. It is usually the handler's implementation JAR, but the list can contain additional external dependencies as well.

Make sure you provide correct path to the files: use relative path to the directory where the work item configuration file is located.

If the dependencies are located in a Maven repository, you can define them in the **mavenDependencies** property:

```
"mavenDependencies" : [
  "org.jbpm:jbpm-twitter:1.0",
  "org.twitter4j:twitter4j-core:2.2.2" ]
```

4.14.2. 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"
]
```

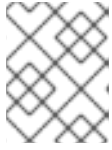


NOTE

The icon defined in the WID must be set and exist in your project. Otherwise, Red Hat JBoss Developer Studio does not display the custom task.

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:

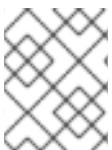
1. Click **New Item → Uploaded file**.
2. In the **Create new Uploaded file** dialog box, define the resource name, including file extension.
3. Click **Choose File** and upload the file (**png** or **gif**, 16x16 pixels).
4. Click **Ok**.

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

4.14.3. 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
 - JavalInvocationWorkItemHandler
 - 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>6.5.0.Final-redhat-2</version>  
</dependency>
```

To abort the work item, use the **WorkItemHandler.abortWorkItem()** before it is completed. For more information about asynchronous execution, see *Red Hat JBoss BPM Suite Development Guide* .

4.14.4. 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**.

4.14.5. Registering Work Item Handler Outside of Business Central

To register your Work Item Handler in the **kie-deployment-descriptor.xml** file:

1. Open the **PROJECT_HOME/META_INF/kie-deployment-descriptor.xml** file.
2. Locate the **<work-item-handlers>** tag.
3. Add your Work Item Handler, for example:

```
<work-item-handler>
  <resolver>mvel</resolver>
  <identifier>
    new org.jbpm.process.workitem.rest.RESTWorkItemHandler(classLoader)
  </identifier>
  <parameters/>
  <name>Rest</name>
</work-item-handler>
```

4. If your Work Item Handler uses a custom **JAR** file, include it in your **pom.xml** as a dependency.

Alternatively, if you use **RuntimeManager** directly, see the following example:

```
import java.util.Map;

import org.kie.api.KieServices;
import org.kie.api.io.ResourceType;
import org.kie.api.runtime.process.WorkItemHandler;
import org.kie.api.runtime.manager.RuntimeEngine;
import org.kie.api.runtime.manager.RuntimeEnvironment;
import org.kie.api.runtime.manager.RuntimeEnvironmentBuilder;
import org.kie.api.runtime.manager.RuntimeManagerFactory;
import org.jbpm.executor.impl.wih.AsyncWorkItemHandler;
import org.jbpm.runtime.manager.impl.DefaultRegisterableItemsFactory;

...

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);
```

- Implementations of the **org.kie.api.task.UserGroupCallback** interface are in the **org.jbpm.services.task.identity** package.
- Use CDI injection to get an instance of the **org.kie.api.executor.ExecutorService** interface. If your container does not support CDI injection, use factory **org.jbpm.executor.ExecutorServiceFactory**.

To include a custom **WorkItemHandler**, 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.

For a list of Maven dependencies, see example *Embedded jBPM Engine Dependencies* in chapter [Dependency Management](#) of the *Red Hat JBoss BPM Suite Development Guide* .



NOTE

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

4.15. SERVICE REPOSITORY

The service repository feature enables you to import an already existing service from a repository directly into your project. It allows multiple users to reuse generic services, 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.

If you connect to a service repository using its URL, a list of available provided services opens. Each of the listed services can then be installed into your project. If you install a service:

- The service configuration (work item definition file, **.wid**) is installed into the project as well. This file can later be edited. If there is already a work item definition file present, it will *not* be overwritten.
- A service icon defined in the service configuration is installed as well. If the icon does not exist, a default one is provided.
- The service's Maven dependencies are added into the project's **pom.xml** file.
- The service default handler is added into the project's deployment descriptor.



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 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.

4.15.1. Installing Services from Service Repository

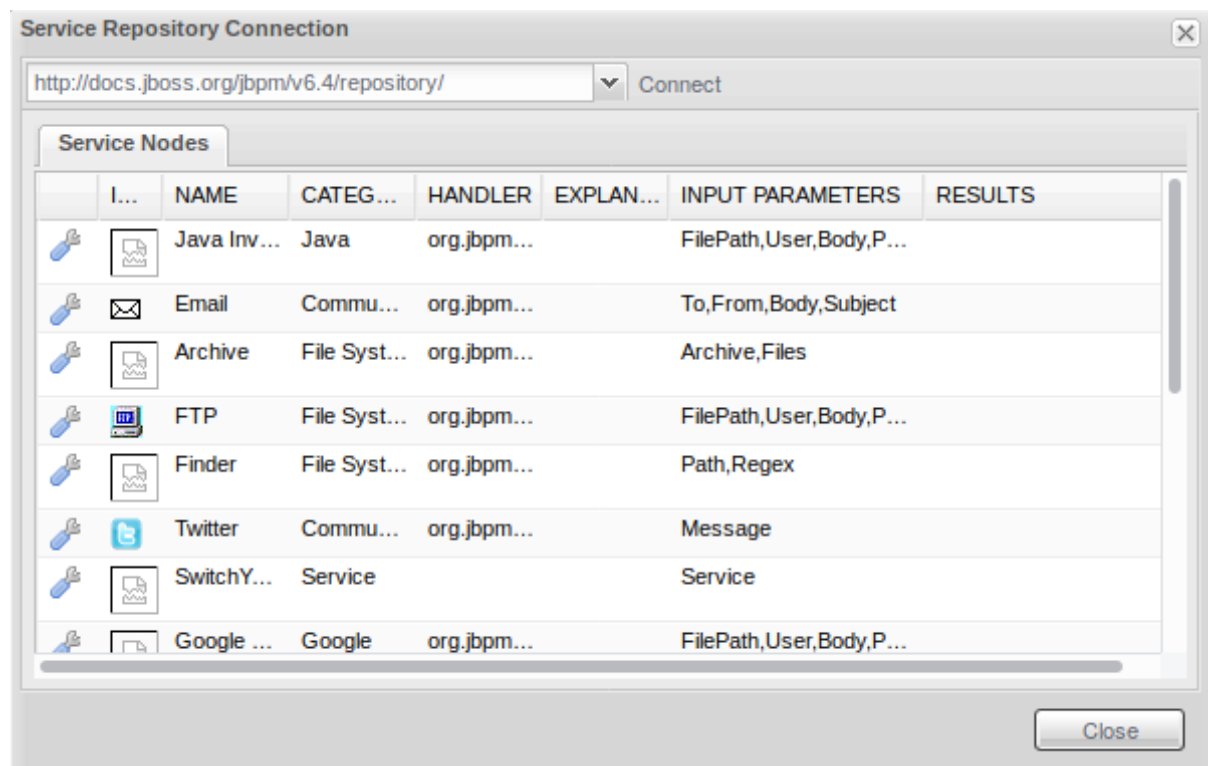
There are two ways of installing services from a service repository: using Process Designer in Business Central or during the Business Central startup process.

Installing Services in Process Designer

To import a work item from a service repository directly in Business Central, do the following:

1. Open your process in Process Designer.
2. In the editor menu, click **Connect to a Service Repository** (🔑).
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. To install an asset, click next to the asset you want to install. After the service is successfully installed, a notification will appear on the screen. To start using the service, save and reopen your process.

Installing Services During Business Central Startup

The automatic installation enables you to specify the repository URL and a list of services to be installed during the Business Central startup process. The services are then ready for use after you create or open a process in Process Designer.



NOTE

Make sure you have the correct service names specified in the service's **.wid** file ready.

To install a service (for example Twitter) from the repository located at <http://docs.jboss.org/jbpm/v6.4/repository/>, start the server using the following command:

```
./standalone.sh -Dorg.jbpm.service.repository=http://docs.jboss.org/jbpm/v6.4/repository/ -
Dorg.jbpm.service.servicetasknames=Twitter
```

You can specify more services at once by separating them with a comma. Install-all option is not currently available.

```
./standalone.sh -Dorg.jbpm.service.repository=http://docs.jboss.org/jbpm/v6.4/repository/ -
Dorg.jbpm.service.servicetasknames=Twitter,Jabber
```



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.

4.15.2. 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 folders to be processed when searching for services in the service repository.

Example 4.7. index.conf

```
Email
FileSystem
ESB
FTP
Google
Java
Jabber
Rest
RSS
Transform
Twitter
```

Each directory can contain another **index.conf** file. In that case, a new hierarchical structure is created and additional subfolders are scanned. 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 Item Configuration File

Directories with work items must contain:

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

A *work item configuration file* is a file with the same name as the parent directory, for example **Twitter.wid**, 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 4.14.1, “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 (MVEL)

```
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.SampleCustomEditor",
    "icon" : "twitter.gif",
    "category" : "Communication",
    "defaultHandler" : "org.jbpm.process.workitem.twitter.TwitterHandler",
    "documentation" : "index.html",
    "dependencies" : [
      "file:./lib/jbpm-twitter.jar",
      "file:./lib/twitter4j-core-2.2.2.jar" ]
  ]
]
```

When creating a work item configuration file, it is also possible to use JSON instead of MVEL. See the previous example written in JSON:

Example 4.9. Work Item Configuration File (JSON)

```
[
  [
    "java.util.HashMap",
    {
      "name": "TestServiceFour",
      "displayName": "Twitter",
      "description": "Send a Twitter message",
      "parameters": [
        "java.util.HashMap",
        { "Message": ["org.drools.core.process.core.datatype.impl.type.StringDataType", {}] } ],
    }
  ]
]
```

```
"eclipse:customEditor": "org.drools.eclipse.flow.common.editor.editpart.work.SampleCustomEditor",

    "defaultHandler" : "org.jbpm.process.workitem.twitter.TwitterHandler",
    "documentation" : "index.html",
    "dependencies": [
        "java.util.ArrayList", ["file:./lib/jbpm-twitter.jar", "file:./lib/twitter4j-core-2.2.2.jar"] ]
    }
}
]
```

4.15.3. Retrieving Service Repository Information

Classes provided in the **org.jbpm.process.workitem** package allow you to connect to the service and retrieve service information. For example, to list all the services contained in a repository and declared in **index.conf**, use:

```
Map<String, WorkDefinitionImpl> workitemsFromRepo =
    WorkItemRepository.getWorkDefinitions("http://docs.jboss.org/jbpm/v6.4/repository/");
```



NOTE

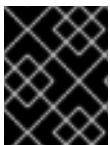
In the following text, Twitter is used as an example service. To interact with a different service, replace **Twitter** with a name declared in the service **.wid** file.

To get more detailed information about a service, use:

```
workitemsFromRepo.get("Twitter").getName(); // "Twitter"
workitemsFromRepo.get("Twitter").getDescription(); // "Send a Twitter message."
workitemsFromRepo.get("Twitter").getDefaultHandler(); //
"org.jbpm.process.workitem.twitter.TwitterHandler"
workitemsFromRepo.get("Twitter").getDependencies(); // String["file:./lib/jbpm-
twitter.jar", "file:./lib/twitter4j-core-2.2.2.jar"]
...
```

To check whether the correct version of a service is contained in the repository:

```
if(workitemsFromRepo.containsKey("Twitter") &&
workitemsFromRepo.get("Twitter").getVersion().equals("1.0")) {
    // Do something here.
}
```



IMPORTANT

All operations are read-only. It is not possible to update the service repository automatically.

4.16. 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.

4.17. 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 **ldaps://localhost:636**)

4.17.1. Connecting to LDAP

To be able to use the LDAP UserGroupCallback implementation configure the respective LDAP properties (see [Section 4.17, "LDAP connection"](#)) in one of the following ways:

- *programmatically*: 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(ldapUserGroupCallback);
```

- *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=
```

4.18. 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 23.1, "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 *Red Hat JBoss BPM Suite Development Guide*).

CHAPTER 5. 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.

5.1. 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.

Person.java - Data Objects

Save Delete Rename Copy Validate Latest Version

Editor Overview Source

Identifier	Label	Type	
firstName		String	Delete
hourlyRate		Integer	Delete
lastName		String	Delete
wage		Integer	Delete

'firstName'- general properties

Identifier: firstName

Label:

Description:

Type: String

List:

- 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.

The screenshot shows the 'Person.java - Data Objects' window in the Data Modeler. The 'Person' data object is selected and highlighted with a red box. The interface displays a table of fields and their properties, along with a 'Person'- general properties panel.

Identifier	Label	Type	
firstName		String	Delete
hourlyRate		Integer	Delete
lastName		String	Delete
wage		Integer	Delete

'Person'- general properties

- Identifier: Person
- Label:
- Description:
- Package: org.brms.myproject
- Superclass: java.lang.Object

5.2. AVAILABLE FIELD TYPES

Data object fields can be assigned to any of the following types:

- **Java Object Primitive Types:**
BigDecimal, BigInteger, Boolean, Byte, Character, Date, Double, Float, Integer, Long, Short, and **String**.
- **Java Primitive Types:**
boolean, byte, char, double, float, int, long, and **short**.
- **Java Enum Types:**
Java enum types defined in current project or imported as a dependency. See [Adding Dependencies](#).
- **Current project Data Objects:**
Any user defined data object automatically becomes available to be assigned as a field type.
- **Project Dependencies:**
Other Java classes imported as a Java dependency in current project. See [Adding Dependencies](#).

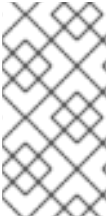
5.3. ANNOTATIONS IN DATA MODELER

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 5.6, "Data Object Domain Screens"](#).

To add or edit custom or pre-defined annotations, switch to the **Source** tab and modify the source code directly. You can edit the source code directly in both Red Hat JBoss Developer Studio and Business Central. 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 on 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.

5.4. CREATING A DATA OBJECT

1. In the Project Authoring perspective, click **New Item → Data Object** on the perspective menu.
2. Enter the name and select the package. The name must be unique across the package, but it is possible to have two data objects with the same name in two different packages.
3. To make your Data Object persistable, check the **Persistable** checkbox.
4. Click **Ok**.
5. Create fields of the data object:
 - a. Click **add field** in the main editor window to add a field to the object with the attributes **Id**, **Label** and **Type**. Required attributes are marked with *.

- **Id**: The ID of the field unique within the data object.
- **Label**: The label to be used in the **Fields** panel. This field is optional.
- **Type**: The data type of the field.

New Field
✕

Id *

Label

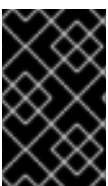
Type *

List ⓘ

Cancel
Create
Create and continue

- b. Click **Create** to create the new field and close the **New field** window. Alternatively, click **Create and continue** to keep the **New field** window open.

To edit an attribute, select the attribute and use the general properties screen.



USING A DATA OBJECT

To use a data object, make sure you import the data model into your resource. Unless both the data model and your resource, for example a guided rule editor, are in the same package, this is necessary even if both are in the same project.

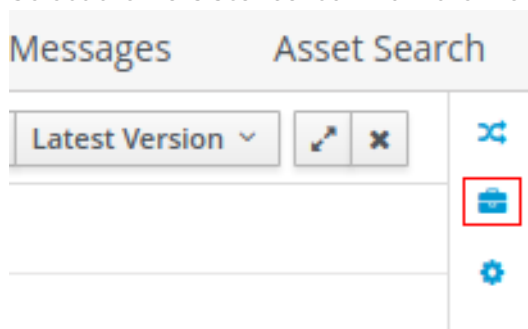
5.5. PERSISTABLE DATA OBJECTS

From Red Hat JBoss BPM Suite 6.2 onwards, the Data Modeler supports the generation of persistable data objects. Persistable data objects are based on the JPA specification. When you check the **Persistable** checkbox, the platform will use default persistence settings. You can make a data object persistable in two ways:

- When creating a new data object.
When creating a new object, follow the procedure in [Section 5.4, “Creating a Data Object”](#).
- When a data object has already been created.

To make an already created data object persistable:

1. Open your data object in Business Central.
2. Click the **Editor** tab.
3. Select the **Persistence** icon from the menu on the right:



4. Check **Persistable**.
5. Click **Save** to save your changes.

5.6. 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 editing of the 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**
- **ClassReactive**
- **PropertyReactive**
- **Role**
- **Timestamp**
- **Duration**
- **Expires**

- **Remotable**

Figure 5.1. The Drools & jBPM Class View

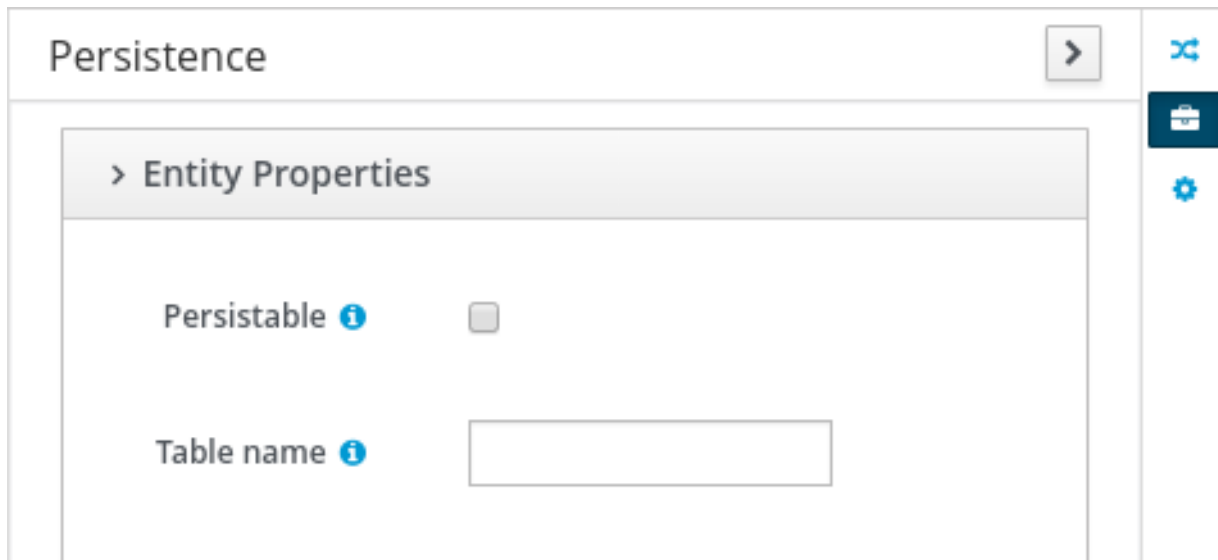
For the fields within the fact model, the **position** and **Equals** annotations are supported. The **Drools & jBPM** screen when a specific field is selected looks as follows:

Figure 5.2. The Drools & jBPM Field View

Persistence

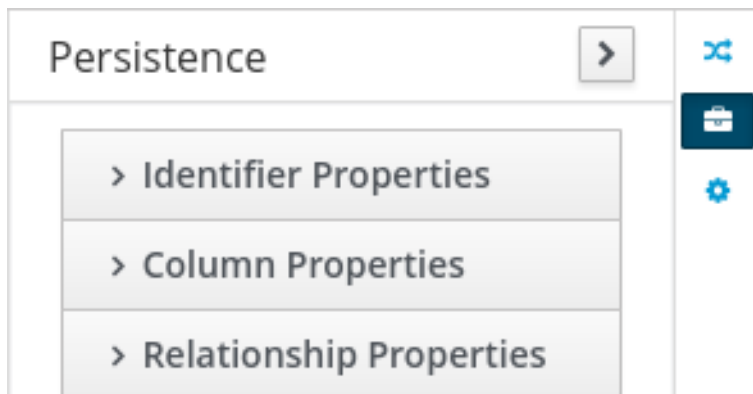
The **Persistence** screen can be used to configure attributes on basic JPA annotations for persistence. For fine tuning of annotations, or to add specific annotations, use the **Advanced** screen.

Figure 5.3. The Class Persistence View



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

Figure 5.4. The Field Persistence View



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

Table 5.1. Type Annotations

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

Table 5.2. Field Annotations

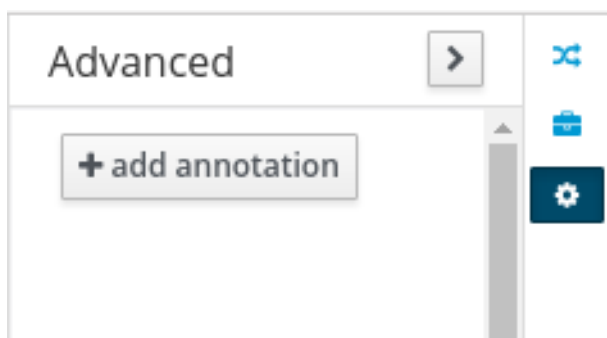
Annotation	Automatically Generated when the Data Object is Persistable	Responsible UI Element
javax.persistence.Id	Yes	Is Identifier
javax.persistence.GeneratedValue	Yes	Generation strategy

Annotation	Automatically Generated when the Data Object is Persistable	Responsible UI Element
javax.persistence.SequenceGenerator	Yes	Sequence Generator
javax.persistence.Column	No	Column Properties attributes
javax.persistence.OneToOne	No	Relationship Type
javax.persistence.OneToMany	Yes - when a field has one or multiple values	Relationship Type
javax.persistence.ManyToOne	Yes - when a field has multiple values	Relationship Type
javax.persistence.ManyToMany	No	Relationship Type
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).	Created by a field marked as list .

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 Domain 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 *).

Add new Annotation
✕

Search annotation

-> cascade

-> fetch

-> optional

-> targetEntity

Annotation class name *

Q

Annotation definition was loaded successfully.

< Previous
Next >
Cancel
✓ Finish

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

Add new Annotation
✕

Search annotation

-> cascade

-> fetch

-> optional

-> targetEntity

cascade

ALL

PERSIST

MERGE

REMOVE

REFRESH

DETACH

{}

< Previous
Next >
Cancel
✓ Finish

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

Add new Annotation
✕

Search annotation

-> cascade

-> fetch

-> optional

-> targetEntity

targetEntity


1

Enter an optional value for the annotation value pair and press the validate button

< Previous
Next >
Cancel
✓ Finish

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.

5.7. 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.

Relationship configuration
✕

Relationship type ▼

Many to One

Cascade mode

All
 Persist
 Merge
 Remove
 Refresh
 Detach

Fetch mode ▼

EAGER

Optional

+ Ok

Cancel

Attempting to delete a data object that is used by a different 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.

5.8. PERSISTENCE DESCRIPTOR

Business central contains a **persistence.xml** file with default persistence settings. To configure persistence settings, click **Project Settings: Project General Settings** → **Persistence descriptor**.

🔒 persistence.xml - Persistence descriptor ▼

Save Delete Rename Copy Validate Latest Version ▼ ↕ ✕

Editor
Overview
Source

Persistence Unit input field

Persistence Provider input field







Data Source input field

Transactions Type JTA

> Advanced properties

> Project persistable Data Objects


Use the **Advanced properties** section to change or delete or add properties.

▼ Advanced properties		
Property Name	Property Value	Action
hibernate.dialect	org.hibernate.dialect.H2Dialect	 Delete
hibernate.max_fetch_depth	3	 Delete
hibernate.hbm2ddl.auto	update	 Delete
hibernate.show_sql	false	 Delete
hibernate.id.new_generator_mappings	false	 Delete
hibernate.transaction.jta.platform	org.hibernate.service.jta.platform.internal.JBossAppServerJtaPlatfor...	 Delete

> New property

If you open the **Project persistable Data Objects** section in the Persistence Descriptor, you will see two 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 persistable data objects in the current project.

▼ Project persistable Data Objects	
Class name	Action
No classes selected	
	
<input type="text" value="enter a persistable class name"/>	<input type="button" value="Add class"/> <input type="button" value="Add project persistable classes"/>

5.9. DEPLOYMENT DESCRIPTOR

Deployment Descriptor editor can also be accessed through 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.

kie-deployment-descriptor.xml - Deployment Save Validate Latest Version

Deployment descriptor editor

Runtime strategy
SINGLETON

Persistence unit name
org.jbpm.domain

Persistence mode
JPA

Audit persistence unit name
org.jbpm.domain

Audit mode
JPA

Marshalling strategies

Value	Resolver type	Parameters	Remove
new org.drools.persistence.jpa.marshaller.JPAPlaceholderResolverStrategy("com.redhat.test-project-1.0", classLoader)	mvel	Parameters(0)	Remove

[Add](#)

DropDownScreen JPADataSourceScreen AdvancedDataSourceScreen

CHAPTER 6. ADVANCED PROCESS MODELING

6.1. 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.

Using an XML editor

You can use any XML or text editor to create a process specification using the BPMN2 XML schema. See the [Defining Processes Using XML](#) chapter of the *Red Hat JBoss BPM Suite Development Guide* for further information.

Using the Process Fluent API

You can use the Red Hat JBoss BPM Suite API directly. The most important process model elements are defined in the following packages:

- **org.jbpm.workflow.core**
- **org.jbpm.workflow.core.node**

See the [Process Fluent API](#) chapter of the *Red Hat JBoss BPM Suite Development Guide* for further information.

6.2. WORKFLOW PATTERNS

Workflow patterns are predefined blocks of process elements that enable you to reuse a predefined combination of process elements. Workflow patterns 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 Workflow Patterns section of the Object Library and can be dragged and dropped onto the canvas just like any other elements. To attach a pattern to an element on the canvas, select the element and drag and drop the pattern from the palette onto the canvas. The pattern automatically connects to the element.

Multiple predefined workflow patterns are provided by default and you can define your own workflow patterns as necessary.

The definitions are defined as JSON objects in the ***EAP_HOME/standalone/deployments/business-central.war/org.kie.workbench.KIEWebapp/defaults/patterns.json*** file.

6.2.1. Defining workflow patterns

To define custom workflow patterns:

1. In the **Workflow Patterns** section of the **Object Library**, locate a workflow pattern that will serve as a base for your workflow pattern.
2. Open the ***EAP_HOME/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.

6.2.2. Changing workflow patterns for an existing project

You can change the **patterns.json** file as needed for projects.

1. In Business Central, open the process in the Process Editor.
2. Open the **Project Explorer** panel on the left side of the editor and switch to the **Repository View**.
3. Expand the **global** folder and download and modify the **patterns.json** file as needed.
4. Upload the revised file.
5. Close and reopen the business process. You can view the new patterns under the the **Workflow Patterns** section of the **Object Library**.

CHAPTER 7. 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 8. 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.

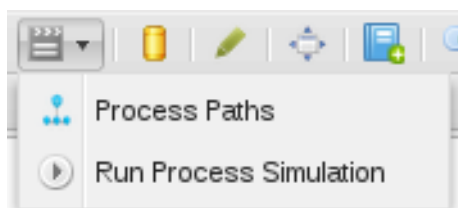
The simulation process runs in the simulation engine extension, which relies on the possible execution paths rather than process data. On simulation, the engine generates events for every simulated activity, which are stored in the simulation repository.

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 [Chapter 34, Process](#) and the subsequent sections.

8.1. 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**.

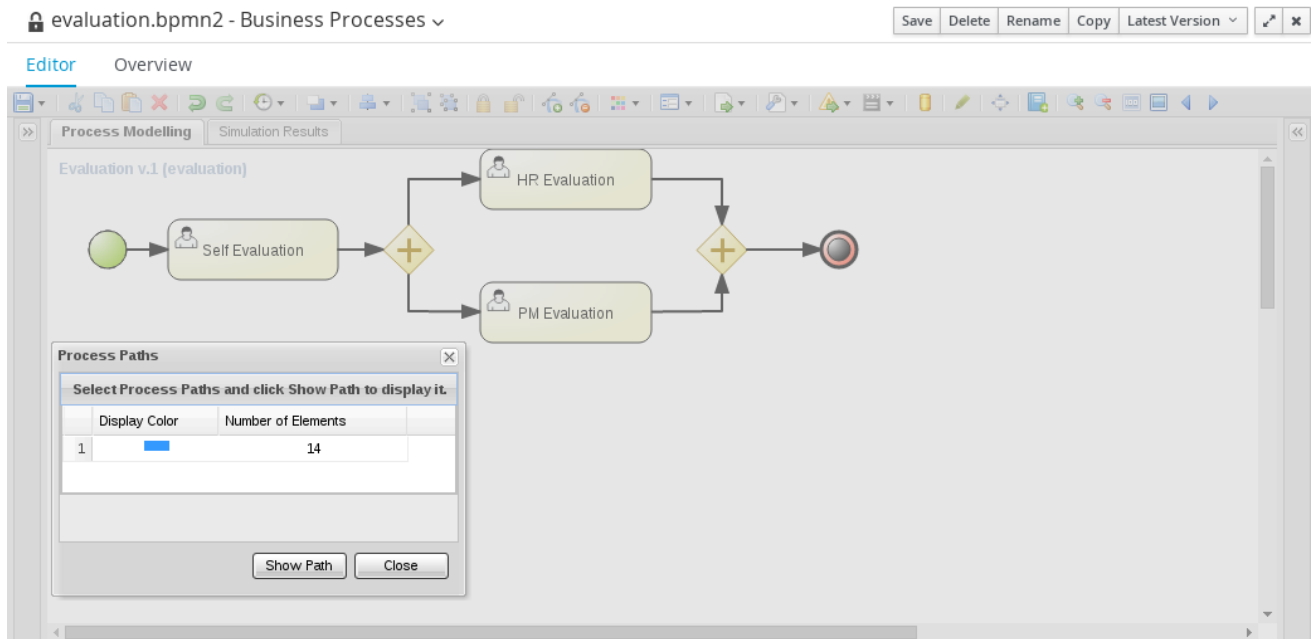


NOTE

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 8.1. Process Paths



8.2. SIMULATING PROCESSES

8.2.1. Defining Simulation Data on Elements

To run a process simulation, the input simulation data must be specified for the process and each of its elements. To specify the simulation data:

1. In Business Central, open the process in the Process Editor.
2. Open the **Properties** panel on the right.
3. For each individual element and the process itself, specify the simulation data in the **Simulation Properties** section.

Figure 8.2. Simulation Properties

Simulation Properties	
Cost per time unit	20.0
Distribution Type	normal
Processing time (mean)	10.0
Staff availability	20.0
Standard Deviation	1.0
Working Hours	8.0

4. Save the process.

For more information about the simulation data for individual process elements, see [Appendix C, Simulation Data](#).

8.2.2. Running Process Simulations


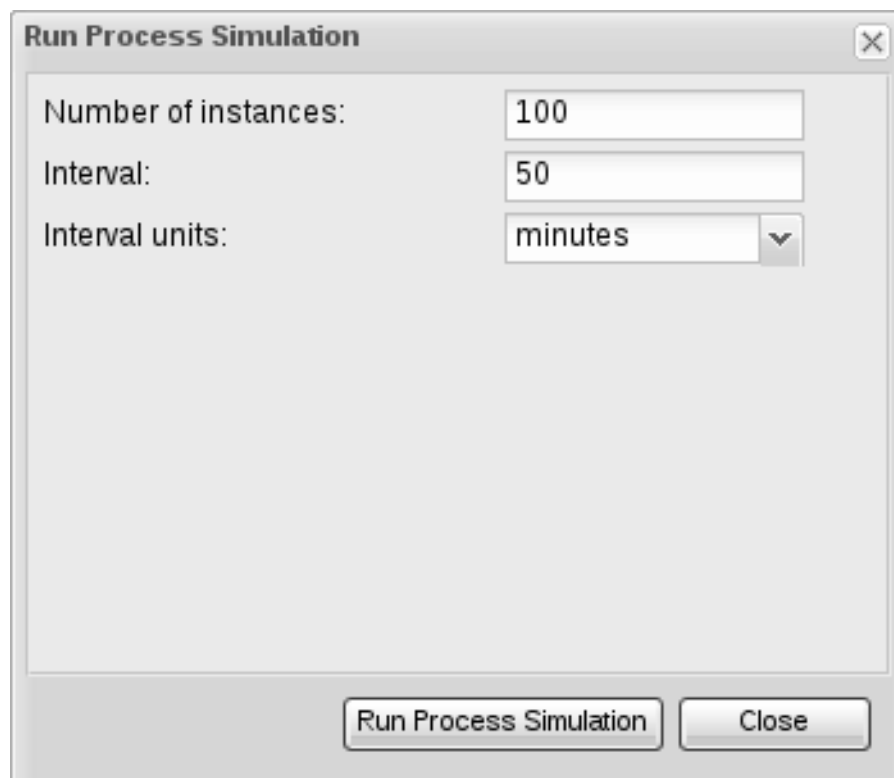
1. Open the corresponding process in the Process Editor.
2. Click  **Run Process Simulation**.
3. In the **Run Process Simulation** dialog window, define the simulation session details:
 - **Number of instances:** the number of process instances the simulation creates and triggers.
 - **Interval:** the interval between individual process instantiations.
 - **Interval units:** the time unit in which you defined the interval.

Figure 8.3. Run Process Simulation Dialog Window



4. Click **Run Process Simulation**.

After the simulation starts, the **Simulation Results** tab opens. The tab includes the **Simulation Graphs** panel where you can select the process, different process elements, or paths to view the corresponding results. For more information, see [Section 8.2.3, “Examining Simulation Results”](#).

If the simulation fails, a notification message is displayed. To prevent the simulation from failing, make sure the process is valid and not too complex. You can split complex processes into multiple processes. See the following examples of complex processes:

- Processes that contain a complex series of XOR and AND gateways.
- Processes that contain a loop where one instance of the loop is not finished before a new instance starts.

8.2.3. Examining Simulation Results

In the **Simulation Graphs** panel that opens after you run a simulation, the results are divided into the following categories:

Category	Contains
Process	Graphs with general results of a process simulation.
Process Elements	Simulation results for individual elements. Each Human Task element in this category contains the following graphs: <ul style="list-style-type: none"> ● Execution Times with the maximum, minimum, and average execution times for the given Human Task. The graph is also available for Script Tasks and Intermediate Events. ● Resource Utilization contains information about resource allocations. ● Resource Cost with the maximum, minimum, and average resource costs. To display the graph correctly, set the Cost per time unit property.
Paths	Simulation results of the paths used during the simulation.

8.2.3.1. Switching Between Graph Types

To change the type of the displayed graphs, click the corresponding icon at the upper right hand corner of the Process Editor. The available graph types are:







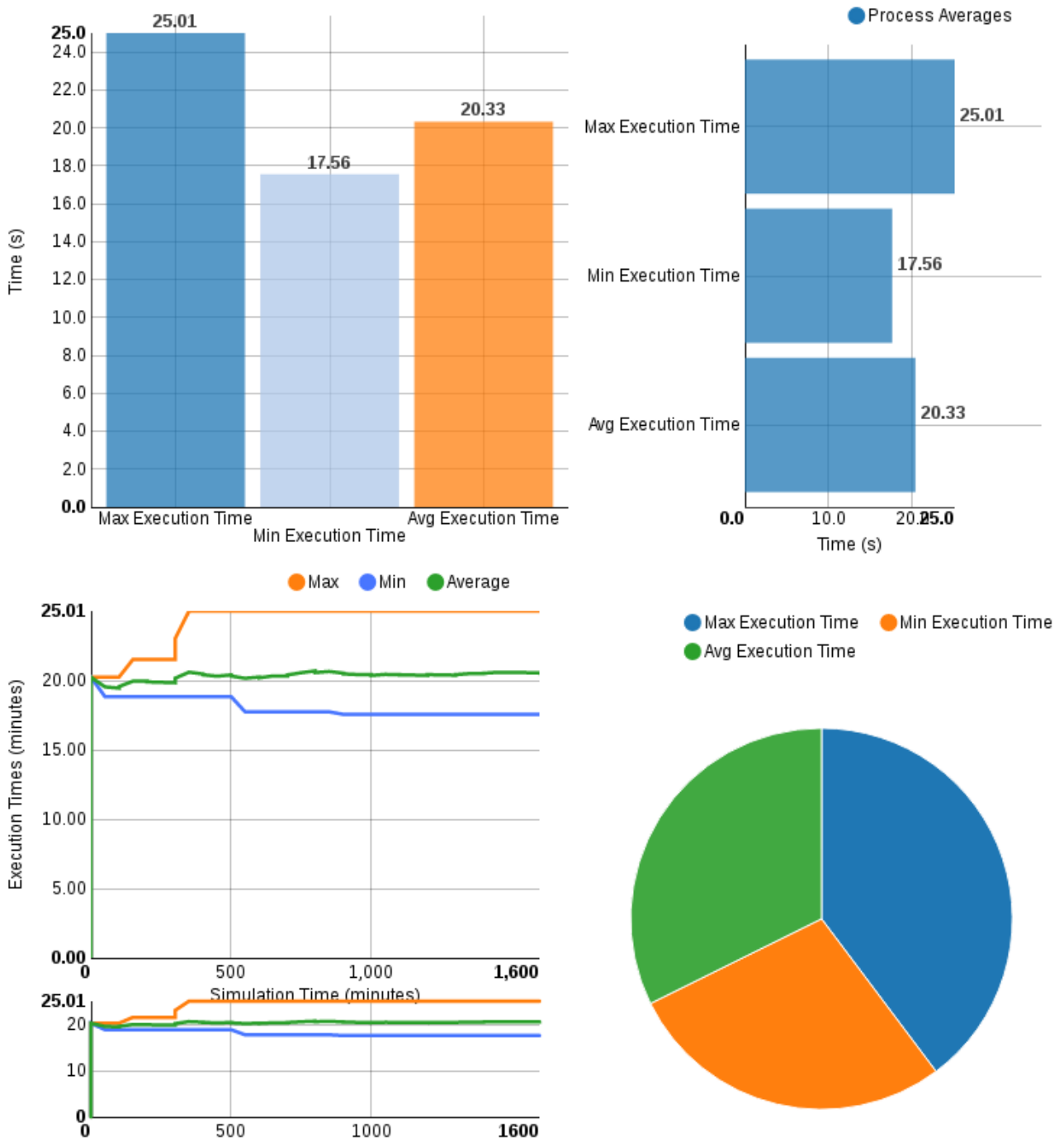
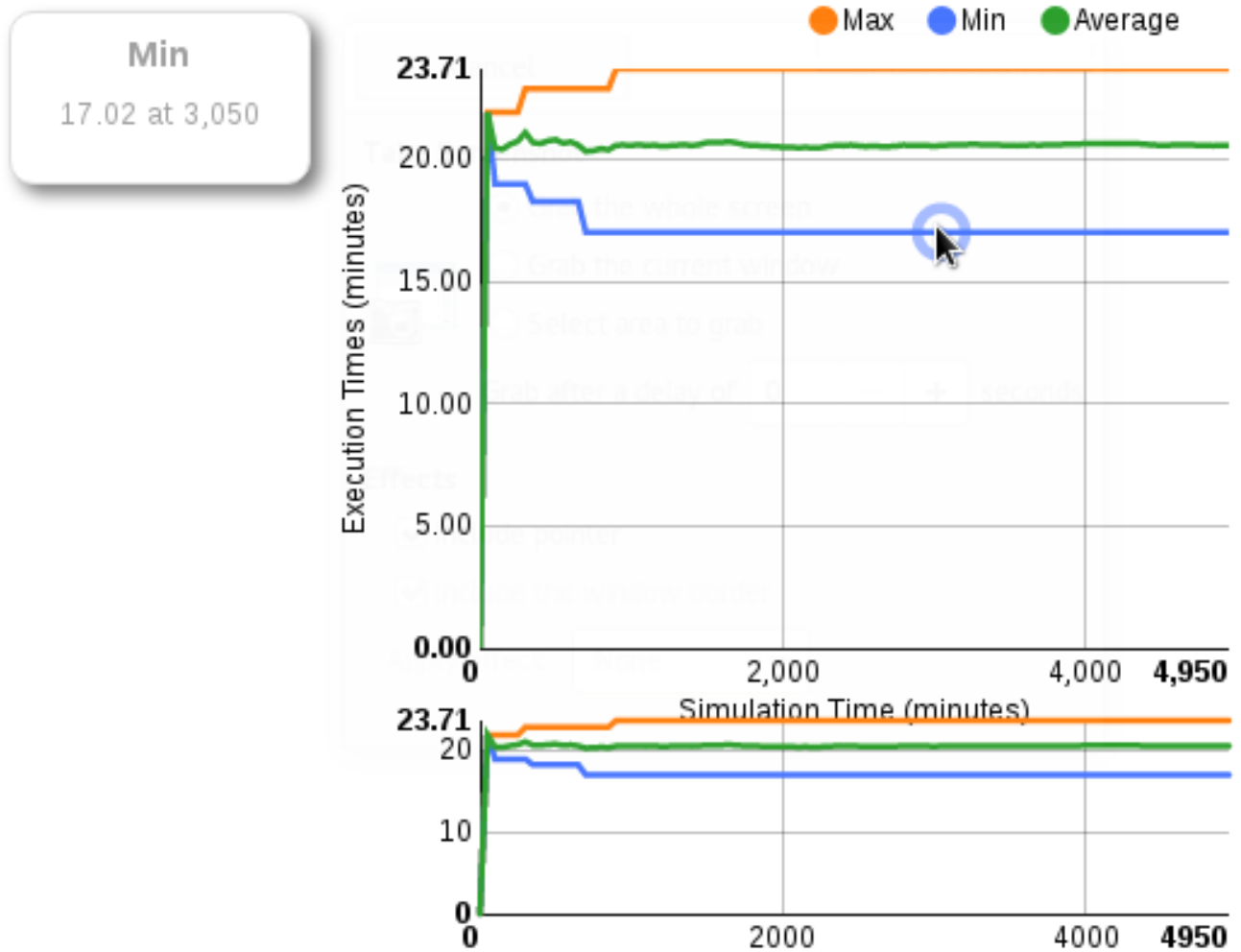
-  Bar Chart
-  Horizontal Bar Chart
-  Pie Chart
-  Table
-  Timeline
-  Line Chart

Figure 8.4. Different Types of Simulation Graphs



In line charts, point to a particular place on a line to view the value of the item at the given time.

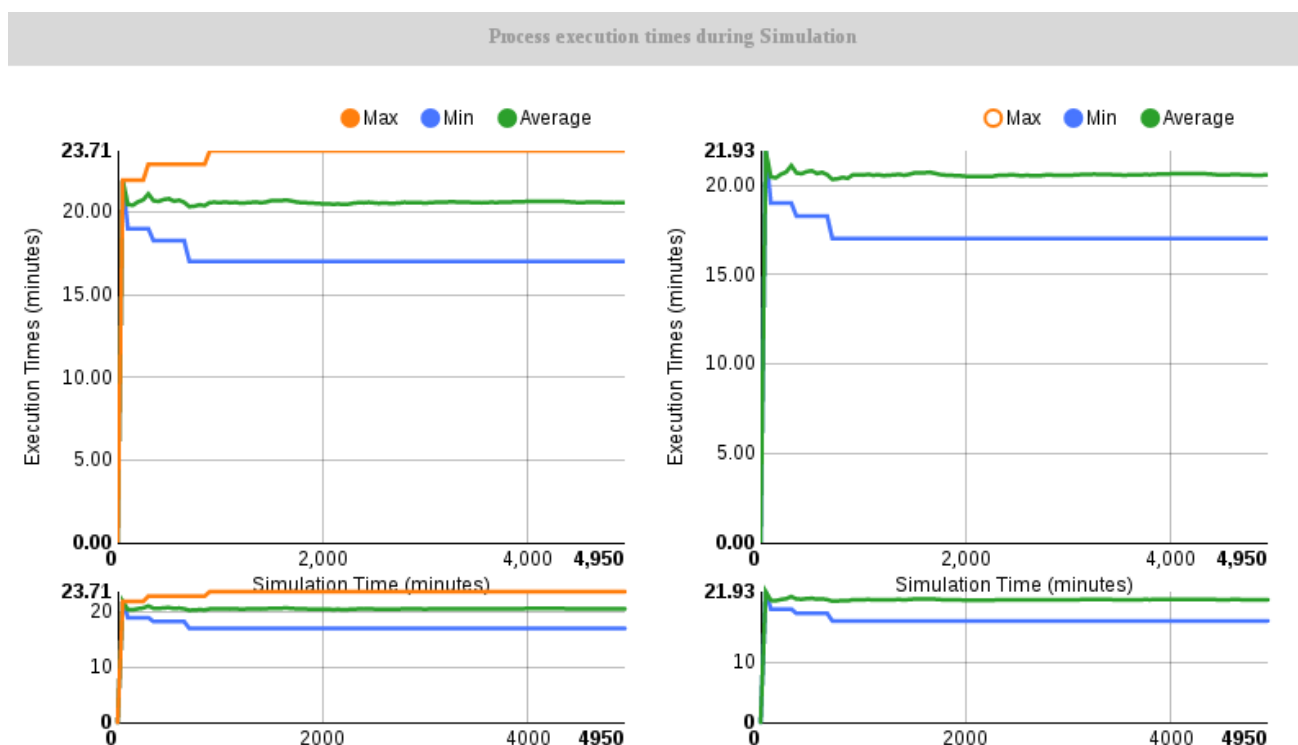
Figure 8.5. Line Chart



8.2.3.2. Filtering in Graphs

To filter the displayed data in a chart, click the corresponding coloured radio button in the chart legend.

Figure 8.6. Filtering the Maximum Value



8.2.3.3. Viewing Graph Timeline

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


To activate the feature, click  at the upper right hand corner. After the timeline opens, you can click the arrows on the right and left from the chart to move through the timeline. The data for the particular moment are applied to the chart instantly.

Figure 8.7. Process Simulation Timeline



CHAPTER 9. TESTING

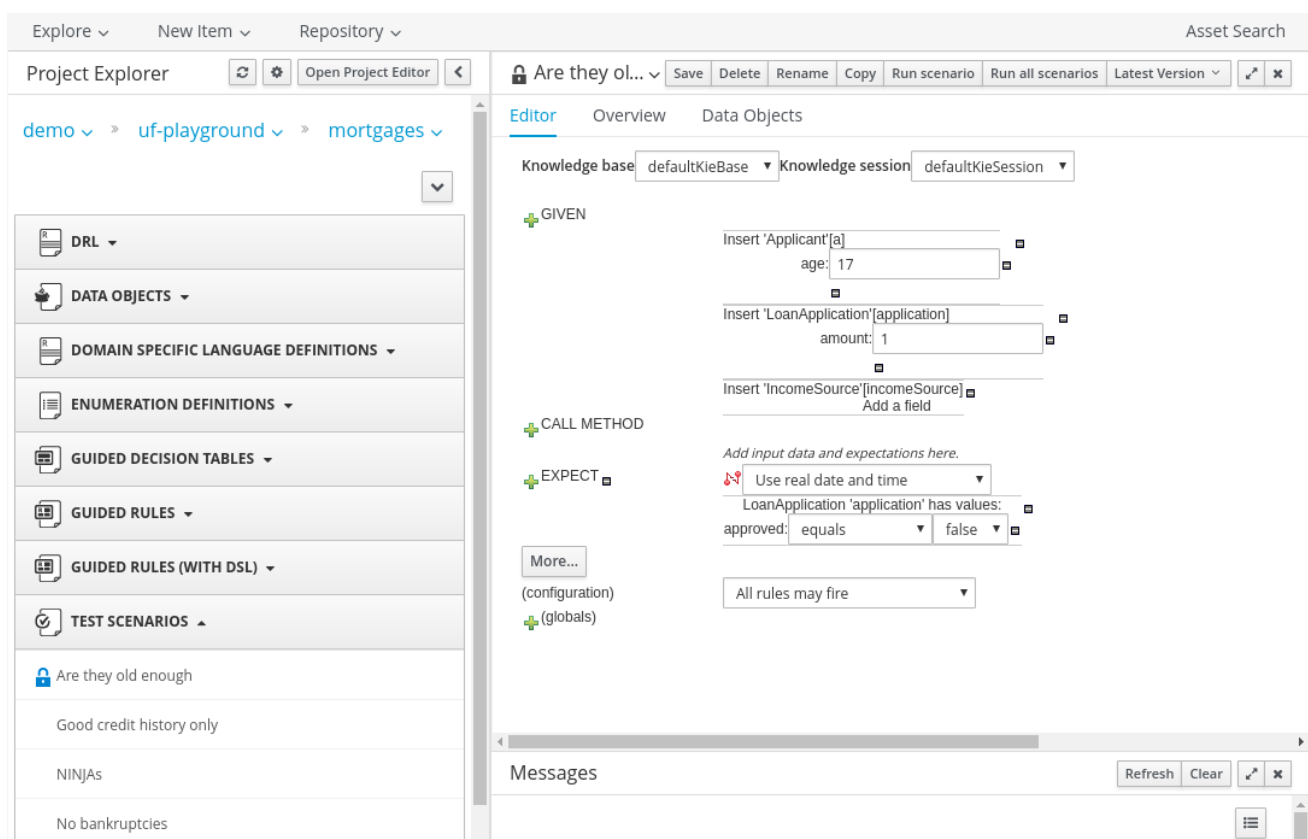
9.1. TEST SCENARIOS

Test Scenarios is a powerful feature that provides the ability for developers to validate the functionality of rules, models, and events. In short, Test Scenarios provide you the ability to test your knowledge base before deploying it and putting it into production.

Test Scenarios can be executed one at the time or as a group. The group execution contains all the Scenarios from one package. Test Scenarios are independent, one Scenario can not affect or modify the other.

After running all the Test Scenarios a report panel is shown. It contains either a success message or a failure message for test scenarios that were run.

Figure 9.1. Test Scenario Screen



9.2. CREATING A TEST SCENARIO

Creating a Test Scenario requires you to provide data for conditions which resemble an instance of your fact or project model. This is matched against a given set of rules and if the expected results are matched against the actual results, the Test Scenario is deemed to have passed.

Creating a new Test Scenario

1. In Business Central, click **Authoring** → **Project Authoring** to open the **Projects** view.
2. Select a project for your test scenario.
3. Click **New Item** → **Test Scenario**.

4. Enter the name, select the package, and click **OK**.
5. You will be presented with the Test Scenario edit screen.

Importing a model for the Test Scenario

Data objects from the same package are available by default. For example, given the package structure **org.company.project**, and the following:

- A data object *Fact1* in package **org.company**.
- A *Fact2* in package **org.company.project**.

If you create your test scenario in **org.company**, **org.company.Fact1** is available but you must import **org.company.Fact2**. To import data objects:


1. Open your test scenario.
2. Click the **Data Objects** tab.
3. Click **New Item**, select your import and click **Ok**. The imports can be specific to your project's data model or generic ones like **String** or **Double** objects.

Providing Test Scenario Facts

1. After importing data objects, click the **Editor** tab. At minimum, there are two sections that require input: **GIVEN** and **EXPECT**.
 - **GIVEN**: The input facts for the test.
 - **EXPECT**: The expected results given the input facts.
GIVEN the input parameters, **EXPECT** these rules to be activated or fired. You can also **EXPECT** facts to be present and to have specific field values or **EXPECT** rules not to fire at all.

If the expectations are met, the test scenario has passed and your rules are correct. Otherwise, the test scenario fails.

Providing Given Facts

- To add a new fact, click  next to the **GIVEN** label. Provide your fact data in the **New Input** dialog window based on the data models that you have imported in the **Data Objects** tab.

New Input
✕


Insert a new fact: Bankruptcy Fact name: Add


Modify an existing fact: a Add

Delete an existing fact: application Add


Activate rule flow group: Add

You can select a particular data object from the model and give it a variable name, called **Fact Name** in the window, or choose to activate a rule flow group instead. Activating a rule flow group allows rules from the specified rule flow group to be tested by activating the group in advance. To add a given fact and activate a rule flow group:

- Add the given fact.
 - Click  again and add the rule flow group activation.
- Optionally, add restrictions on the object you will insert.
 - Click **Add a field** and select a property of your object.

Insert 'Customer' [Cust] ✖
 hasInternetService:  ✖

✖

- Click  next to the property.
 - Click **Create a new fact** if the property type is another fact object.
 - Click **Literal value** otherwise.
See [Section 9.3, "Additional Test Scenario Features"](#) for more information.
- Provide the value. For example:

Insert 'Customer' [Cust] ✖
 hasInternetService: true ✖

✖

The example above is equivalent to the following:

```
Customer fact1 = new Customer();
fact1.setHasInternetService(true);
insert(fact1);
```

Providing Expected Rules

- Once you are satisfied with the **GIVEN** conditions, you can expect rules that will be fired, facts created, or field values in existing facts changed. Click **+** next to the **EXPECT** label to start adding expected results.

New expectation
✕

Rule: -- please choose -- ▼ OK

Fact value: ▼ Add

Any fact that matches: ▼ Add

- You can provide one of three expectations given the set of data that was created in the Given section:

- **Rule:** enables you to check for firing of a particular rule. Either type the name of a rule that is expected to be fired or select it from the list of rules. Click the **OK** when done.
- **Fact value:** enables you to check a specific object instance and its values. In the following example, given a Customer object with the **hasInternetService** boolean set to **true**, we expect the same object to have the **hasPhoneService** boolean set to **true**:

Customer 'Cust' has values:

hasPhoneService: ▼ ▼ ✕

- **Any fact that matches:** enables you to check any objects in the working memory and the values of their field. In the following example, given a Customer object which has internet service, a new object RecurringPayment is expected to be inserted into the working memory with the **amount** field set to **5**:

A fact of type 'RecurringPayment' has values:

amount: ▼ ▼ ✕

Reviewing, Saving, and Running a Scenario

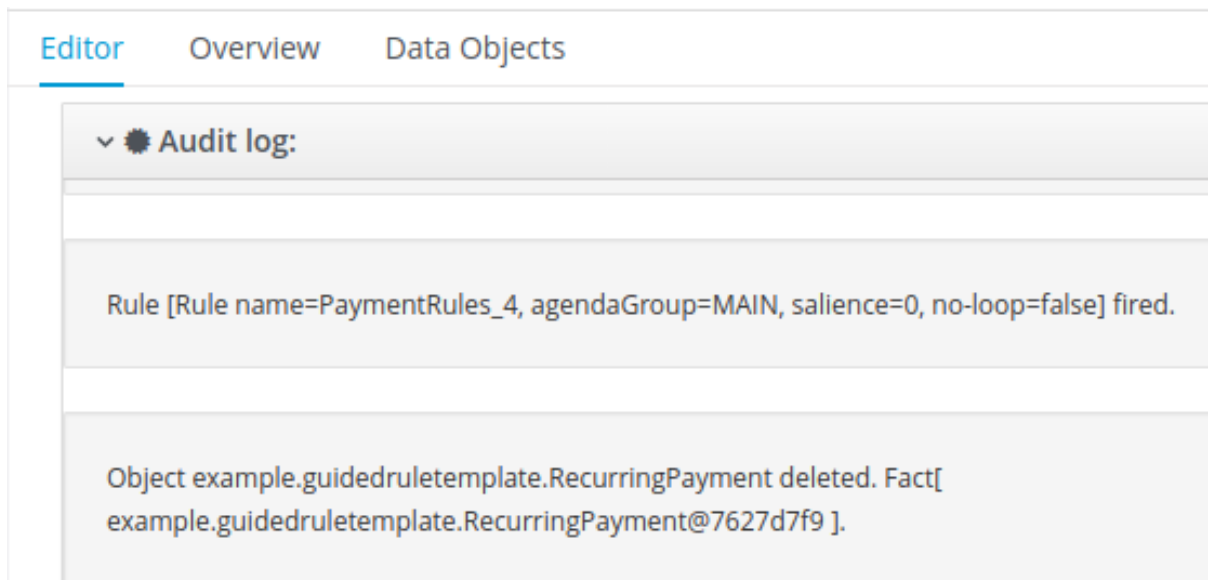
- Click **Save** in the upper right corner to save your scenario. Ensure you regularly save and review your scenarios.
- Click **Run scenario** in the upper right corner to execute your test. The results are displayed at the bottom of this screen in a new panel called **Reporting**.

Reporting
🔍 ✕

Success

1 test(s) ran in 0 minutes 0 seconds.

- If you created more tests in one file, you can run all the tests in a sequence. Click **Run all scenarios** to do so.
- Also note the **Audit log**, which informs you about inserted facts and fired rules:



9.3. ADDITIONAL TEST SCENARIO FEATURES

In addition to the previous Test Scenario features, Test Scenarios include various other features.

Calling Methods


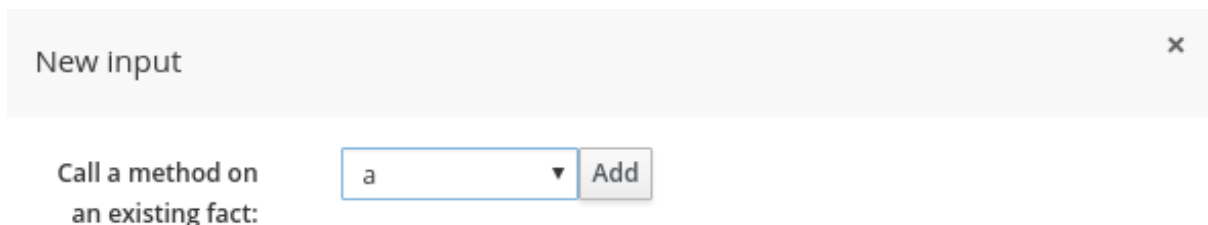
1. **Call Method** enables you to call a method on an existing fact in the beginning of the rule execution. This feature is accessed by clicking  next to the **CALL METHOD** label.

Figure 9.2. Call Method



2. After selecting an existing fact from the drop-down list, click **Add**. The green arrow button  enables you to call a method on the fact.

Figure 9.3. Invoke a Method



Using Globals in a Test Scenario

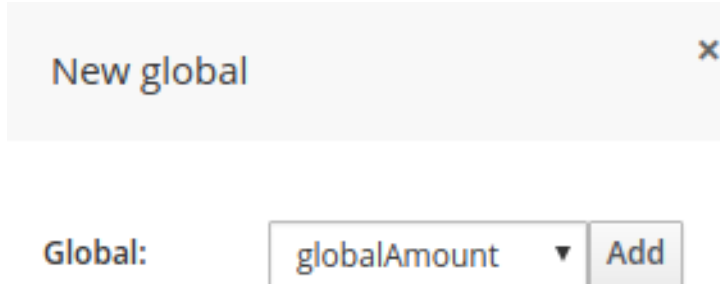
Globals are named objects that are visible to the rule engine but are different from the objects for facts. Accordingly, the changes in the object of a global do not trigger the reevaluation of rules. You can use and validate global fields in a Test Scenario.

To make a global variable accessible for your test scenario:

1. Click **New Item** → **Global Variable(s)** to create a global definition.
2. Define your global variable name and type.
3. Import the object type in your test. If you do not import the type of your global variable, the variable will not be accessible for your test.

Adding a New Global

1. Click **+** next to the **(globals)** label to add a global and click **Add**.

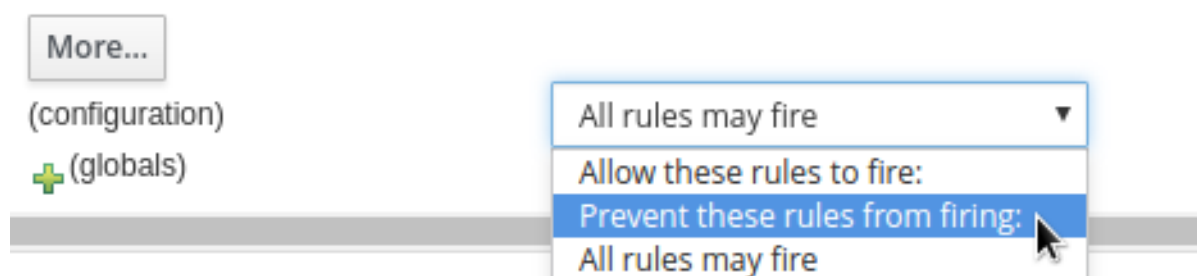


Adding restrictions on fields is similar to adding fields and restrictions in the **Given** section. See [Providing Given Facts](#) for further information.

Configuring Rules

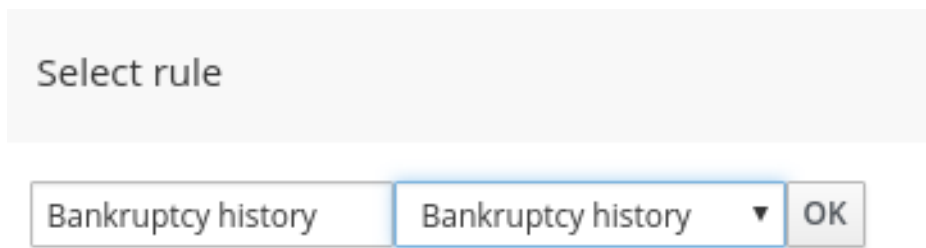
1. The **(configuration)** label enables the you to set additional constraints on the firing of rules by providing the following options:
 - **Allow these rules to fire:** enables you to select which rules are allowed to fire.
 - **Prevent these rules from firing:** enables you to prevent certain rules from firing for the test scenario.
 - **All rules may fire** allows all the rules to fire for the given test.

Figure 9.4. Configuration



2. If you select one of the following:
 - **Allow these rules to fire:**
 - **Prevent these rules from firing:**
Enter the rules into the empty field. Clicking **+** next to the empty field to select which rules are affected by the condition.

Figure 9.5. Selecting rules

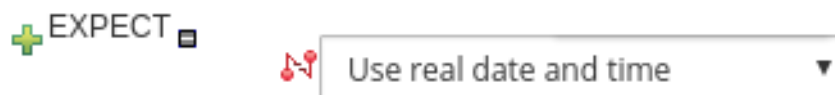


3. Choose a rule from the drop-down list and click **OK**. The selected rules will appear in the field next to the rules configuration option.

Date and Time Configuration

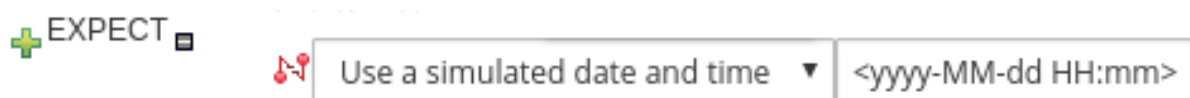
1. The **Use real date and time** option uses real time when running the test scenario.

Figure 9.6. Real Date and Time



2. The **Use a simulated date and time** option enables you to specify the year, month, day, hour, and minute associated with the test scenario.

Figure 9.7. Title



Advanced Fact Data


1. After providing fields to editable properties as part of your created fact, click  to open the **Field value** dialogue. You can edit literal values or provide advanced fact data.

Figure 9.8. Advanced Options

Field value ×

Literal value: Literal value ?

Advanced Options...

A variable: Bound variable ?

Fact: Create new fact ?

2. In the **Advanced Options...** section, you can choose between the following, depending on the type of fact created and the model objects used for the particular test scenario.
 - **Bound variable** sets the value of the field to the fact bound to the selected variable. The field type must match the bound variable type.
 - **Create new fact** enables you to create a new fact and assign it as a field value of the parent fact. Click on the fact to be assigned as a field value to be supplied with a drop down of various field values. These values may be given further field values.

Adding More Sections

- The **Editor** tab enables you to add **GIVEN**, **CCALL METHOD**, and **EXPECT** sections to the scenario. Click **More** below the **EXPECT** section to do so. This will open a block with all three sections that can be removed by clicking .

Modifying or Deleting an Existing Fact

When you create more tests in one file, it is recommended to delete facts inserted by previous tests. When you insert a new **GIVEN** fact, notice the following fields:

- **Modify an existing fact** enables you to edit a fact between knowledge base executions.
- **Delete an existing fact** enables you to remove facts between executions.

Figure 9.9. Modifying and Deleting Existing Facts

Modify an existing fact: Cust ▼ Add

Delete an existing fact: Cust ▼ Add

PART III. 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*.

CHAPTER 10. 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.

CHAPTER 11. 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.

CHAPTER 12. 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:

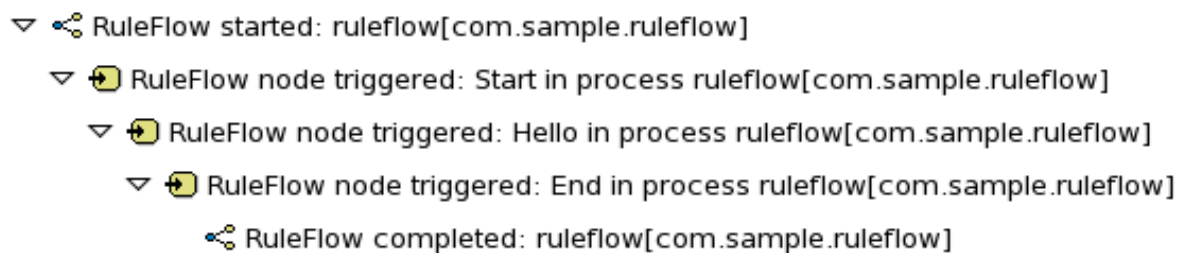
```
KieRuntimeLogger logger =
KieServices.Factory.get().getLoggers().newThreadedFileLogger(ksession, "mylogfile", 1000);
// Do something with the ksession here.
logger.close();
```

2. Do not forget to close the logger when you finish using it.

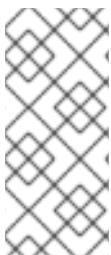
Procedure: Using Audit View

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 12.1. 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 threshold 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

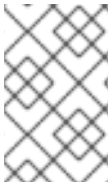
CHAPTER 13. DEPLOYING AND MANAGING PROJECTS

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.

13.1. 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.

13.1.1. 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 13.1. List of Repositories to Be Checked

Repositories: Validation ▾



These Maven Repositories are used to check the uniqueness of a Project's GAV when (1) creating a new Project or Module, (2) Installing or Deploying a Project to a Maven Repository.

They are obtained from the Project's pom, the Project's Distribution Management configuration and Maven's global settings.

Include	Id	URL	Source
<input checked="" type="checkbox"/>	local	/home/kkufova/.m2/repository	Local
<input checked="" type="checkbox"/>	central	https://repo.maven.apache.org/maven2	Project
<input checked="" type="checkbox"/>	guvnor-m2-repo	/maven2/	Project
<input checked="" type="checkbox"/>	jboss-ga-plugin-repository	http://maven.repository.redhat.com/techpreview/all	Maven settings
<input checked="" type="checkbox"/>	jboss-ga-repository	http://maven.repository.redhat.com/techpreview/all	Maven settings

Figure 13.2. Duplicate GAV Detected

Conflicting Repositories
✕



The following Repositories already contain Artifact "org.jbpm:human-resources:1.0".

Id	URL	Source
local	/home/kkufova/.m2/repository	Local

+ Ok
Override

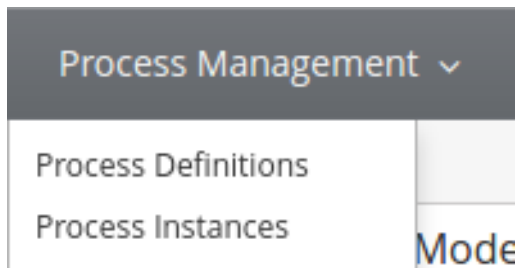
**NOTE**

To disable this feature, set the **org.guvnor.project.gav.check.disabled** system property to **true**.

13.2. PROCESS MANAGEMENT

The following sections describe the features provided by the options available under the Process Management menu in Business Central.

Figure 13.3. Process Management



13.2.1. 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.

13.2.2. 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.

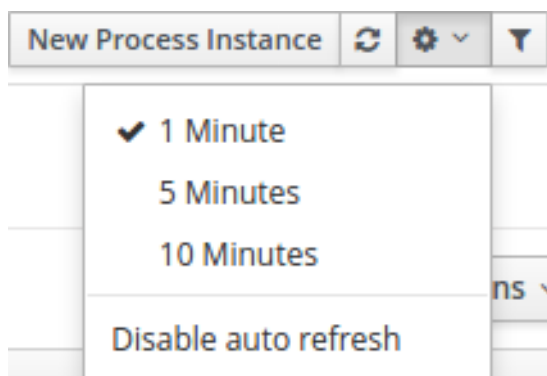
You can view the list of all the running process instances in the Process Instance List under **Process Management** → **Process Instances**. The process instances view comprises two main sections:

- 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 13.4. 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 activities in grey and the current activities highlighted in red.

13.2.2.1. 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.

13.2.2.2. 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.

13.2.3. 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 13.5. 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.

13.2.4. 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.

13.3. SIGNALING PROCESS INSTANCE

You can signal a running process instance using either API or Business Central. For further information about signals, signalling external deployment, catching and processing signals, and more, see [Chapter 22, Collaboration mechanisms](#).

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:

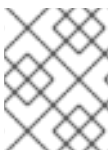
1. Log into Business Central.
2. Click **Process Management** → **Process Instances**.
3. Locate the required process instance and click **Signal** in the instance row.
4. Fill the following fields:
 - **Signal Name:** corresponds to the **SignalRef** or **MessageRef** attributes of the signal. This field is required.



NOTE

You can also send a **Message** event to the process. To do so, add the **Message-** prefix in front of the **MessageRef** value.

- **Signal Data:** corresponds to data accompanying the signal. This field is optional.



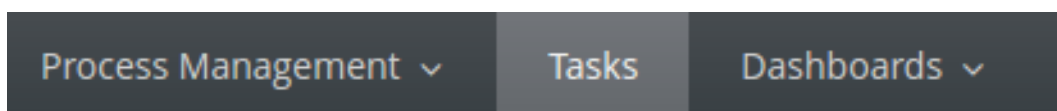
NOTE

When using the Business Central user interface, you may signal only Signal intermediate catch events.

13.4. TASK MANAGEMENT

The following sections describe the features provided by the options available under the Tasks menu in Business Central.

Figure 13.6. Task Management



13.4.1. Tasks List

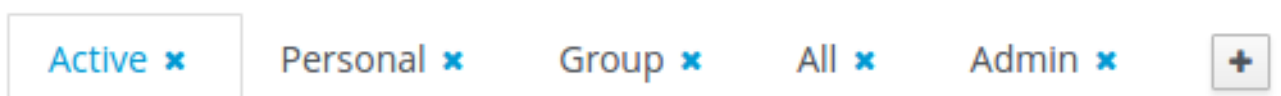
A User Task represents a piece of work the given user can claim and perform. User Tasks can be handled within the **Tasks** perspective of the Business Central: the view displays the Task List for the given user. You can think about it as a to-do item. The User Task appears in your list either because 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 13.7. 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 13.4.2, "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.
- **Assignments:** 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).

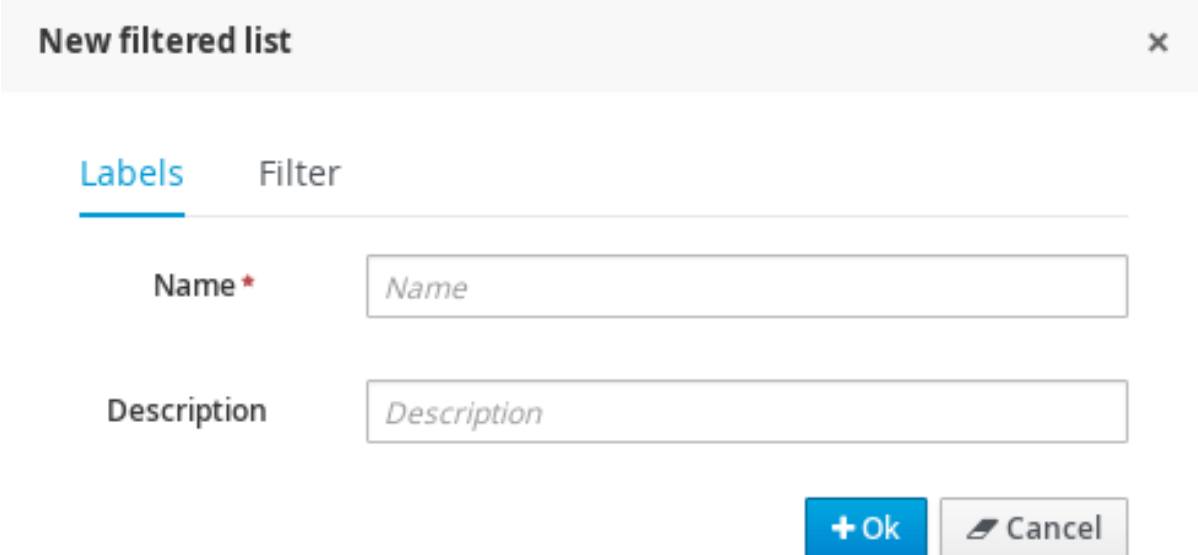
13.4.2. Creating Custom Tasks Filters

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.

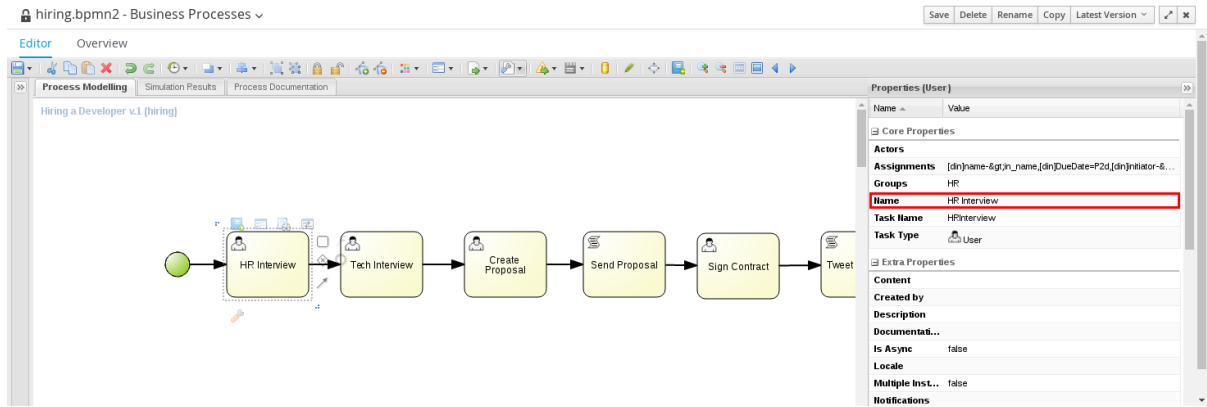
The following procedure shows how to create a custom filter which allows you to view a list of tasks with a specified name.

Procedure: Filtering Tasks by Name

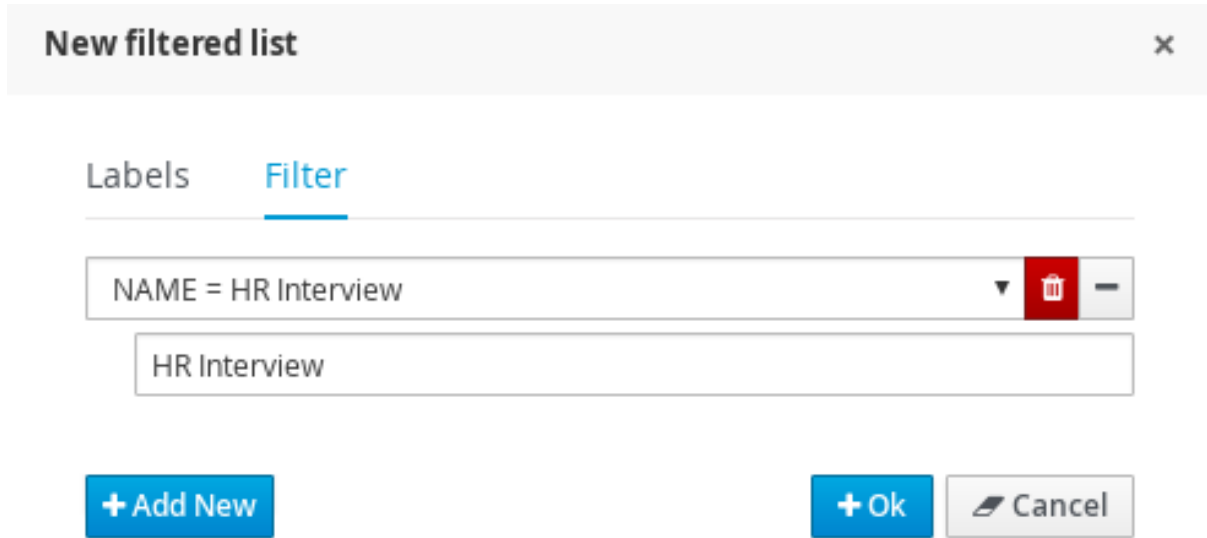
1. In the main menu of Business Central, click **Tasks**.
2. Click the  button on the right side of the Tasks Lists tabs. The **New filtered list** pop-up window is displayed.



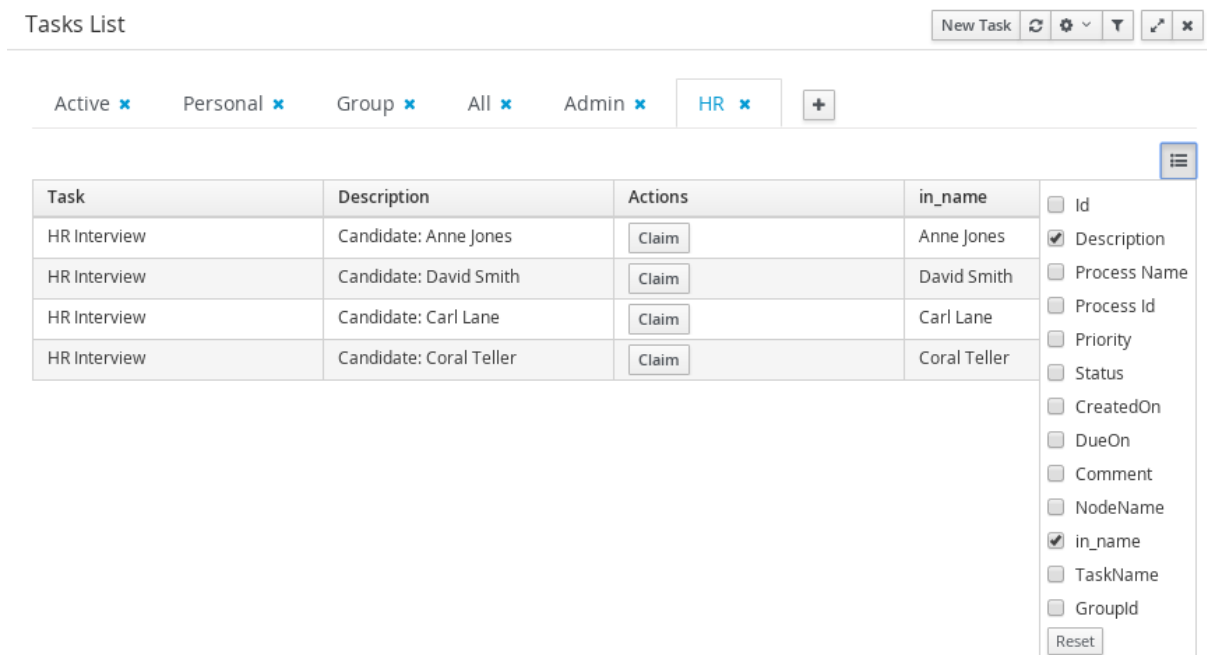
3. Fill in the **Name** (this is the label of the new Tasks Lists tab) and click **Filter**.
4. Click **Add New**.
5. In the **Select column** drop-down menu, choose **NAME**.
The content of the drop-down menu changes to **NAME != value1**.
6. Click on the drop-down menu again and choose **equals to**.
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.



13.4.3. 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 13.8. 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.

13.4.4. 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 14. 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 15. 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 16. 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).

16.1. 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.

16.2. 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 16.3, “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.

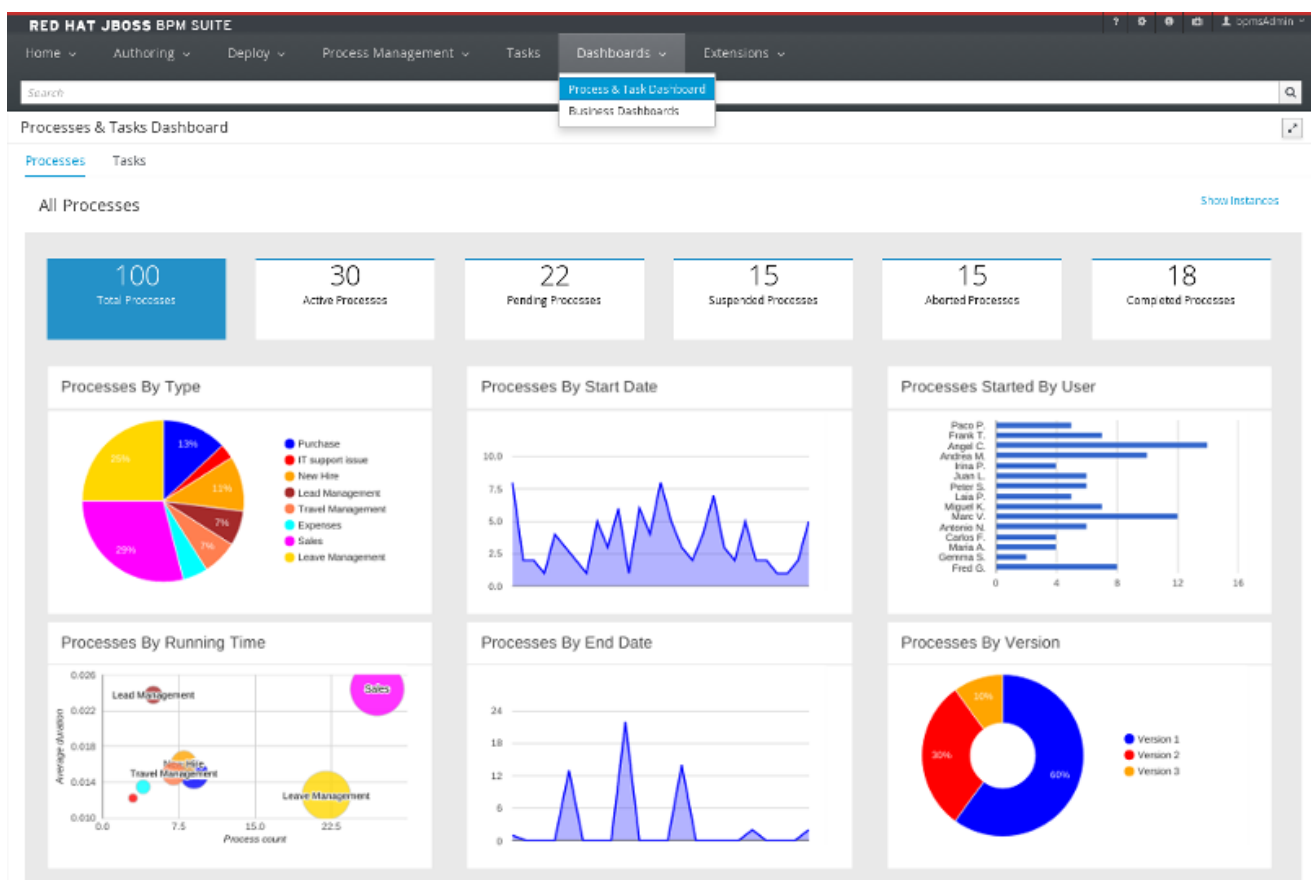
At the top of the page, you can change the workspace, the page, as well as find general configuration buttons. This interface area is common for all workspaces.

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.

16.3. 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 16.1. 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.



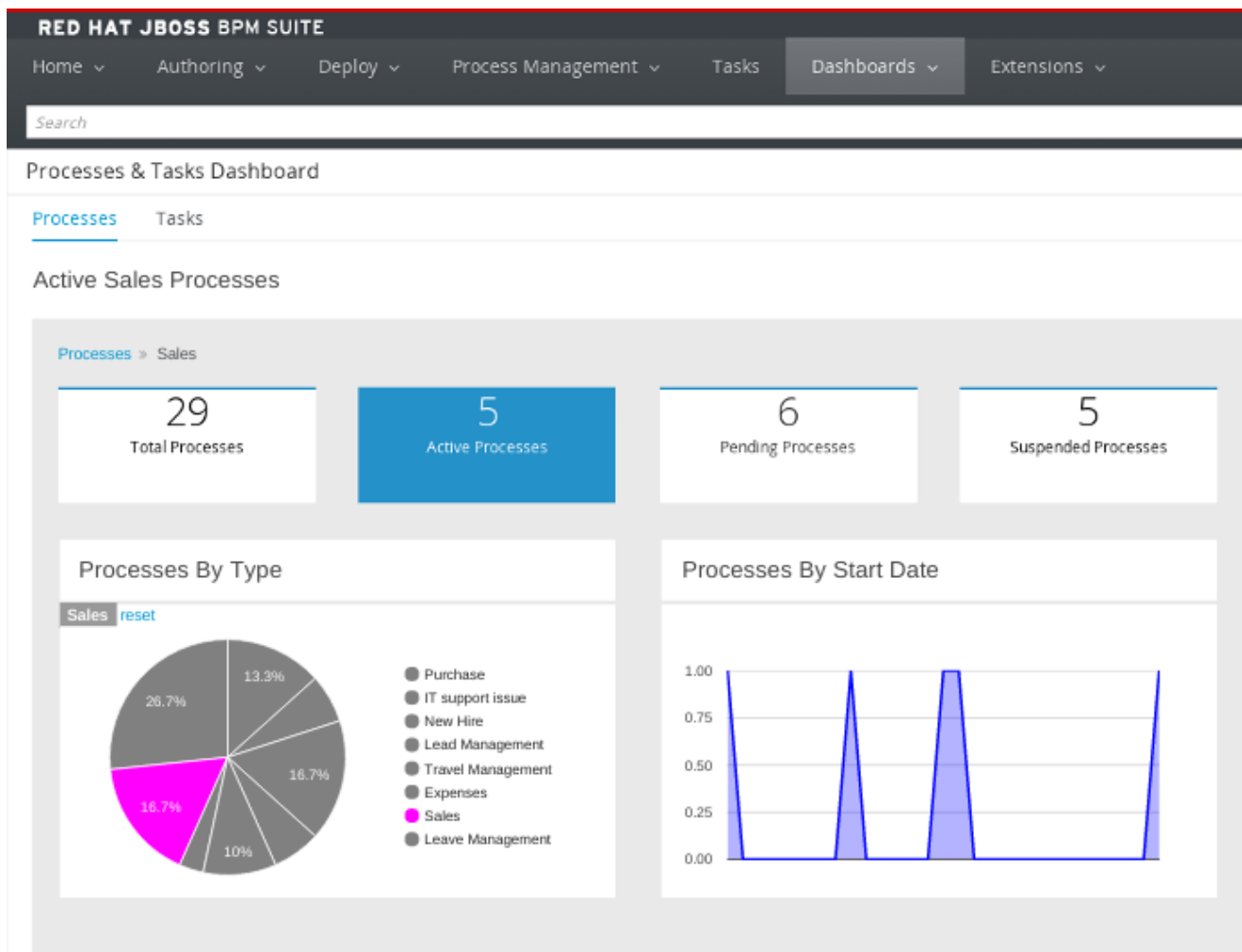
NOTE

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 16.2. 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 16.3. Process Instances List

The screenshot shows the 'Processes & Tasks Dashboard' for 'All Processes'. The navigation bar includes 'Home', 'Authoring', 'Deploy', 'Process Management', 'Tasks', 'Dashboards', and 'Extensions'. A search bar is located below the navigation bar. The main content area is titled 'All Processes' and includes a 'Show Dashboard' link. Below this is a table listing process instances.

Id	Deployment id	Process id	Process name	Initiator	Status	Version	Start	End	Duration
1	org.jboss.Evaluato...	evaluation	Evaluation	saliboy	Aborted	1	Oct 15, 2015 11:44	Oct 15, 2015 11:59	15m 31s
2	org.jboss.Customer...	CustomerRelatio...	Customer Relation...	saliboy	Active	1.0	Oct 15, 2015 11:55	---	17m 48s
3	org.jboss.Customer...	CustomerRelatio...	Customer Relation...	saliboy	Active	1.0	Oct 15, 2015 11:55	---	17m 36s
4	org.jboss.Customer...	CustomerRelatio...	Customer Relation...	saliboy	Active	1.0	Oct 15, 2015 11:55	---	17m 28s
5	org.jboss.Evaluato...	evaluation	Evaluation	saliboy	Active	1	Oct 15, 2015 11:55	---	17m 15s
6	org.jboss.Evaluato...	evaluation	Evaluation	saliboy	Active	1	Oct 15, 2015 11:56	---	17m 4s
7	org.jboss.Evaluato...	evaluation	Evaluation	saliboy	Active	1	Oct 15, 2015 11:56	---	16m 49s
8	org.jboss.Evaluato...	evaluation	Evaluation	saliboy	Active	1	Oct 15, 2015 11:56	---	16m 33s
9	org.jboss.HR.1.0	hiring	Hiring a Developer	saliboy	Active	1	Oct 15, 2015 11:56	---	16m 25s
10	org.jboss.HR.1.0	hiring	Hiring a Developer	saliboy	Active	1	Oct 15, 2015 11:56	---	16m 14s

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 16.4. Process Instance Details Panel

The screenshot displays the Red Hat JBoss BPM Suite interface. The top navigation bar includes 'Home', 'Authoring', 'Deploy', 'Process Management', 'Tasks', 'Dashboards', and 'Extensions'. The main content area is divided into two panels. The left panel, titled 'Processes & Tasks Dashboard', shows a table of 'Active Customer Relationships Processes'. The right panel, titled '4 - Customer Relationships', displays the 'Instance Details' for a selected process instance.

Id	Deployment Id	Process Id	Process name	Initiator	Status
4	org.jboss.Customer...	CustomersRelation...	Customer Relation...	salaboy	Active
3	org.jboss.Customer...	CustomersRelation...	Customer Relation...	salaboy	Active
2	org.jboss.Customer...	CustomersRelation...	Customer Relation...	salaboy	Active

The 'Instance Details' panel on the right shows the following information:

- Definition Id:** CustomersRelationship.customers
- Instance State:** Active
- Deployment:** org.jboss.CustomerRelationship:1.0
- Definition Version:** 1.0
- Correlation key:**
- Parent Process Instance:** No Parent Process Instance
- Active user tasks:** Obtain Customer Info (Ready) owner: ---
- Current Activities:** 15/Oct/15 11:55:38: 1 - Obtain Customer Info (-HumanTaskNode)

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.

16.4. 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.

16.4.1. 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**  **Create 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.

16.4.2. 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.

16.4.3. 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.


16.4.4. 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.

16.4.4.1. 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**  **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.

16.4.5. 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.

16.4.5.1. 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.

16.4.5.2. Configuring a default workspace

You can configure a default workspace in Red Hat JBoss BPM Suite Dashboard. For details, see [How to configure default workspace in BPM Suite dashbuilder](#).

16.4.5.3. 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.

16.4.5.3.1. 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

16.4.5.3.2. 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 assignment** 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**.

16.4.5.4. 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

16.4.5.4.1. 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.

16.5. 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.

16.5.1. 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

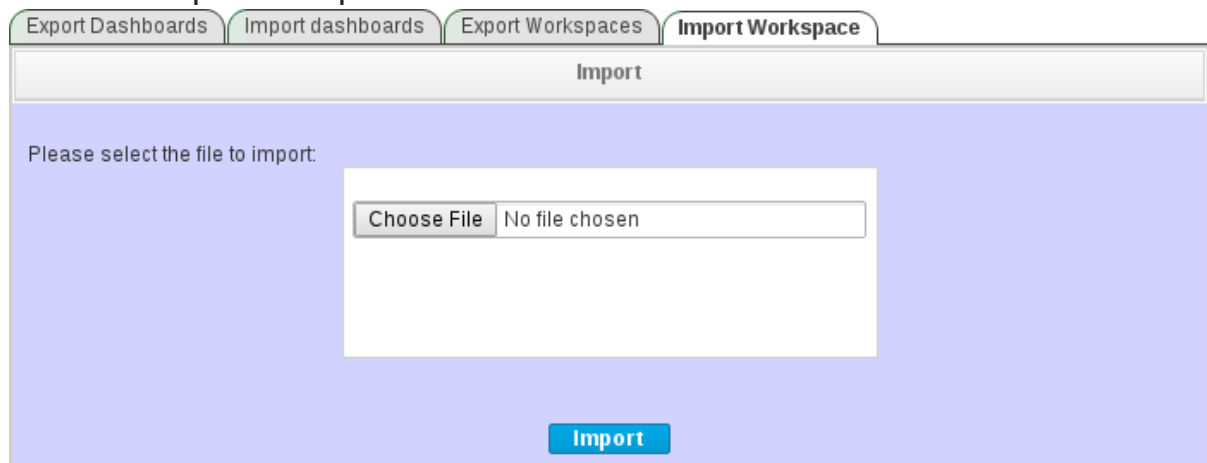
1. In Business Central, go to **Dashboards** → **Business Dashboards**.
2. In the menu on the left, click **Administration** → **Import and export**
3. Choose the **Export Workspaces** tab.
4. In the list of all existing workspaces that opens, select the ones you want to export and click **Export**.



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.

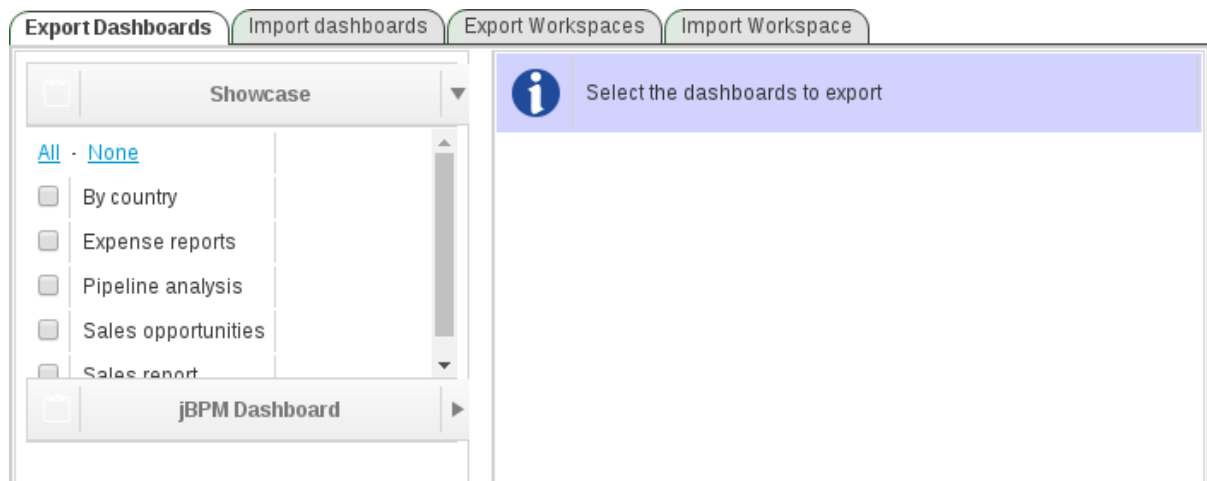
The workspaces are imported once during the application deployment.

16.5.2. Importing and Exporting KPIs

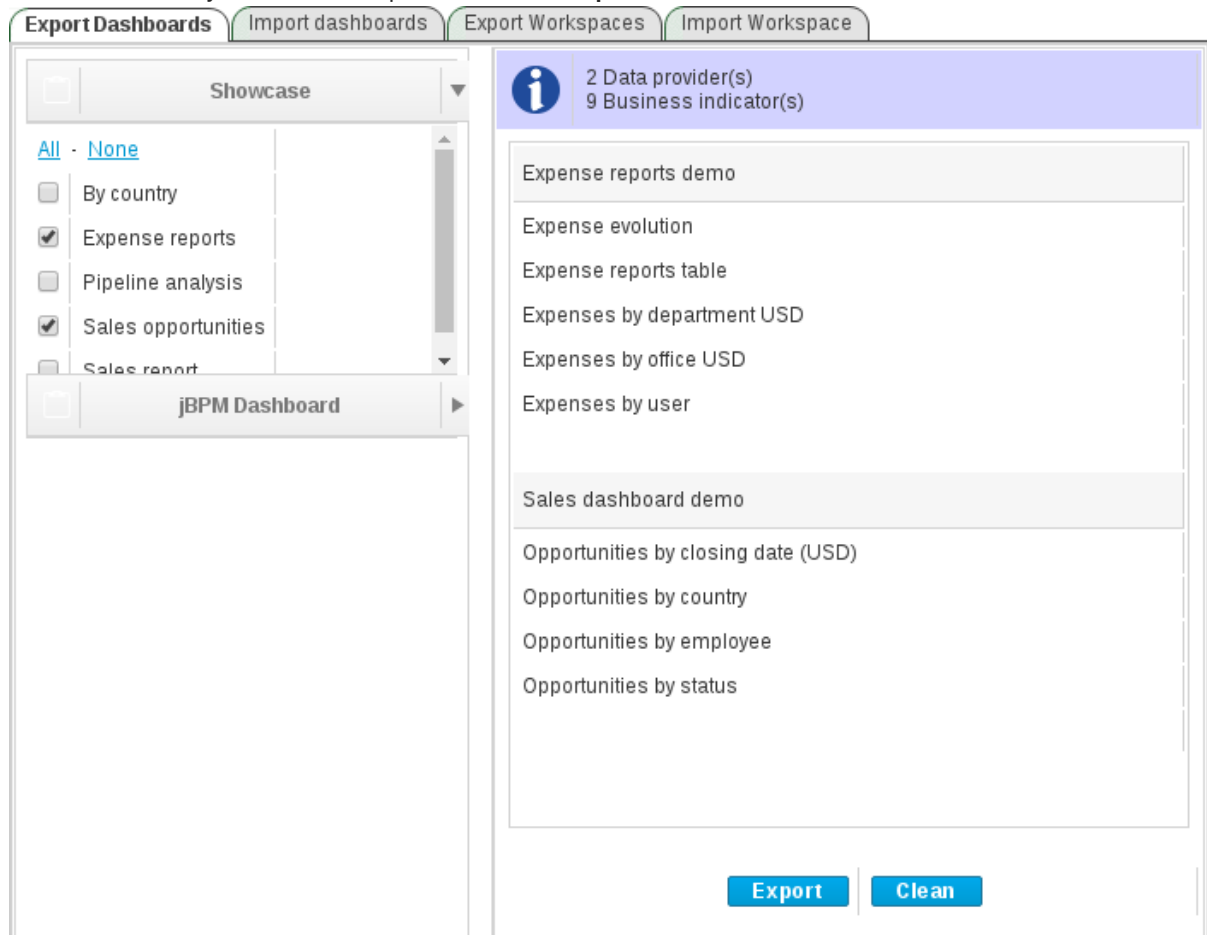
By importing and exporting KPIs, you can move key performance indicator definitions (the KPI type, its columns and display configuration) and their data providers between two Dashboard Builder installations.

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.

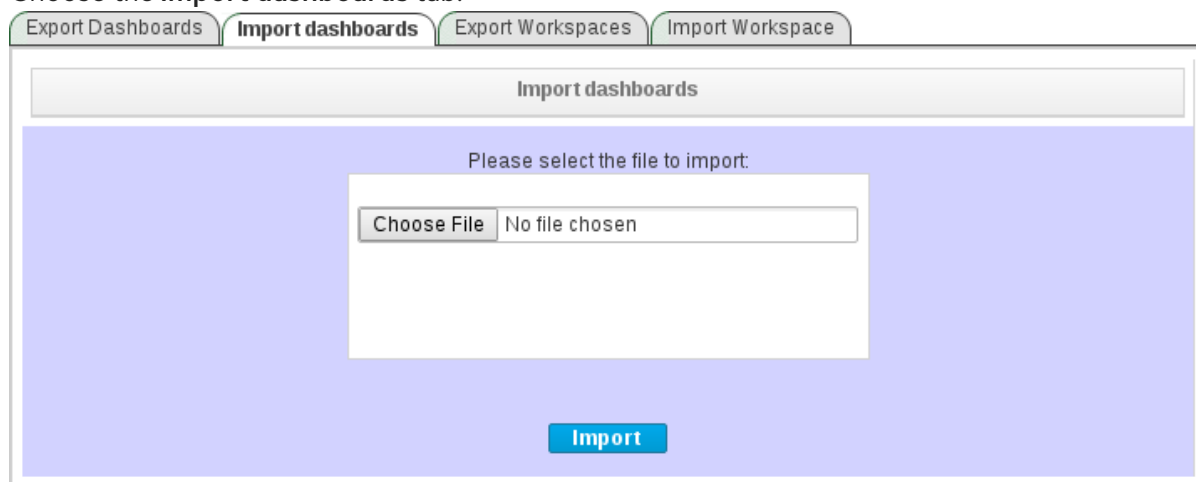


4. Select the KPIs you want to export and click **Export**.



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.

16.5.3. 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.

External Connections

[Administration](#) > [External connections](#)

Creation of new DataSource

Type	<input checked="" type="radio"/> JNDI <input type="radio"/> Custom DataSource
Name	<input type="text" value="Test JNDI Data Source"/>
JNDI path	<input type="text" value="java:jboss/datasources/ExampleDS"/>
Test Query	<div style="border: 1px solid #ccc; min-height: 100px; padding: 5px;">SELECT 1</div>

Check DataSource
Save
Cancel

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**.

External Connections

[Administration](#) > [External connections](#)

[+ Create new DataSource](#)

Actions	Name	Type	Path	State
	Local	System		
	Test JNDI Data Source	JNDI	java:jboss/datasources/ExampleDS	

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 16.1. 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 16.2. JDBC 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

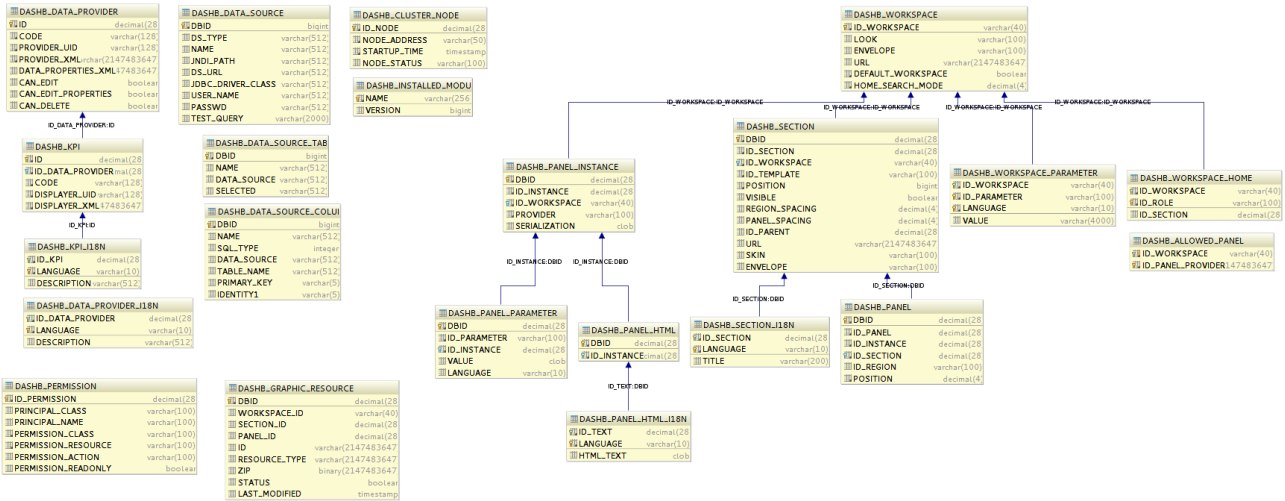
```

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.

16.6. 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 16.1. 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).

Table	Attributes	Description
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.
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 <i>not</i> 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.

Table	Attributes	Description
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).
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.

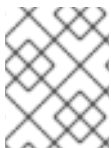
CHAPTER 17. DATA SETS

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.

17.1. 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.
 - Elasticsearch – generate a data set from Elasticsearch nodes.



NOTE

Elasticsearch data set integration support is limited to commercially reasonable efforts. For details, see [What is commercially reasonable support?](#)

1. Complete the **Data Set Creation Wizard** and click **Test**.
2. 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 17.2, “Caching”](#) for more information.
- Cache life-cycle – see [Section 17.3, “Data Refresh”](#) for more information.

17.2. 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 backend.

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**.

17.3. 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.

CHAPTER 18. 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**  **Workspace** in the upper left corner.

CHAPTER 19. 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.

19.1. 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



DISCLAIMER

This chapter contains introduction to BPMN elements and their semantics. For details about BPMN, see the Business Process Model and Notation, Version 2.0. The BPMN 2.0 specification is an Object Management Group (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.

Note that Red Hat JBoss BPM Suite focuses exclusively 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 nodes with execution semantics as well as those without.

Elements with execution semantics define the execution workflow of the process.

Elements without execution semantics, such as artifacts, allow users to provide notes and further information on the process or any of its elements to accommodate collaboration of multiple users with different roles, such as, business analyst, business manager, or process designer.

All elements with execution semantics define their generic properties.

Generic Process Element Properties

ID

The ID defined as a String, unique in the parent knowledge base.

Name

The display name of the element.

CHAPTER 20. 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 parts 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 the visualisation of the process execution workflow, 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, similarly to a class and its objects. A process instance is managed by a session, which may contain multiple process instances. This enables the instances to share data, for example, using global variables. Global variables are stored in the session instance, not in the process instance, which enables communication across process instances. Every process instance has its own context and ID.

Knowledge Runtime, called **kcontext**, holds all the process runtime data. Use it to retrieve or modify the runtime data, for example in Action Scripts:

- Getting the currently executed element instance. You can then query further element data, such as its name and type, or cancel the element instance.

```
NodeInstance element = kcontext.getNodeInstance();
String name = element.getNodeName();
```

- Getting the currently executed process instance. You can then query further process instance data, such as its name, ID. You can also abort the process instance, or send an event, such as a Signal Event.

```
ProcessInstance proc = kcontext.getProcessInstance();
proc.signalEvent(type, eventObject);
```

- Getting and setting the values of variables.

```
kcontext.setVariable("myVariableName", "myVariableValue");
```

- Execute calls on the Knowledge runtime, for example, start process instances, insert facts, and similar.

```
kcontext.getKnowledgeRuntime().signalEvent(eventType, data,
kcontext.getProcessInstance().getId());
```

A process instance has the following lifecycle phases:

CREATED

When you call the **createProcessInstance** method on a process, a new process instance is created. The process variables are initialized and the status of the process instance is **CREATED**.

PENDING

When a process instance is created, but not yet started.

ACTIVE

When you call the **start()** method on a process instance, its execution is triggered and its status is **ACTIVE**. If the process is instantiated using an event, such as Signal, Message, or Error Events, the flow will start on the respective type of start event. Otherwise, the flow starts on the None Start Event.

COMPLETED

Once there is no token in the flow the process instance is finished and its status is **COMPLETED**. Tokens in the flow are consumed by End Events and destroyed by Terminating Events.

ABORTED

If you call the **abortProcessInstance** method, the process instance is interrupted and its status is **ABORTED**.

The runtime state of a process instance can be made persistent, for example, in a database. This enables you 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 chapter [Persistence](#) of 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>**.

Process 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

The location of the XML schema definition of the BPMN2 standard.

Executable

Enables or disables the process to be instantiated. Set to **false** to disable process instantiation.
Possible values: **true, false**.

Imports

Comma-separated values of imported processes.

Documentation

Contains element description, 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.

Globals

Set of global variables visible for other processes to allow data sharing.

Variable Definitions

Enables you to define variables available for the process.

Process Instance Description

Contains description of the process, has no impact on runtime.

TypeLanguage

Identifies a type system used for the process.

Base Currency

Identifies the currency in simulation scenarios. Uses the ISO 4217 standard, for example **EUR**, **GBP**, or **USD**.

CHAPTER 21. EVENTS MECHANISM

During process execution, the Process Engine ensures 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 nevent it was not directly requesting. Such events can be created and caught by the Intermediate Event elements. See [Chapter 39, *Throwing Intermediate Events*](#) for further information. Using these events in a process enables you to specify how to handle a particular event.

An event must specify the type of event it should handle. It can also define the name of a variable that will store the data associated with the event. This enables subsequent elements in the process to access and react to the 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 action a node, can signal the occurrence of an internal event to the process instance.

```
kcontext.getProcessInstance().signalEvent(type, eventData);
```

- **External event**
A process instance can be notified of an event from the outside.

```
processInstance.signalEvent(type, eventData);
```

- **External event using event correlation**
You can notify the entire session and use the event correlation to notify particular processes. Event correlation is determined based on the event type. A process instance that contains an event element listening to external events is notified whenever such an event occurs. To signal such an event to the process engine:

```
ksession.signalEvent(type, eventData);
```

You can also use events to start a process. When a Message Start Event defines an event trigger, a new process instance starts every time the 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.

CHAPTER 22. COLLABORATION MECHANISMS

Elements with execution semantics use collaboration mechanisms. Different elements use the collaboration mechanism differently. For example, if you use signalling, the Throw Signal Intermediate Event element sends a signal, and the Catch Signal Intermediate Event element receives the signal. That means Red Hat JBoss BPM Suite provides you with two elements with execution semantics that make use of the same signal mechanism in a collaborative way.

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 KIE session instance.

For external system interaction, use Send and Receive Tasks with proper handler implementation.

Escalations

Used as signalling between processes to trigger escalation handling.

Errors

Used as 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.

22.1. SIGNALS

Signals in Red Hat JBoss BPM Suite correspond to the Signal Event in the specification 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.

22.1.1. Triggering Signals

The following Throw Events trigger signals:

- Intermediate Throw Event
- End Throw Event

Every signal defines its signal reference, that is the **SignalRef** property, which is unique in the respective session.

A signal can have one of the following scopes, which restricts its propagation to the selected elements:

Default (ksession)

Signal only propagates to elements within the given KIE session. The behavior varies depending on what runtime strategy is used:

- **Singleton**: All instances available for the KIE session 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 22.1. Selecting Signal Scope (Default)

The screenshot shows a process flow with a 'Throwing default signal' event. To the right, the 'Properties (Signal)' panel is open, displaying a table of properties. A blue arrow points to the 'Signal Scope' property, which is set to 'Default'.

Name	Value
Core Properties	
DataInputAs...	
Name	
Signal Scope	Default
SignalRef	mysignal
Extra Properties	
Documentati...	
Graphical Settings	
Background ...	■
Border Color	■
Font Size	
Font color	■
Simulation Properties	
Distribution ...	uniform
Processing t...	10
Processing t...	5

Signalling External Deployments

When creating an external signal event, you need to specify the work item handler for the External Send Task manually. Use the **org.jbpm.process.workitem.jms.JMSSendTaskWorkItemHandler** work item handler, 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 22.2. Registered External Send Task Handler

Work Item handlers

Value	Value	Resolver type	Parameters	Remove
Log	<code>new org.jbpm.process.instance.impl.demo.SystemOutWorkItemHandler()</code>	mvel	Parameters(0)	Remove
Service Task	<code>new org.jbpm.process.workitem.bpmn2.ServiceTaskHandler(ksession, classLoader)</code>	mvel	Parameters(0)	Remove
WebService	<code>new org.jbpm.process.workitem.webservice.WebServiceWorkItemHandler(ksession, classLoader)</code>	mvel	Parameters(0)	Remove
Rest	<code>new org.jbpm.process.workitem.rest.RESTWorkItemHandler(classLoader)</code>	mvel	Parameters(0)	Remove
External Send Task	<code>new org.jbpm.process.workitem.jms.JMSSendTaskWorkItemHandler()</code>	mvel	Parameters(0)	Remove

+ Add

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_FACTOR_Y_NAME", "DESTINATION_NAME")`
- **new**
`org.jbpm.process.workitem.jms.JMSSendTaskWorkItemHandler("CONNECTION_FACTOR_Y_NAME", "DESTINATION_NAME", TRANSACTED)`, where **TRANSACTED** is **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, specify the following data inputs in the **DataInputAssociations** property of the signal event in the Process Designer. See [Section 22.1.2, "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. The parameter is optional.
- **SignalDeploymentId:** The target deployment ID.

Figure 22.3. Specifying SignalDeploymentId Data Input

Data I/O
×

+ Add

Name	Data Type	Source	
<input style="width: 100%;" type="text" value="SignalDeploymentId"/>	<input style="width: 100%;" type="text" value="String"/>	<input style="width: 100%;" type="text" value="Enter constant ..."/>	

Cancel
Save

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, that is to limit a number of sessions 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.

22.1.2. Catching and Processing Signals

Signals are caught by the following catch event types:

- 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 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, enter 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"](#).

22.1.3. 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**.

processInstanceld

The process ID of the signalled process.

You can use a more general version of the above function, which does not specify the parameter **processInstanceld**. 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.

22.2. 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.

22.2.1. Sending Messages

Like signals, messages are sent by throw events of one of the following types:

- 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.

22.2.2. Catching Messages

The process for catching messages does not differ from receiving signals, with the exception of using the **MessageRef** element property instead of **SignalRef**. See [Section 22.1.2, “Catching and Processing Signals”](#) for further information.

**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 22.2.3, “Sending Messages Using API”](#) for further information.

22.2.3. Sending Messages Using API

To send a message using the API, use the following method:

```
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 Task or using on-entry or on-exit actions of a node, use the following method:

```
kcontext.getKieRuntime().signalEvent(
    eventType, data, kcontext.getProcessInstance().getId());
```

The usage of the arguments **eventType** and **data** is the same as above.

22.3. 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**.

Attributes

Mandatory Attributes

Escalation Code

string with the escalation code

[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>

CHAPTER 23. TRANSACTION MECHANISMS

23.1. 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 event. 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 a 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.

23.2. COMPENSATION

Compensation is a mechanism that allows you to handle business exceptions that might occur in a process or sub-process, that is in a business transaction. Its purpose 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

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 cannot 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 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 to which the Compensation Intermediate Event points may 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.

CHAPTER 24. TIMING

Timing is a mechanism for scheduling actions and is used by Timer Intermediate and Timer Start events. It enables you to delay further execution of a process or task.



NOTE

A timer event can be triggered only after the transaction is committed, while the timer countdown starts right after entering the node, that is 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 24.1. 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 once after 2 seconds
```

Repetable Intervals

```
R/PT1S - fires every second, no limit, alternatively R5/PT1S will fire 5 times every second
```

None

```
#{expression}
```

Example 24.2. Timer period with expression

`myVariable.getValue()`

The element will be executed after time period returned by the call `myVariable.getValue()`.

CHAPTER 25. EVENT TYPES

Events are triggers that impact a business process. Events are classified as:

- Start events
Indicate the beginning of a business process.
- End events
Indicate the completion of a business process.
- 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 the process flow does not complete.

25.1. 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.

25.1.1. Start Event types

25.1.1.1. None Start Event

The None Start Event is a start event without a trigger condition. A process or a 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. That means that the token is taken from the parent sub-process activity and the None Start Event of the sub-process generates a token.

25.1.1.2. Message Start Event

A process or an event sub-process can contain multiple Message Start Events, which are 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. After the message is received, 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

MessageRef

ID of the expected Message object

25.1.1.3. Timer Start Event

The Timer Start Event is a Start Event with a timing mechanism. For more information about timing, see [Chapter 24, *Timing*](#).

A process can contain multiple Timer Start Events, which are triggered at the start of the process, after which the timing mechanism 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 timer to trigger. Once the time defined by the timing definition has been reached, the outgoing flow is taken.

Attributes

Time Cycle

Repeatedly triggers the timer after a specific time period. If the period is **0**, the timer is triggered only once.

Time Cycle Language

Set to **None** for the default interval, or **Cron** for the following **Time Cycle** property format:

```
[#d][\#h][\#m][\#s][#[ms]]
```

Time Duration

Marks the timer as a one-time expiration timer. It is the delay after which the timer fires. Possible values are a String interval, a process variable, or the ISO-8601 date format.

Time Date

Starts the process at the specified date and time in the ISO-8601 date format.

25.1.1.4. Escalation Start Event

The Escalation Start Event is a start event that is triggered by an escalation with a particular escalation code. For further information, see [Section 22.3, "Escalation"](#).

Process can contain multiple Escalation Start Events. The process instance with an Escalation Start Event starts its execution when it receives the defined escalation object. The process is instantiated and the Escalation Start Event is executed immediately, which means its outgoing flow is taken.

Attributes

Escalation Code

Expected escalation Code.

25.1.1.5. Conditional Start Event

The Conditional Start Event is a start event with a Boolean condition definition. The execution is triggered always when the condition is first evaluated to **false** and then to **true**. The process execution starts only if the condition is evaluated to **true** after the start event has been instantiated.

A process can contain multiple Conditional Start Events.

Attributes

Expression

A Boolean condition that starts the process execution when evaluated to **true**.

Language

A language of the **Expression** attribute.

25.1.1.6. Error Start Event

A process or sub-process can contain multiple Error Start Events, which are triggered when an Error object with a particular **ErrorRef** property is received. The error object can be produced by an Error End Event, and it signals an incorrect process ending. The process instance with the Error Start Event starts execution after it has received the respective error object. The Error Start Event is executed immediately upon receiving the error object, which means its outgoing Flow is taken.

Attributes

ErrorRef

A code of the expected error object.

25.1.1.7. Compensation Start Event

A Compensation Start Event is used to start a Compensation Event sub-process when using a sub-process as the target activity of a Compensation Intermediate Event.

25.1.1.8. Signal Start Event

The Signal Start Event is triggered by a signal with a particular signal code. For further information, see [Section 22.1, "Signals"](#).

A 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. Then, the Signal Start Event is executed, which means its outgoing flow is taken.

Attributes

SignalRef

The expected Signal Code.

25.2. INTERMEDIATE EVENTS

25.2.1. 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]"

An intermediate event handles a particular situation that occurs during process execution. The situation is a trigger for an intermediate event.

In a process, intermediate events can be placed as follows:

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 types:

- 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 the event trigger, 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, which is triggered by an escalation event.
- Throwing Escalation Intermediate Event, which produces an escalation event when executed.

Signal Intermediate Event

Has two subtypes:

- Catching Signal Intermediate Event, which is triggered by a signal.
- Throwing Signal Intermediate Event, which produces a signal when executed.

Message Intermediate Event

Has two subtypes:

- Catching Message Intermediate Event, which is triggered by a message object.
- Throwing Message Intermediate Event, which produces a message object when executed.

Compensation Intermediate Event

Has two subtypes:

- Catching Compensation Intermediate Event, which is triggered by a compensation object.

- Catching Compensation Intermediate Event, which is triggered by a compensation object.
- Throwing Compensation Intermediate Event, which produces a compensation object when executed.

25.2.2. Intermediate Event types

25.2.2.1. 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, that is the defined time, is checked and the outgoing flow is taken. For more information about timing, see [Chapter 24, 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 triggered 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

Time Cycle

Repeatedly triggers the timer after a specific time period. If the period is **0**, the timer is triggered only once.

Time Cycle Language

Set to **None** for the default interval, or **Cron** for the following **Time Cycle** property format:

```
█ [#d][\#h][\#m][\#s][#\ms]
```

Time Duration

Marks the timer as a one-time expiration timer. It is the delay after which the timer fires. Possible values are a String interval, a process variable, or the ISO-8601 date format.

Time Date

Triggers the timer at the specified date and time in the ISO-8601 date format.

25.2.2.2. 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 the **Expression** property. When placed in the process workflow, a Conditional Intermediate Event has one incoming flow, 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

Expression

A Boolean condition that triggers the execution when evaluated to **true**.

Language

A language of the **Expression** attribute.

25.2.2.3. 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.

25.2.2.4. 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.

Throwing Message Intermediate Event

When reached during execution, a Throwing Message Intermediate Event produces a message object and the execution continues to its outgoing Flow.

Attributes

MessageRef

ID of the produced Message object.

Catching Message Intermediate Event

When reached during execution, a Catching Message Intermediate Event awaits a message object defined in its properties. Once the message object is received, the event triggers execution of its outgoing flow.

Attributes

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.

25.2.2.5. 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 should 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.

Throwing Escalation Intermediate Event

When reached during 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.

Catching Escalation Intermediate Event

When reached during execution, a Catching Escalation Intermediate Event awaits an escalation object defined in its properties. When the object is received, the event triggers execution of its outgoing Flow.

Attributes

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.

25.2.2.6. 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 object with the respective **ErrorCode** property, the Error Intermediate Event catches the error object and execution continues to its outgoing flow.

25.2.2.6.1. Catching Error Intermediate Event

When reached during 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

The reference number of the expected error object.

25.2.2.7. Signal Intermediate Event

A Signal Intermediate Event enables you to produce or consume a signal object. 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.

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 will be sent.

Signal Scope

You can choose one of the following scopes:

- **Process Instance:** Catch events in the same process instance can catch this signal.
- **Default:** Catch events in a given KIE session can catch this signal. The behavior varies depending on the KIE session strategy:
 - **Singleton:** Signal reaches all the process instances available to the KIE session.
 - **Per request:** Signal reaches only the current process instance and start processes with a Signal Start Event.
 - **Per process:** same as **per request**.
- **Project:** Signal reaches only active process instances of a given deployment and starts processes with a Signal Start Event.
- **External:** Enables the signal to reach the same process instances as with the **Project** scope, as well as process instances across deployments. To send the signal to a process instance across deployments, create a **SignalDeploymentId** process variable that provides information about what deployment or project should be the target of the signal. Broadcasting the signal would have negative impact on performance in larger environments.

25.2.2.7.1. Catching Signal Intermediate Event

When reached during 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.

25.3. 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. The end event can finish only the workflow that reached it, or all workflows in the process instance, depending on the end event type.

25.3.1. End Event types

25.3.1.1. Simple End Event

The Simple End Event finishes the incoming workflow, that means it 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 that turns a Simple End Event into a Terminate End Event when set to **true**.

25.3.1.2. Message End Event

When a flow enters a Message End Event, the flow finishes and the end event produces a message as defined in its properties.

25.3.1.3. Escalation End Event

The Escalation End Event finishes the incoming workflow, that means consumes the incoming token, and produces an escalation signal as defined in its properties, triggering the escalation process.

25.3.1.4. 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.

25.3.1.5. Throwing Error End Event

The Throwing Error End Event finishes the incoming workflow, that means consumes the incoming token, and produces an error object. Any other running workflows in the process or sub-process remain uninfluenced.

Attributes

ErrorRef

The reference code of the produced error object.

25.3.1.6. Cancel End Event

The Cancel End Event triggers compensation events defined for the namespace, and the process or sub-process finishes as **CANCELED**.

25.3.1.7. 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.

25.3.1.8. 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 produces a signal identified by its **SignalRef** property.

25.4. 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 or end events.

The Scope data input is an optional property implemented to provide the following scope of throw events:

- **Process Instance:** Catch events only in the process instance will be able to catch this signal.
- **Default:** Catch events in a given KIE session will be able to catch this signal. The behavior varies depending on the KIE session strategy:
 - **Singleton:** Signal reaches all process instances available to the KIE session.
 - **Per request:** Signal reaches only the current process instance and start processes with a Signal Start Event.
 - **Per process:** same as **per request**.
- **Project:** Signal reaches all active process instances of a given deployment and start processes with a Signal Start Event.
- **External:** Enables the signal to reach the same process instances as with the **Project** scope, as well as process instances across deployments. To send the signal to a process instance across deployments, create a **SignalDeploymentId** process variable that provides information about what deployment or project should be the target of the signal. Broadcasting the signal would have negative impact on performance in larger environments.

[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>

CHAPTER 26. GATEWAYS

26.1. 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 of two types:

- Converging, that is merging multiple flows into one flow.
- Diverging, that is splitting one Flow into multiple flows.

One Gateway cannot have multiple incoming *and* multiple outgoing flows.

You can use the following types of gateways:

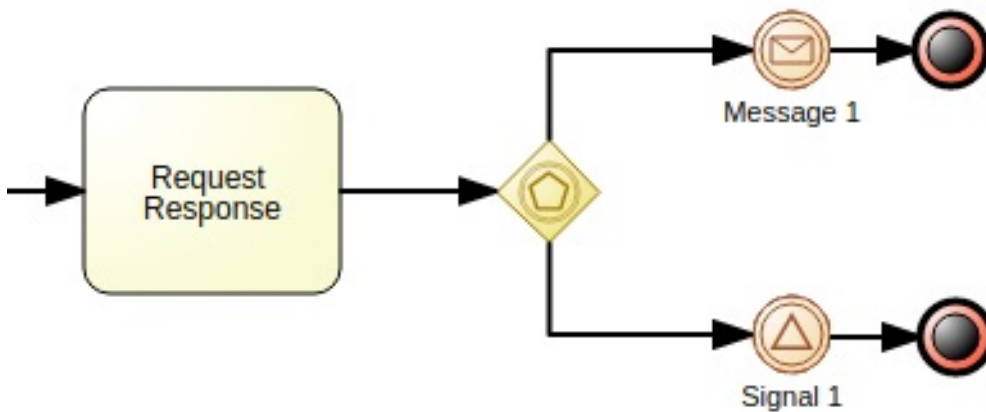
- Parallel (AND)
 - Converging AND gateway waits for all incoming flows before continuing to the outgoing flow.
 - Diverging AND gateway starts all outgoing flows simultaneously.
- Inclusive (OR)
 - Converging OR gateway waits for all incoming flows whose condition evaluates to true.
 - Diverging OR gateway starts all outgoing flows whose condition evaluates to true.
- Exclusive (XOR)
 - Converging XOR gateway waits for the first incoming flow whose condition evaluates to true.
 - Diverging XOR gateway starts only one outgoing flow.
 - Data-based exclusive gateways, which can be both diverging and converging, and are used to make decisions based on available data. For further information, see [Section 26.2.4, “Data-based Exclusive Gateway”](#).
- Event-based gateways, which can only be diverging, and are used for reacting to events. For further information, see [Section 26.2.1, “Event-based Gateway”](#).

26.2. GATEWAY TYPES

26.2.1. 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. The outgoing flow is taken based on the event that occurs. Only one outgoing flow is taken at a time.



The gateway might act as a start event, where the process is instantiated only if one of the intermediate events connected to the Event-Based Gateway occurs.

26.2.2. 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.

26.2.3. Inclusive Gateway

Diverging

Once the incoming flow is taken, all outgoing flows that evaluate to true are taken. Connections with lower priority numbers are triggered before triggering higher priority ones. Priorities are evaluated but the BPMN2 specification does not guarantee the priority order. It is recommended that you do not depend on the **priority** attribute in your workflow.



IMPORTANT

Ensure 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.

26.2.4. Data-based Exclusive Gateway

Diverging

The gateway triggers exactly one outgoing flow. The flow with the constraint evaluated to true and the *lowest* priority number is taken.



IMPORTANT

Ensure 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 allows a workflow branch to continue to its outgoing flow as soon as it reaches the gateway. When one of the incoming flows triggers the gateway, the workflow continues to the outgoing flow of the gateway. If it is triggered from more than one incoming flow, it triggers the next node for each trigger.

Attributes

Default gate

The outgoing flow taken by default if no other flow can be taken.

[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>

CHAPTER 27. ACTIVITIES, TASKS AND SUB-PROCESSES

27.1. 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:

- A sub-process; compound, can be broken down into multiple process elements.
- A task; atomic, represents a single unit of work.

An activity in Red Hat JBoss BPM Suite expects one incoming and one outgoing flow. If you want to design an activity with multiple incoming and multiple outgoing flows, set the system property **jbpm.enable.multi.con** to **true**. For more information about system properties, see chapter [System Properties](#) of the *Red Hat JBoss BPM Suite Administration and Configuration Guide* .

Activities share properties **ID** and **Name**. Note that activities, that is all tasks and sub-processes, have additional properties specific for the given activity or task type.

27.2. ACTIVITY MECHANISMS

27.2.1. Multiple Instances

You can run activities in multiple instances during execution. Individual instances are executed in a sequence. The instances are run based on a collection of elements. For every element in the collection, a new activity instance is created.

Every multiple-instance activity has the **Collection Expression** attribute that maps the input collection of elements to a single element. The multiple-instance activity then iterates through all the elements of the collection.

27.2.2. Activity Types

27.2.2.1. 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, that is a 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, the activity creates an instance of a process with the defined ID.

Attributes

Called Element

The ID of the process to be called and instantiated by the activity.

27.3. TASKS

A task is the smallest unit of work in a process flow. Red Hat JBoss BPM Suite uses the BPMN guidelines to separate tasks based on the types of inherent behavior that the tasks represent. This section defines all task types available in Red Hat JBoss BPM Suite except for the User Task. For more information about the User Task, see [Section 27.5, "User Task"](#).

27.3.1. None 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]

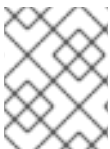
27.3.2. Send Task

"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 ID of the generated message object.



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.

27.3.3. 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

ID of the associated message object.

27.3.4. 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]

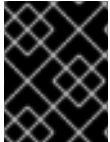
27.3.5. Service Task

Use a Service Task to invoke web services and Java methods.

Table 27.1. Service Task Attributes

Attribute	Description
Implementation	The underlying technology used for implementing the task. Possible values are WebService , which is the default value, and unspecified .
OperationRef	Specifies the operation that is invoked by the task: typically a particular method of a Java class or a web service method.

27.3.5.1. Using Service Task to Invoke Web Service



IMPORTANT

The preferred way of invoking web services is to use a WS Task, as opposed to a generic Service Task. For more information, see [Chapter 31, WS Task](#).

The default implementation of a Service Task in the BPMN2 specification is a web service. The web service support is based on the Apache CXF dynamic client, which provides a dedicated Service Task handler that implements the **WorkItemHandler** interface:

```
org.jbpm.process.workitem.bpmn2.ServiceTaskHandler
```

As a part of the process definition, you must first configure the web service:

1. Open the process in Process Editor.
2. Open the **Properties** panel on the right and click the **Value** field next to the **Imports** property. Click the arrow that appears on the right to open the **Editor for Imports** window.
3. Click **Add Import** to import the required WSDL (*Web Services Description Language*) values. For example:
 - **Import Type:** **wsdl**
 - **WSDL Location:** <http://localhost:8080/sample-ws-1/SimpleService?wsdl>
The WSDL location points to the WSDL file of your service.
 - **WSDL Namespace:** <http://bpmn2.workitem.process.jbpm.org/>
The WSDL namespace must match **targetNamespace** from your WSDL file.
4. Drag a Service Task (**Tasks** → **Service**) from the **Object Library** into the canvas.
5. Click the task, and in the **Properties** panel on the right, set the following:
 - **Service Implementation:** **Webservice**
 - **Service Interface:** **SimpleService**
 - **Service Operation:** **hello**
6. In the **Core Properties** section, click the **Value** field next to the **Assignments** property. Click the arrow that appears on the right to open the **Data I/O** window and do the following:
 - a. Provide a data input named **Parameter**.

- b. Optionally, provide a data output named **Result**.

For an example setting in the Service Task **Data I/O** window, see the image below:

The screenshot shows the 'Service Task Data I/O' dialog box. It contains two main sections: 'Data Inputs and Assignments' and 'Data Outputs and Assignments'. Each section has a table with columns for Name, Data Type, and Source/Target, along with an '+ Add' button and a trash icon. At the bottom, there are 'Cancel' and 'Save' buttons.

Data Inputs and Assignments			
Name	Data Type	Source	
Parameter	String	"John"	

Data Outputs and Assignments			
Name	Data Type	Target	
Result	Integer	count	

To use a request or a response object of the service as a variable, the objects must all implement the **java.io.Serializable** interface to use persistence properly. To add the interface while generating classes from WSDL, configure the JAXB API:

1. Create an XML binding file with the following contents.

```
<?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 plug-in (**cxf-codegen-plugin**) to the **pom.xml** file of the project:



```
<build>
  <plugins>
    <plugin>
      <groupId>org.apache.cxf</groupId>
      <artifactId>cxf-codegen-plugin</artifactId>
      <version>CXF_VERSION</version>
      ...
    </plugin>
  </plugins>
</build>
```


27.3.5.2. Using Service Task to Invoke Java Method

You can use a Service Task to invoke a method of a particular Java class. The method can have only one parameter and returns a single value. If the invoked Java class is not a part of the project, add all the required dependencies to the **pom.xml** file of the project.

The following procedures use an example class **WeatherService** with a method **int getTemperature(String location)**. The method has one parameter (**String location**) and returns a single value (**int temperature**).

Invoking Java Method in Red Hat JBoss Developer Studio

1. In Red Hat JBoss Developer Studio, open the business process that you want to add a Service Task to, or create a new process with a start and an end event.
2. Select **Window** → **Show View** → **Properties**, and click **Interfaces** in the lower-right corner of the **Properties** panel.
3. Click the **Import** icon () to open the **Browse for a Java type to Import** window. To find the Java type, start typing **WeatherService** in the **Type** field. In the **Available Methods** list box below, select the **int getTemperature(String)** method. Click **OK**.
Note that it is also possible to select the **Create Process Variables** check box to automatically import process variables with generated names. In this procedure, the process variables are created manually.
4. In the **Properties** panel, click **Data Items**. Click the **Add** icon () to create a *local* process variable:
 - a. Enter the process variable details:
 - **Name: location**
 - **Data Type: java.lang.String**
 - b. Create a second process variable:
 - **Name: temperature**
 - **Data Type: java.lang.Integer**
5. Add a Service Task to the process:
 - a. Drag a Service Task (**Tasks** → **Service Task**) from the **Palette** panel on the right to the canvas.
 - b. Double-click the Service Task on the canvas to open the **Edit Service Task** window. Click **Service Task** and set the following properties:
 - **Implementation: Java**
 - **Operation: WeatherService/getTemperature**
 - **Source: location**
 - **Target: temperature**
 - c. Click **OK** and save the process.

6. The Java application that starts the business process must be available. If you created a new business process and do not have the application, create a new jBPM project with an example application:
 - a. Click **File** → **New** → **Other** → **jBPM** → **jBPM project**. Click **Next**.
 - b. Select the second option and click **Next** to create a project and populate it with some example files to help you get started quickly.
 - c. Enter a project name and select the **Maven** radio button. Click **Finish**.
7. Register work item handlers. In the **src/main/resources/META-INF/** directory, create a file named **kie-deployment-descriptor.xml** with the following contents:

```

<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<deployment-descriptor xsi:schemaLocation="http://www.jboss.org/jbpm deployment-
descriptor.xsd" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <persistence-unit>org.jboss.domain</persistence-unit>
  <audit-persistence-unit>org.jboss.domain</audit-persistence-unit>
  <audit-mode>JPA</audit-mode>
  <persistence-mode>JPA</persistence-mode>
  <runtime-strategy>SINGLETON</runtime-strategy>
  <marshalling-strategies/>
  <event-listeners/>
  <task-event-listeners/>
  <globals/>
  <work-item-handlers>
    <work-item-handler>
      <resolver>mvel</resolver>
      <identifier>new
org.jboss.process.instance.impl.demo.SystemOutWorkItemHandler()</identifier>
      <parameters/>
      <name>Log</name>
    </work-item-handler>
    <work-item-handler>
      <resolver>mvel</resolver>
      <identifier>new org.jboss.process.workitem.bpmn2.ServiceTaskHandler(ksession,
classLoader)</identifier>
      <parameters/>
      <name>Service Task</name>
    </work-item-handler>
    <work-item-handler>
      <resolver>mvel</resolver>
      <identifier>new
org.jboss.process.workitem.webservice.WebServiceWorkItemHandler(ksession,
classLoader)</identifier>
      <parameters/>
      <name>WebService</name>
    </work-item-handler>
    <work-item-handler>
      <resolver>mvel</resolver>
      <identifier>new
org.jboss.process.workitem.rest.RESTWorkItemHandler(classLoader)</identifier>
      <parameters/>
      <name>Rest</name>
    </work-item-handler>
  </work-item-handlers>
</deployment-descriptor>

```

```

</work-item-handlers>
<environment-entries/>
<configurations/>
<required-roles/>
<remoteable-classes/>
<limit-serialization-classes>true</limit-serialization-classes>
</deployment-descriptor>

```

8. Open the **ProcessMain.java** file that is located in the **src/main/java** directory, and modify the code of the application that starts the business process:

- a. Initialize the process variables:

```

Map<String, Object> arguments = new HashMap<>();
arguments.put("location", "Brno");
arguments.put("temperature", -1);

```


- b. Start the process:

```

ksession.startProcess("demo-package.demo-service-task", arguments);

```

Invoking Java Method in Business Central

1. The invoked Java class must be available either on the class path or in the dependencies of the project. To add the class to the dependencies of the project:
 - a. In Business Central, click **Authoring** → **Artifact Repository**.
 - b. Click **Upload** to open the **Artifact upload** window.
 - c. Choose the **.jar** file, and click  .
 - d. Click **Authoring** → **Project Authoring**, and find or create the project you want to use.
 - e. Click **Open Project Editor** and then **Project Settings: Project General Settings** → **Dependencies**.
 - f. Click **Add from repository**, locate the uploaded **.jar** file, and click **Select**.
 - g. Save the project.
2. Open or create the business process to which you want to add a Service Task.
3. In Process Editor, open the **Properties** panel on the right and click the **Value** field next to the **Imports** property. Click the arrow that appears to open the **Editor for Imports** window. In the window:
 - a. Click **Add Import** and specify the following values:
 - **Import Type:** **default**
 - **Custom Class Name** fully qualified name of the invoked Java class, for example **org.jboss.weather.WeatherService**
 - b. Click **Ok**.

4. Create process variables:
 - a. In the **Properties** panel, click the **Value** field next to the **Variable Definitions** property. Click the arrow that appears to open the **Editor for Variable Definitions** window.
 - b. Click **Add Variable** to add the following two process variables:
 - **Name: temperature, Defined Types: Integer** (or **Custom Type: java.lang.Integer**)
 - **Name: location, Defined Types: String** (or **Custom Type: java.lang.String**)
 - c. Click **Ok**.
5. To add a Service Task into the process, drag and drop a Service Task (**Tasks** → **Service**) from the **Object Library** panel on the left into the canvas.
6. Click the Service Task on the canvas to open its properties on the right, and set the following properties:
 - **Service Interface: org.jboss.weather.WeatherService**
 - **Service Operation: getTemperature**
7. Click the **Value** field next to the **Assignments** property. Click the arrow that appears to open the **Data I/O** window and do the following:
 - a. Click **Add** next to **Data Inputs and Assignments** and add the following:
 - **Name: Parameter, Data Type: String, Source: location**
 - **Name: ParameterType, Data Type: String, Source: java.lang.String** (to add this value, click **Constant ...** and type it manually)
 - b. Click **Add** next to **Data Outputs and Assignments** and add the following:
 - **Name: Result, Data Type: Integer, Target: temperature**
 - c. Click **Save**.

27.3.6. 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 Business 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

ruleflow-group

The name of the ruleflow group that includes the set of rules to be evaluated by the task. This attribute refers to the **ruleflow-group** keyword in your **DRL** file.

27.3.7. Script Task

A Script Task represents a script to be executed during the process execution.

The associated **Script** can access process variables and global variables. When using a Script Task:

- Avoid low-level implementation details in the process. A Script Task could be used to manipulate variables, but consider using a Service Task when modelling more complex operations.
- 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 notifying 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 managed, for example, 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 executed and the outgoing flow is taken.

Attributes

Script

The script to be executed.

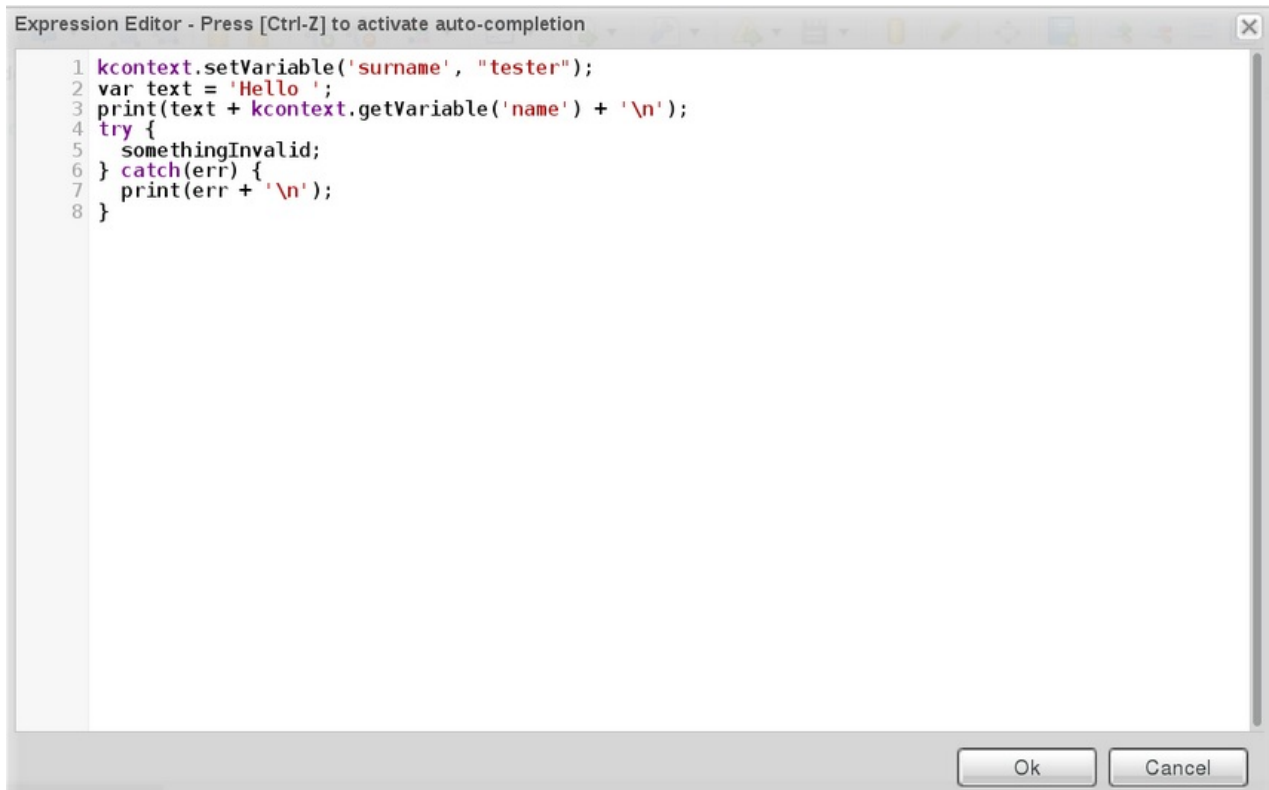
Script Language

The language in which the script is written.

From Red Hat JBoss BPM Suite 6.2 onwards, JavaScript is supported as a 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. Write the script to be executed in the **Expression Editor** window and click **Ok**.

Example 27.1. Script Task in Business Central using JavaScript



```

Expression Editor - Press [Ctrl-Z] to activate auto-completion
1 kcontext.setVariable('surname', "tester");
2 var text = 'Hello ';
3 print(text + kcontext.getVariable('name') + '\n');
4 try {
5     somethingInvalid;
6 } catch(err) {
7     print(err + '\n');
8 }
Ok Cancel

```

27.4. 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]”

You can understand a sub-process 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 ends when there are no more active elements in it.

The following sub-process types are supported:

- Ad-Hoc sub-process, which has no strict element execution order.
- Embedded sub-process, which is a part of the parent process execution and shares its data.
- Reusable sub-process, which is independent from its parent process.
- Event sub-process, which is only triggered on a start event or a timer.

Note that any sub-process type can be a multi-instance sub-process.

27.4.1. Embedded Sub-Process

An embedded sub-process 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.

27.4.2. 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 description--i.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]”

Attributes

AdHocCompletionCondition

When this condition evaluates to **true**, the execution finishes.

AdHocOrdering

Enables you to choose paralel or sequential execution of elements inside of the sub-process.

Variable Definitions

Enables you to define process variables available only for elements of the sub-process.

27.4.3. Multi-instance Sub-Process

A Multiple Instances Sub-Process is instantiated multiple times when its execution is triggered. The instances are created in a sequential manner, that means a new sub-process instance is created only after the previous instance has finished.

A Multiple Instances Sub-Process has one incoming connection and one outgoing connection.

Attributes

MI collection input

A collection to be iterated through. It is used to create individual instances of given activity. The sub-process will be run with each element of this collection.

MI collection output

A collection of the sub-process execution results.

MI completion condition

An MVEL expression evaluated at the end of every instance. When evaluated as **true**, the sub-process is evaluated as finished and the sub-process's outgoing flow is taken. Possible remaining sub-process instances are cancelled.

MI data input

A variable name for each element from the collection that will be used in the process.

MI data output

An optional variable name for the collection of the results.

27.4.4. 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 the sub-process. The sub-process is not part of the regular control flow. Although self-contained, it is executed in the context of the bounding sub-process.

Use an event sub-process within a process flow to handle events that happen outside of the main process flow. For example, while booking a flight, two events may occur:

- Cancel booking (interrupting).
- Check booking status (non-interrupting).

Both these events can be modeled using the event sub-process.

27.5. 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. The User Task is atomic.

On execution, the User Task element is instantiated as a task that appears in the list of tasks of one or multiple actors.

If a User Task element defines the **Groups** attribute, 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 list of the other users.

Note that User Task is implemented as a domain-specific task and serves as a base for your custom tasks. For further information, see [Section 4.14.1, "Work Item Definition"](#).

Attributes

Actors

A comma-separated list of users who can perform the generated task.

Content

The data associated with this task. This attribute does not affect TaskService behavior.

CreatedBy

The name of the user or ID of the process that created the task.

Groups

A comma-separated list of groups who can perform the generated task.

Locale

The locale for which the element is defined. This property is not used by the Red Hat JBoss BPM Suite engine at the moment.

Notifications

A definition of notification applied to the User Task. For further information, see [Section 27.5.3, “Notification”](#).

Priority

An integer value defining the User Task priority. The value influences the User Task ordering in the user Task list and the simulation outcome.

Reassignment

The definition of escalation applied to the User Task. For further information, see [Section 27.5.2, “Reassignment”](#).

ScriptLanguage

The language of the script. Choose between **Java**, **MVEL**, or **Javascript**.

Skippable

A 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 during runtime. It is displayed in the task list in 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.

27.5.1. User Task lifecycle

When a User Task element is triggered 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. For further information about the Task Execution Engine, see the *Red Hat JBoss BPM Suite Administration and Configuration Guide*. The Process instance continues the execution only when the associated User Task has been completed or aborted.

See the User Task lifecycle:

- 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 task appears in the task list of all the actors that are allowed to execute the task.
- When one of the actors claims the User Task, 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.

27.5.2. Reassignment

The reassignment mechanism implements 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.

A reassignment can start if a User Task is in one of the following states for a defined amount of time:

- When not started: **READY** or **RESERVED**.
- When 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 set 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:** A comma-separated list of user IDs that are reassigned to the task on escalation. It can be a String or an expression, such as **#{user-id}**.
- **Groups:** A comma separated list of group IDs that are reassigned to the task on escalation. It can be a String or an expression, such as **#{user-id}**.
- **Expires At:** A time definition when escalation is triggered. It can be a String or an expression, such as **#{expiresAt}**. For further information about time format, see [Chapter 24, Timing](#).
- **Type:** A state in which the task needs to be at the given **Expires At** time so that the escalation is triggered.

27.5.3. Notification

The notification mechanism provides the capability to send an e-mail notification if a User Task is in one of the following states for the specified time:

- When not started: **READY** or **RESERVED**.
- When not completed: **IN_PROGRESS**.

A notification is defined in the **Notification** property of User Task elements. The property accepts an arbitrary number of notification definitions with the following parameters:

- **Type:** The state in which the User Task needs to be at the given **Expires At** time so that the notification is triggered.
- **Expires At:** A time definition when notification is triggered. It can be a String value or expression, such as **#{expiresAt}**. For information about time format, see [Chapter 24, Timing](#).
- **From:** The user or group ID of users used in the **From** field of the email notification message. It can be a String or expression.
- **To Users:** A comma-separated list of user IDs to which the notification is sent. It can be a String or expression, such as **#{user-id}**.
- **To Groups:** A comma separated list of group IDs to which the notification is be sent. It can be a String or expression, such as **#{group-id}**.
- **Reply To:** A user or group ID that receives any replies to the notification. It can be a String or expression, such as **#{group-id}**.
- **Subject:** The subject of the email notification. It can be a String or an expression.

- **Body:** The body of the email notification. It can be a String or an expression.

Available variables

A notification can reference process variables by using the **#{processVariable}** syntax. Similarly, task variables use the **\${taskVariable}** syntax.

In addition to custom task variables, the notification mechanism can use the following local task variables:

- **taskId:** The internal ID of the User Task instance.
- **processInstanceId:** The internal ID of task's parent process instance.
- **workItemId:** The internal ID of a work item that created the User Task.
- **processSessionId:** The knowledge session ID of the parent process instance.
- **owners:** A list of users and groups that are potential owners of the User Task.
- **doc:** A map that contains task variables.

Example 27.2. 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>
```

- [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>

CHAPTER 28. CONNECTING OBJECTS

28.1. 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

28.2. CONNECTING OBJECTS TYPES

28.2.1. 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

When this condition evaluates to **true**, the workflow takes the sequence flow.

If a sequence flow has a gateway element as its source, you need to define a conditional expression that is evaluated before the sequence flow is taken. If evaluated to **false**, the workflow attempts to switch to another sequence flow. If evaluated to **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, Javascript, MVEL, 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.

CHAPTER 29. SWIMLANES

Swimlanes visually group tasks related to one group or user. For example, you can create a marketing task swimlane to group all User Tasks related to marketing activities into one Lane.

29.1. 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:

1. Click the **Swimlanes** menu item in the Object Library.
2. Drag and drop the Lane artifact to your process model.

This artifact is a box into which you can add your User Tasks.

Lanes should be given unique names and background colors to fully separate them into functional groups. You can do so in the properties panel of a lane.

During runtime, lanes auto-claim or assign tasks to a user who has completed a different task in 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. At runtime, the process will stop its execution.

For example, suppose 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, or completed by an **analyst** user, UT2 gets claimed and assigned to the user who completed UT1. If only UT1 has the **analyst** group assigned, and UT2 has no user or group assignments, the process stops after UT1 had been completed.

[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>

CHAPTER 30. ARTIFACTS

30.1. ARTIFACTS

Any object in the BPMN diagram that is not a part of the process workflow is an artifact. Artifacts have no incoming or outgoing flow objects. The purpose of artifacts is to provide additional information needed to understand the diagram.

30.2. 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. Data Objects have separate visualization properties and variable properties.

APPENDIX B. SERVICE TASKS: WS TASK, EMAIL TASK, REST TASK


Service task is a task that uses a service, such as a mail service, web service, or another service.

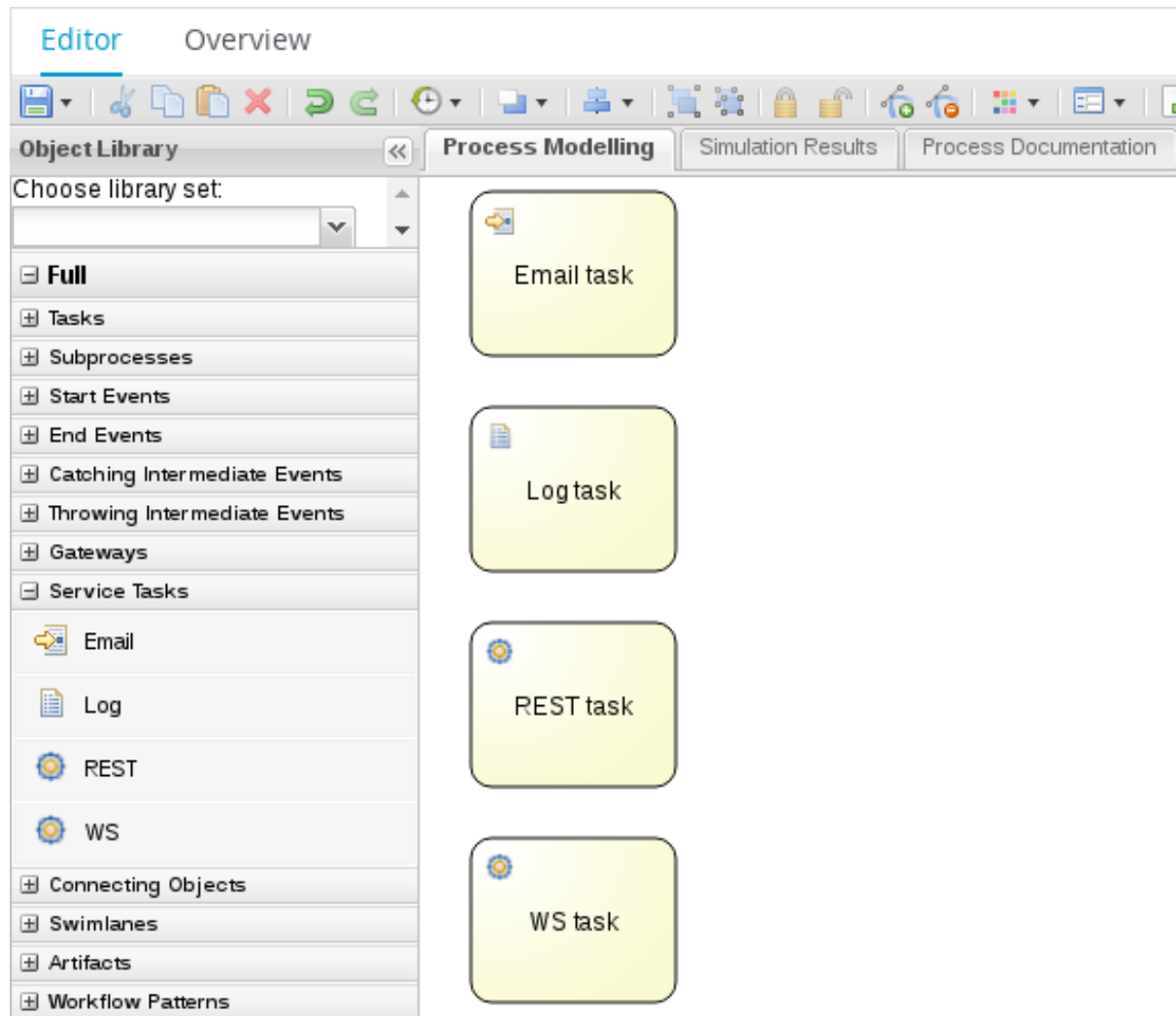
Red Hat JBoss BPM Suite contains the following predefined service tasks:

- A Web Service task for invoking a web service from a BPMN2 process.
- An Email task for sending emails through a mail server.
- A Log task that calls the **SystemOutWorkItemHandler** class.
- A REST task for sending REST calls.

Note that since the tasks extend the service task, their attributes are implemented as **Data Input Set** assignments, and **Data Output Set** assignments, not as separate properties.

To use the service tasks in your process:

1. In Business Central, click **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 click  to expand the **Object Library**.
4. Expand the **Service Tasks** section and drag and drop the selected service task to the required position on the canvas.



For more information about service tasks, see [Chapter 31, *WS Task*](#), [Chapter 32, *Email Task*](#), and [Chapter 33, *REST Task*](#).




If you require other task types, implement your task as instructed in [Section 4.14, "Domain-Specific Tasks"](#).

CHAPTER 31. WS TASK

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.

31.1. 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, click  to open the **Properties** panel.
2. Select the **Variable Definitions** property and create a process variable called **pVar** with the custom type **Object[]**.
3. Click the **WS** task in the Process Designer and click  to open the **Properties** panel.
4. Click  next to the **Assignments** property.
5. Change the **Parameter** input variable from **String** to **Custom** and enter **Object[]**. Select **pVar** in the **Source** field. Click **Ok**.
6. In the property panel of the **WS** task, enter the following in the **On Entry Actions** property:

```
Object[] params = {"firstParam", "secondParam"}; kcontext.setVariable("pVar", params);
```

31.2. CUSTOM OBJECTS

In addition to primitive object types, the WebService task can use custom objects, such as **Person** or **Employee**.

To use custom objects:

1. Create a custom model object 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 **WS** 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(ksession,
runtimeManager.getEnvironment().getClassLoader())" name="WebService"/>
    </workItemHandlers>
  </ksession>
</kbase>
</kmodule>

```

The above configuration utilizes the **WebServiceWorkItemHandler**.

31.3. 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.


CHAPTER 32. EMAIL TASK

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 the Email task to the workflow.
2. Select the Email task and click  to open the **Properties** panel.
3. Open **Assignments** and fill in the **To**, **From**, **Subject**, and **Body** properties, and any other relevant input attributes.

Data Inputs and Assignments

[+ Add](#)

Name	Data Type	Source	
To	String	"abc.xyz@test.cc"	
Body	String	"Hello, how are y"	
From	String	"admin@test.cor"	
Subject	String	"Test Hello"	

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","p
assword");
```

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

You can configure the Deadline email feature in two ways:

1. Mail Session on Container Level

With this method, the Deadline email feature uses **EmailSessionProducer** to look up the **mail/jbpmMailSession** using JNDI. The following example is for Red Hat JBoss EAP **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>
```

2. Using email.properties

If the **mail/jbpmMailSession** is not found, Red Hat JBoss BPM Suite searches for **/email.properties** on the class path with content similar to 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.

CHAPTER 33. REST TASK

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=#{carBrand}
```

In this example, `carBrand` is replaced by the value of the **carBrand** process variable during runtime.

Method

The method of the request, such as GET, POST, or other. The default method is GET.

ContentType

The data type if you are sending data. The supported data types are **application/json** and **application/xml**. This attribute is mandatory for POST and PUT requests. If you want to use this attribute, map it as a data input variable in the **Data I/O** dialogue of the task.

Content

The data you want to send. This attribute is mandatory for POST and PUT requests. This is an optional parameter. If you want to use it, map it as a data input variable in the **Data I/O** dialogue of the task.

ConnectTimeout

The connection timeout. The default value is 60 seconds.

ReadTimeout

The timeout on response. The default value is 60 seconds.

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.

ResultClass

This attribute determines the class to which the value from the **Result** attribute will be converted. If not provided, the default value is **String**.

HandleResponseErrors

An optional parameter that instructs the handler to throw errors in case of unsuccessful response codes. For information on how to handle response errors, see [the section called "Handling REST Response Error"](#).

Output Attributes

Result

The result returned by the REST service.

Status

The variable contains a value from interval 200 to 300 if the REST request was successful, or an error response code if the request was unsuccessful. This variable is not mapped by default. If you want to use this variable, map it manually as an output variable of the REST task.

StatusMsg

If the service returned an error response, this variable will contain the error response message. This variable is not mapped by default. If you want to use this variable, map it manually as an output variable of the REST task.

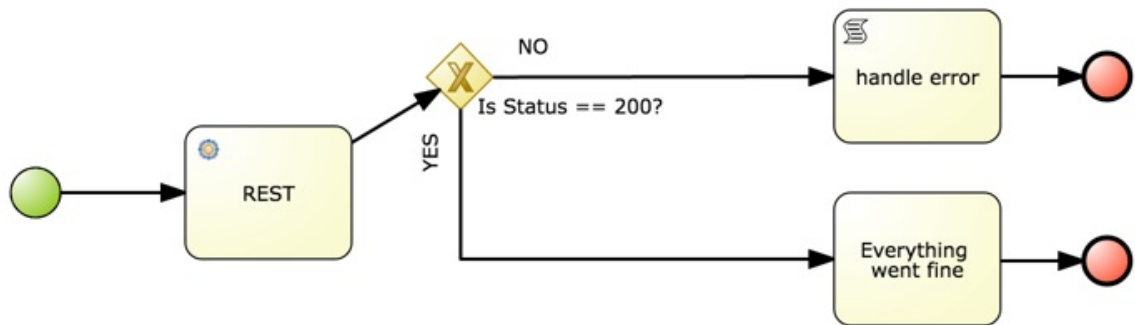
All output attributes are **String** by default.

Handling REST Response Error

HandleResponseErrors can be handled in two ways:

1. In the Process Definition Workflow

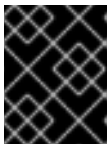
- a. **Status**: When **RESTWorkItemHandler** produces a **Status** output variable that includes an HTTP response code. This can be mapped to a process variable and used in a XOR gateway to determine the service outcome.



- 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

To enable this feature, set the REST work item input variable **HandleResponseErrors** to **true**.



IMPORTANT

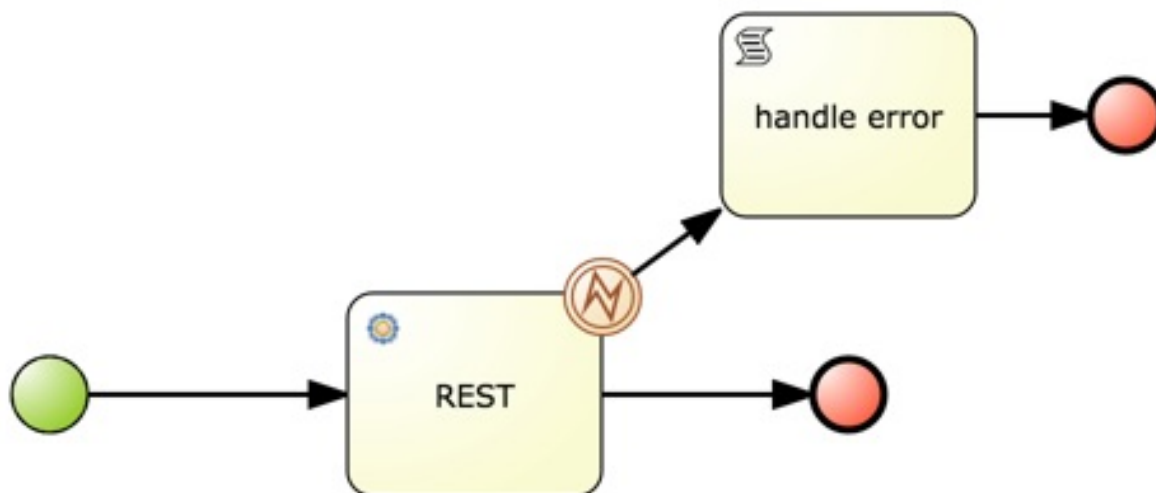
The **HandleResponse** must have a valid boolean expression or be left empty, which is equivalent to **false**. Otherwise, the REST task will throw an exception.

When the REST work item input variable **HandleResponseErrors** is set to **true**, the **RESTWorkItemHandler** handler 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 " + endpoint
        + ", response " + response + "");
}
  
```

With the **HandleResponseErrors** 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.

APPENDIX C. SIMULATION DATA

CHAPTER 34. PROCESS

Simulation Attributes

Base currency

The currency used for simulation.

Base time unit

The time unit to apply to all the time definitions in the process.

CHAPTER 35. ACTIVITIES

All the activities available in Red Hat JBoss BPM Suite, except for the human task, share the following properties. See [Chapter 40, *Human Tasks*](#) for properties specific to human tasks.

Simulation Attributes

Cost per time unit

The cost for every time unit lapsed during simulation.

Distribution Type

For information about the **Distribution type** property, and the **Processing time** property if applicable, see [Chapter 42, *Distribution Types*](#).

CHAPTER 36. START EVENT

Simulation Attributes

Distribution Type

For information about the **Distribution type** property, and the **Processing time** property if applicable, see [Chapter 42, *Distribution Types*](#).

Probability (Boundary Event only)

The probability of triggering the element.

CHAPTER 37. CATCHING INTERMEDIATE EVENTS

Simulation Attributes

Distribution Type

For information about the **Distribution type** property, and the **Processing time** property if applicable, see [Chapter 42, *Distribution Types*](#).

Probability (Boundary Event only)

The probability of triggering the element.

CHAPTER 38. SEQUENCE FLOW

Simulation Attributes

Probability (Boundary Event only)

The percentual probability that the flow is taken.

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, ensure their sum is 100.

CHAPTER 39. THROWING INTERMEDIATE EVENTS

Simulation Attributes

Distribution Type

For information about the **Distribution type** property, and the **Processing time** property if applicable, see [Chapter 42, *Distribution Types*](#).

CHAPTER 40. HUMAN TASKS

Simulation Attributes

Cost per time unit

The cost for every time unit lapsed during simulation.

Distribution Type

For information about the **Distribution type** property, and the **Processing time** property if applicable, see [Chapter 42, Distribution Types](#).

Staff availability

The number of actors available to work on the given task.

Example 40.1. Staff Availability Impact

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** property is set to **3**.
- Only one person is available to work on the Human Tasks; the **Staff availability** property is set to **1**.

That results in the following:

- The Human Task generated by the first process instance will be executed in 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 and becomes available only after another 2.5 hours. It 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

A time period after which the task completes in a simulation. If the task should take an hour to complete, set this property to **1**.

CHAPTER 41. END EVENTS

Simulation Attributes

Distribution Type

For information about the **Distribution type** property, and the **Processing time** property if applicable, see [Chapter 42, *Distribution Types*](#).

CHAPTER 42. DISTRIBUTION TYPES

The **Distribution type** property defines the distribution of possible time values, that is scores, of process elements.

The elements might use one of the following score distribution types on simulation:

- **Normal:** the values are distributed on a bell-shaped, symmetrical curve.
- **Uniform:** the values have a rectangular distribution, that means every value is applied the same number of times.
- **Poisson:** the values are distributed on a negatively-skewed normal distribution curve.

42.1. NORMAL

The element values are picked based on the normal distribution type, which is bell-shaped and symmetrical.

Normal Distribution Attributes

Processing time (mean)

The mean processing time that the element needs in order to be processed. The value uses the time unit defined in the **Base time unit** property.

Standard deviation

The standard deviation of the processing time. The value uses the time unit defined in the **Base time unit** property.

42.2. UNIFORM

The Uniform distribution or rectangular distribution returns the possible values with the same levels of probability.

Uniform Distribution Attributes

Processing time (max)

The maximum processing time of the element.

Processing time (min)

The minimum processing time of the element.

42.3. 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.

42.3.1. Poisson Distribution Attributes

Processing time (mean)

The mean time for the element processing. The value uses the time unit defined in the **Base time unit** property.

APPENDIX D. VERSIONING INFORMATION

Documentation last updated on: Wednesday, Oct 23, 2019.