

Compliments of  RED HAT  
DEVELOPERS



# JBoss BPM

Selections from  
*Effective Business Process Management with JBoss BPM*  
by Eric D. Schabell

 manning



*Selections from*  
*Effective Business Process Management with JBoss BPM*

Selected by Eric D. Schabell

**Chapter 1**  
**Chapter 2**  
**Chapter 3**  
**Chapter 4**  
**Chapter 5**

Copyright 2017 Manning Publications  
To pre-order or learn more about these books go to [www.manning.com](http://www.manning.com)

For online information and ordering of these and other Manning books, please visit [www.manning.com](http://www.manning.com). The publisher offers discounts on these books when ordered in quantity.

For more information, please contact


Special Sales Department  
Manning Publications Co.  
20 Baldwin Road  
PO Box 761  
Shelter Island, NY 11964  
Email: [orders@manning.com](mailto:orders@manning.com)

©2017 by Manning Publications Co. All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by means electronic, mechanical, photocopying, or otherwise, without prior written permission of the publisher.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in the book, and Manning Publications was aware of a trademark claim, the designations have been printed in initial caps or all caps.

- ⊗ Recognizing the importance of preserving what has been written, it is Manning's policy to have the books we publish printed on acid-free paper, and we exert our best efforts to that end. Recognizing also our responsibility to conserve the resources of our planet, Manning books are printed on paper that is at least 15 percent recycled and processed without the use of elemental chlorine.

 Manning Publications Co.  
20 Baldwin Road Technical  
PO Box 761  
Shelter Island, NY 11964

Cover designer: Leslie Haimes

ISBN: 9781617295393  
Printed in the United States of America  
1 2 3 4 5 6 7 8 9 10 - EBM - 21 20 19 18 17 16

# *brief contents*

---

<b>1</b>	<b>WHAT'S IN A PROCESS .....</b>	<b>1</b>
1.1	Introducing BPM	2
1.2	An introduction to rules, events and processes	7
1.3	Understanding the role of community projects	10
1.4	Meet the JBoss BPM Suite	11
1.5	Summary	17
<b>2</b>	<b>PROCESSING FIRST STEPS .....</b>	<b>19</b>
2.1	Installing JBoss BPM	20
2.2	Start a first project	22
2.3	Touring JBoss BPM in the Cloud	34
2.4	Summary	36
<b>3</b>	<b>MODELING PROCESS DATA.....</b>	<b>37</b>
3.1	Data modeling tooling overview	38
3.2	Complete your data model	50
3.3	Summary	58
<b>4</b>	<b>STARTING WITH BUSINESS RULES .....</b>	<b>59</b>
4.1	Business logic central to your process	60
4.2	Considering technical and guided rules	64
4.3	Summary	85

<b>5</b>	<b>CREATING COMPLEX BUSINESS RULES.....</b>	<b>86</b>
5.1	Complex domains as natural language rules	87
5.2	Complex rules made easy with decision tables	96
5.3	Summary	104
	index	105

# 1

## *What's in a process*

---

### ***This chapter covers***

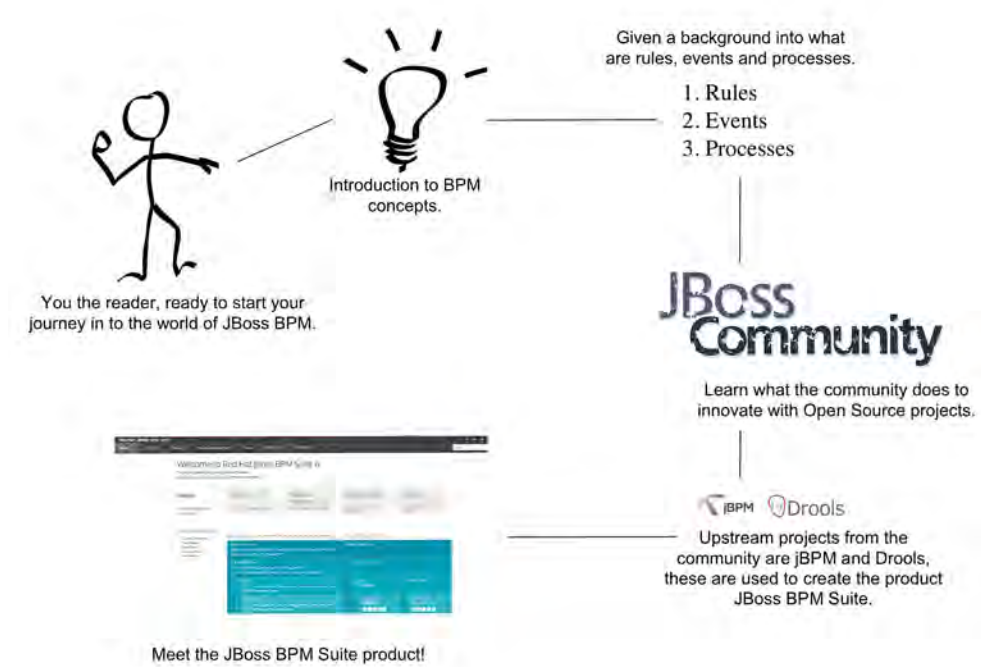
- Introducing basic BPM concepts
- Introducing rules, events and processes
- Discovering the community origins of JBoss BPM
- Walking through JBoss BPM Suite

This chapter introduces you to Business Process Management (BPM) and the important terminology used as you embark on the path to learn this technology. I begin with a process and what's within the context of BPM. As you explore JBoss BPM there are three important aspects that support integrating business activities into the processes you're developing. These aspects are business rules, business events and business processes. I discuss each and provide an overview showing how each can be used to support your process development projects.

Next is a tour of the community of Open Source projects that make up the JBoss BPM product eco-system. Projects are highlighted, specifically outlining how project code's integrated into a supported JBoss BPM Suite product. These communities allow you to keep an eye on the research and development being done in the area of rules, events and processes. You can influence the direction of this technology by providing feedback or code contributions.

Finally, you're taken on a tour of the JBoss BPM Suite architecture to explore the components that help you develop process projects. This is an introduction to how JBoss BPM Suite supports development, testing and runtime execution of processes. Testing's integrated into the chapters where you create artifacts and these tests demonstrate that your artifacts are working correctly. For more in-depth looks

at how a specific component works, refer to the chapter devoted to it. The path you're taking in this chapter's highlighted in figure 1.1.



**Figure 1.1** The path you take through this chapter, which introduces you to BPM concepts, the path from community project to products, and getting started with JBoss BPM.

## 1.1 **Introducing BPM**

Organizations are constantly being tested in the markets they operate in by shifting expectations of customers and by competitors looking to provide better value at a lower cost. This tension's the catalyst that continually pushes organizations to search for ways to improve their services, improve the speed which they deliver value to their customers, enable employees to get more done with less administrative overhead, and, most importantly, to constantly grow by generating more revenue. This is the basis of BPM, to be able to identify and capture processes in an organization to create repeatable, measurable and consistent execution of their goals to drive business forward.

When an organization studies its operations, it discovers how many processes are used in their daily business. These processes are often poorly thought out or were created to complete some aspect of the daily business in an ad-hoc fashion. Little thought was given to improving efficiency within the organization. At this point the organization's interested in finding ways to improve their processes through automation and provide a return which is represented in business value.

Business value could be anything that drives organizational goals forward to make customers happy and thereby generate more revenue. An example of value to your business could be keeping clear and concise track of interactions with your customers. If that data can be captured the marketing department could search a customer's behavioral patterns to decide what products and services to market directly. It'd take mass marketing out of the equation and allow for direct and specific marketing targeting individual customers' needs.

I choose to call this *business value*, but it's sometimes referred to in this domain as *knowledge* and the people working within a process are then known as *knowledge workers*. Either one works, but in this book, I'm sticking with *business value* to capture the spirit of the organizations we work for.

It's possible to identify pieces of business value that aren't worth automating because they're inconsistent or have too many potential paths to justify the effort to automate. Others require traditional human brain power, which isn't easy to capture in automated process form. An example of this is the hiring of employees, a process that can be largely automated, but the actual decision to hire a specific candidate remains a factor of human intelligence. You can automate the process of handling applications, scheduling interviews and the post process of on-boarding a new employee once hired. Let's leave the 'hire or not to hire' task to humans.

Another important facet of capturing business value in processes is that you can monitor processes and tasks as they're completed to provide business owners with valuable information. You can provide insights into aspects that interest the business owner, and make intelligent decisions about when and where to improve a process as the business evolves.

Imagine your business is running a retail process to sell products online. This process has a user task to approve large orders and is staffed by a set number of people during certain business hours. Bottlenecks may develop in your process as the business grows. What can you do when the Christmas holidays arrive and you expect a surge in orders? Does the current staffing of the user task allow you to process ten times the number of orders? What about one hundred times the orders?

By using historical data captured in previous process instances it's possible to determine how many orders are large enough to require human approval, and on average how long each approval took. If you simulate your process using tooling provided by JBoss BPM Suite, you can adjust the number of humans working on the approval task in the process and set how long they take. By simulating hundreds or even thousands of process instances you can record the results of the orders flowing through your process and determine whether you need to staff your user task differently. During the normal months of the year you can process large orders with two employees assigned for eight-hour days. During the holidays, due to expected increases, simulation testing reveals a need for twenty-four hour shifts to approve large orders and you need to increase staffing to four employees. It's always better to know this before hiring new employees for the holidays and discovering it didn't help process orders fast enough to justify the costs.



This process looks at a step-by-step plan to accomplish a set of tasks that deliver business value. The basic series of events that leads to defining a process begins with identifying the piece of business value to be automated. This process is selected for its potential to improve the business because:

- the process can be automated
- the process can be consistently executed in the same way
- you can clearly define human involvement in the process
- automation of the process removes current ad-hoc or inconsistent behavior
- measuring the process gains insights into current business behaviors
- insights provide a better means to decide when changes can and should be made to improve a process
- you can reduce resource waste by efficiently handling *wait states*

Once a potential business process is identified, a workshop's held with the users from the business unit responsible for executing the process. For example, the human resource department might be in charge of registering new employees, getting them a workstation, starting their benefits, and assigning them an e-mail address. This process is discussed and dissected to identify the exact steps and order that they'd need to be accomplished to register a new employee. This results are put in a diagram with each step, from start to finish, drawn up as tasks to be completed.

These are the beginnings of a process, known as a *process diagram*. A process diagram contains all the elements needed to capture the steps in the processes without any of the execution details. You identify various *tasks* or activities to be undertaken in the process and create separate steps for each one, known as *task nodes*. This process diagram with all the tasks contains not only task nodes, but also other variations like start nodes, end nodes, transition arrows, gateways that split paths of the process and gateways that join paths of the process together again. These are standard elements that are part of the *Business Process Modeling Notation (BPMN)* specification<sup>1</sup>.

The executions details needed to complete a process could be that a task in the process needs to send an email. The details missing to send an email from the task are the sender, receiver, subject line, and body of the email. These need to be defined in the e-mail task. Another task might be to fetch data from an existing service in the organization, therefore requiring definitions for the location of the service, the service name, any data it might need, and whatever else is needed.

A process isn't complete until defining the data needed for the process, which tasks are to be automated, which tasks humans must complete, what systems are to be integrated, and the execution details for each task.

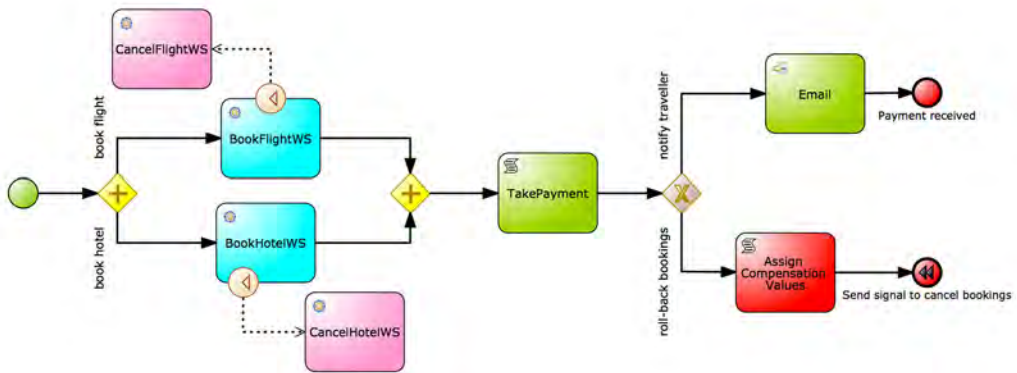
To illustrate this, imagine you're developing a process definition for a travel agency booking system. A part of the larger project's to define a process for register-

---

<sup>1</sup> The BPMN specification's a standard owned by the Object Management Group (OMG) that was put together to provide a single specification that tool vendors could then implement against to provide process definition portability. For more information see <http://www.omg.org/spec/BPMN/2.0>.

ing a selected flight, hotel, and charge the credit card provided, if not fraudulent, and notify the customer of their travel details. If the credit card's fraudulent, you want to cancel the booked flight and the hotel. Figure 1.2 shows a process diagram of the travel agency booking section of the project. It's a fully implemented business process once the execution details have been added to each task, such as the flight booking service details, the hotel booking service details, the flight cancellation service details, the hotel cancellation service details, the credit card payment details, the e-mail details needed to notify the customer of their travel arrangements, and sorting out the details for cancellation services before they're signaled to undo a booking.

compensate@service v.1.0 type=colltr@agencyproject.com@compensation@service



**Figure 1.2** A process that captures a piece of business value; registering a booking for a hotel and a flight by taking payment from the provided credit card before notifying the buyer or determining that the payment was fraudulent and triggering a roll back of the bookings.

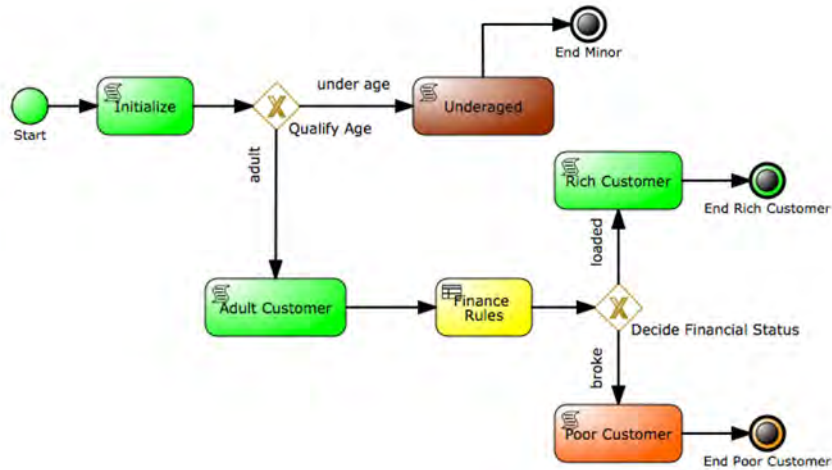
The ideal process is a fully-automated one that removes human involvement. This is a process without any user tasks and it's referred to as Straight Through Processing (STP).

By capturing a process in a static diagram and removing all human interaction you ensure that each instance completes in a consistent manner. Any ad-hoc activities that are part of human nature, like taking a coffee break or visiting with a colleague, are no longer occurring when it's captured as an STP process.

Figure 1.3 is an example of an STP process, where the tasks and decisions are made without human involvement from start to finish. This process always reaches one of the end nodes in the process diagram for each and every instance of the process.

Sometimes a piece of business value which needs to be captured as a process can't be fully automated. There remains human involvement to complete these types of processes, yet automating tasks improves the business enough to justify turning it into

Customer Evaluation Demo V.3 (customer@evaluation)



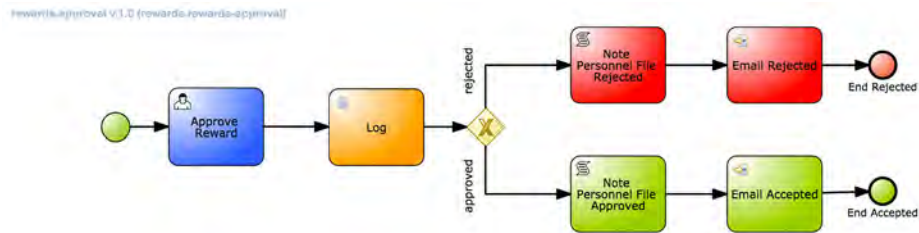
**Figure 1.3** An example of a Straight Through Process (STP), this Customer Evaluation process has only tasks and decisions from start to finish.

a process. By capturing a process that contains user tasks, you've stumbled upon an important feature of BPM which allows you to manage *wait states*. A wait state's any task that requires that you pause processing and wait for some external event to notify the process to move onwards.

In classical application delivery, it's hard to keep track of state. Waiting means putting information into storage to allow the application to be put to sleep until it's ready to continue. With BPM technology, you're provided with a state engine that manages wait states by keeping track of where you are in the process. It optimizes resource usage by releasing resources that other process instances can use until they're put to sleep. It also manages the *rehydration* of the process instance when it's ready to wake and continue from the current wait state. When a process instance reaches the wait state, it saves all state information needed to run the process instance and persist, or go to sleep and wait for something to trigger a restart. When a trigger arrives to restart a process instance, it rehydrates by gathering the necessary state data, and populates the process instance exactly as it was before the wait. It then uses the trigger provided information, such as a user task form with input data, to continue moving the process instance forward from where it stopped previously.

Figure 1.4 shows a process that uses *Approve Reward*. When a process instance starts, the first task is the user task *Approve Reward*, at which time the task's assigned to a manager and waits until a manager has time to work on the task. When this user task's reached, the process engine sets up the task, and then puts it to sleep, releasing resources for other process instances to use. This is how a single BPM process engine

can execute many, many process instances at one time; there's a small set of active process instances. Most are either in a completed state or in a wait state and not utilizing any computing resources.



**Figure 1.4** The Rewards process has a wait state in the form of a user task which is used by a manager to approve or deny a submitted employee reward. The managers decision determines the completion path to be taken, approved or rejected.

Once the task's claimed, worked on and completed, the process instance signals that it's ready to move onwards. The process engine rehydrates the process instance, putting it back into a state to move onwards with the data provided by the user task and evaluate if it should take the reject path or accept path for this particular employee reward.

Now that you've a feel for the basics of what BPM is, let's take a look at the three main elements that make up a BPM solution and need to be supported by any BPM product you might use.

## 1.2 An introduction to rules, events and processes<sup>1</sup>

The basic building blocks for any BPM project requires a product to have the ability to integrate business rules, business events and business processes. This section introduces the concept of business rules, discuss what they are, looks at how events differ from rules, and examines business processes. This approach starts with foundational building blocks that lead to the high-level business process used to tie it together.

### 1.2.1 What are business rules?

In traditional application development, you see business logic's often put into the application itself. This logic's implemented in static application code, becoming part of the artifacts that are compiled, tested and delivered into production. Each change to the logic within such an application requires a complete release cycle. Code's changed, code's compiled, it's tested, and finally delivered to production. This costs time and is susceptible to errors. Changes to any logic's passed from the business owners during requirement discovery phases, to a project team and developers, whose interpretation might differ from what was intended.

<sup>1</sup> This section comes from an [introductory presentation](#) found online.

The next time you're looking at applications in your organization, look for constructs like *if-then-statements* and *case-statements*, which are basic indicators of logic that can be extracted as business rules. These constructs are indicators of business rules that should be externalized from applications. Once such business rules are externalized you can deliver applications, and later modify the externalized business rules without needing new application code. It's now possible to put the business rules into the hands of the business owners, who understand how to maintain the life-cycle of the applications using the business rules.

Business rule management systems are designed to provide exactly this kind of support and tooling to business rules owners and application developers. Business rule management systems provide tooling to express rules in terms that the business owners understand, and allows the developer to focus on application delivery as the business owner retains visibility of the business rules serving customers. Finally, with business rules centralized in an external location, it becomes easier to maintain consistency across applications using the business rules. If business rules are spread out across multiple applications, there's a risk of duplication, and rule maintenance becomes difficult as the application landscape expands.

### **1.2.2 What are business events?**

Business rules are applied based on a condition that has to be met, and when that condition's met the rule triggers an action. Rules are evaluated one-by-one, and they're either triggered or not. Rules can be grouped together, but they're still evaluated one-by-one to determine if a rule has a condition that, if met, causes its action to be executed.

Business rules are also part of a concept called business events. Events can be triggered when a rule, or set of rules, match their conditions over a defined time period. Events that take place within the context of a business rule management system are still business rules, but now you add a *temporal* element.

For example, traditionally there are rules that can be applied to credit card transactions. Imagine a rule that requires a purchase must have a total value that fits the credit limit for that card. Should the purchase being attempted exceed that credit limit, the action taken rejects the purchase. This rule can be applied time and again, without regard for any sort of time sensitive information. It's only when you add a time element that it becomes a business event, such as looking at a period of transactions to determine if any took place in locations that aren't physically possible. Such a series of purchases, say in Tokyo, San Francisco and Amsterdam in a span of 24-hours, results in a business event triggering the blockage of the credit card, and a notification process to alert the card holder of fraudulent usage of the credit card.

A more modern example's how enterprises use business events to monitor their corporate image across all manner of social media channels. In the travel industry, for example, you see event monitoring coupled with large customer contact centers, which are manned by hundreds of employees who receive notifications whenever online comments reference their company. If they can use event monitoring to detect

and respond to messages, negative or positive, directly with the customer who wrote them, they can have a positive effect on their image in the market. This is a powerful use of business events and is only possible with a business rule management system that has event processing.

### **1.2.3 What are business processes?**

I've discussed how business processes can be discovered in an organization to reduce inefficient manual processes by automating as much as possible. What are business processes used for besides automation? They can be used to improve consistency in completing a series of tasks, increases visibility, and reduces errors.

Before an organization starts using processes to streamline their business activities, they're a large pool of employees trying to bring value to their customers. This can be done by filling orders or providing services. These employees are assigned tasks that might require interaction with back-office systems like shipping, financial, or inventory, or it might require they contact a transport company to handle order delivery.

A modern organization evolves over time, automating some interactions with back-office systems, and adding technology to provide a service layer that communicates with various applications. The problem's that these services are used by applications for specific tasks, and not linked together to handle a complete series of tasks that makes up a business process.

Once the back-office and external organizations have been put behind an automated layer of services, business process discovery can identify the processes to be automated. Business processes become the layer of organization or integration that brings a series of identified tasks to complete a part of the business. This can sometimes involve human interaction which are referred to as user tasks. By managing user tasks in your business processes, they're repeatable, can be measured for efficiency, and reduce human errors. Finally, an overview of business activities can be monitored and reports generated to keep track of how various parts of the organization function. This can lead to quicker decisions around adapting existing processes, or implementing new ones, to further accelerate the earning potential of the organization.

To review, it starts with experts from the business helping to identify needed tasks and the sequence in which they're to be completed. A small group of human resource employees could be used to discuss the hiring process. They'd tell their stories about how they put together a job description, place advertisements for the job opening, handle incoming reactions from applicants, schedule interviews using existing calendaring systems for employees selected to interview candidates, gather interview impressions, obtain a decision from the hiring owner of the job, inform rejected candidates, notify the hired candidate, begin the onboarding, etc. Portions of this process are worth automating, like the onboarding process that uses manual tasks, when automation saves valuable time and resources.

A BPM suite's used to automate the process by integrating services in the organization, directly with systems, or managing human interaction with tasks users need to

complete. The BPM suite captures the process instance data and generates reports to provide business owners up to date information and visibility into every aspect of business operations.

An often-asked question's when *not to use* a BPM suite. Although there are many cases that can be made to fit into a BPM type of solution, sometimes the complexity of a process lies in the judgements of human interactions. For example, a process to onboard new hires is a good candidate, but the decision process of who to hire after interviewing candidates requires a human decision process that you don't want to automate. This is a process which isn't a good candidate nor should it be.

When looking at BRMS products, note that they support rules and events. A BPM suite product needs to be a super-set by encompassing BRMS functionality and adding in support for process development and execution. Therefore, when talking about a BPM suite, you're referring to rules, events, and processes in a single suite.

### **1.3 Understanding the role of community projects**

When looking at Open Source software solutions it's important to understand how the products in that market are created, maintained, and from where they originate. This is no different when looking at JBoss BPM solutions, as the entire portfolio of Red Hat JBoss products is based on Open Source software.

Before a product can be created, there are community projects that are *upstream*. Upstream's where research and development takes place, where new features are tested in an open community of coders. These coders can be employed by companies, work on projects that use community code and want to contribute back fixes and findings they encounter in daily use. Some members of these community projects are only interested in rules, events and processes, and they use these projects to explore ideas that interest them.

Whatever their reasons are, there's an ever-shifting group of developers working on projects that make up the community known as Drools (<http://www.drools.org/>) and jBPM (<http://www.jbpm.org>). The Drools and jBPM projects are the foundations for rules, events, planning, processes, and tooling that can be found co-hosted on GitHub (<https://github.com/kiegroup>) for anyone to use and explore. Where the Drools project's focused on business rules and events technology, jBPM is focused on business process management technology. Both are released under the Apache License 2.0, a common Open Source license offered by the Apache Software Foundation that gives users freedom to use the software for any purpose, to distribute it, to modify it, and to distribute modified versions of the software, under the terms of the license, without concern for royalties (<http://www.apache.org/licenses/LICENSE-2.0>). The user's only required to preserve the copyright, notice and disclaimer.

Let's take a look at a few of the projects which are found in the Drools and jBPM communities to get an idea of some of the work available. Projects are listed for maintaining websites, specific tools to support users in their tasks, core functionality or engines, that provide developers with application programming interfaces (API's), and special projects that provide functionality for users.

Here are a few selected projects that give you an idea of what's available:

- *drools-website* (<https://github.com/kiegroup/drools-website>) – project used to generate the Drools project website.
- *jbpm-designer* (<https://github.com/kiegroup/jbpm-designer>) – project for the web-based process designer found in jBPM web console.
- *jbpm* (<https://github.com/kiegroup/jbpm>) – main project for core jBPM engine.
- *droolsjbpm-knowledge* (<https://github.com/kiegroup/droolsjbpm-knowledge>) – project for the common API for Drools and jBPM.
- *jbpmmigration* (<https://github.com/kiegroup/jbpmmigration>) – small side project for migration from jBPM 3.x jPDL process format to standard BPMN.

The community has many more projects, and you're encouraged to explore and keep an eye on the work done there to see what the future might hold for the JBoss BPM Suite.

A final question often posed is why not use the jBPM community projects instead of a product like JBoss BPM Suite. Although it's easy to obtain community versions of these projects, it's not easy to integrate and maintain them in your organization's architecture. I mentioned that there are many individual projects taken from the community by Red Hat and then tested, integrated, quality assured and more before they present it with a new release of JBoss BPM Suite. This has the extra advantage of JBoss BPM Suite being able to work seamlessly with other products in the JBoss portfolio.

Again, community projects seem readily usable, but they're often not easy to integrate, maintain, keep updated and secured over time. Many organizations choose to have a supported version of a product where they can rely on the stability offered and service level agreements to ensure their projects run well in production. Remember, community's about innovation and fast moving changes to the projects found there as new features, engines and theories are tested on the fly. In a product, you're less interested in the risks involved in running community versions and more interested in a solid platform for developing and running your applications.

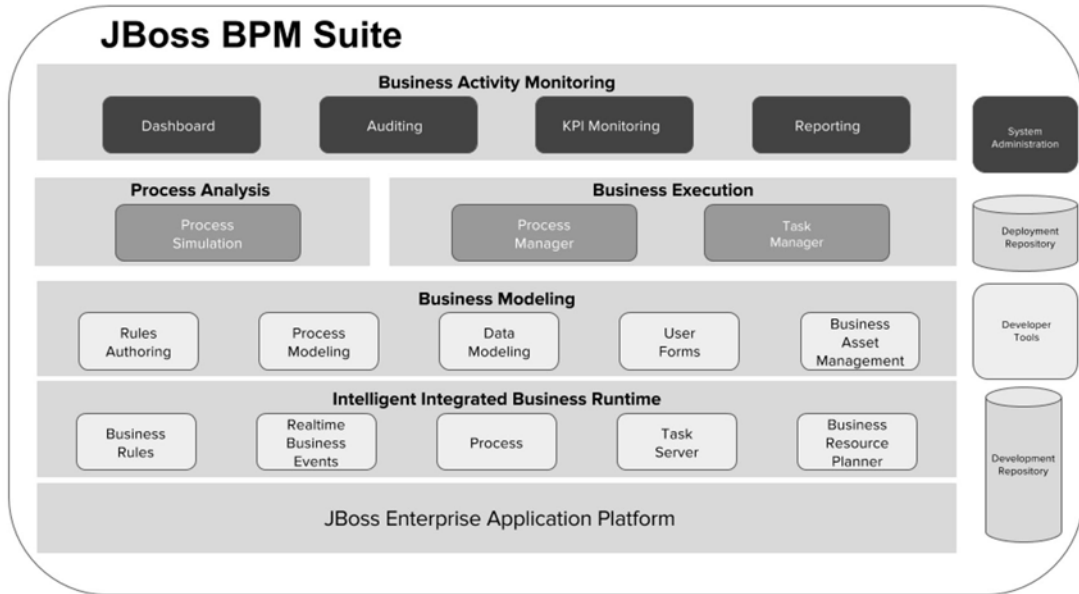
## **1.4 Meet the JBoss BPM Suite**

In this book, the focus is on the JBoss BPM Suite product. JBoss BPM Suite provides you with all the engines, tools, testing and execution environments needed to develop and deploy projects containing rules, events and processes. It brings together, in one easy to use web console, the designers, modelers, reports and other information that reduces the complexity of having to install each one individually. It covers the needs of process project teams, and allows for good visibility into the activities of team members as they work on project artifacts. It attempts to make common rules, events and process tasks easier to accomplish, and with less code development than if done without using JBoss BPM Suite.



JBoss BPM Suite's a collection of components made up of projects found in the Drools and jBPM communities. Specific projects are hard to list as certain features are sometimes deemed not ready for inclusion in a supported product. These almost-ready features are sometimes released as *technical previews* and in later releases become fully-supported features.

The components that are found in JBoss BPM Suite are shown in figure 1.5, an architectural overview of the suite layers. Starting at the bottom, I describe the components shown in each architectural layer.



**Figure 1.5** JBoss BPM Suite component architecture shows the layers that make up a BPM suite. From engines, modeling tools, execution monitoring, analysis tools and reporting, the JBoss BPM Suite has everything you need to design, develop, run and manage your business processes.

In the first layer, this architecture shows JBoss BPM Suite running on the JBoss Enterprise Application Platform (EAP). This is a certified configuration, meaning a configuration the Red Hat supports and recommends you use with JBoss BPM Suite. A true spirit of Open Source and interoperability can be found with JBoss BPM Suite, and support outside of the certified JBoss EAP platform's based on the use of certain Java Virtual Machine (JVM) configurations. For more on certified and supported configurations see the product documentation (<https://access.redhat.com/articles/704703>).

### 1.4.1 *Introducing the core runtime engines*

The next layer's called the *Intelligent Integrated Business Runtime* and contains the engines that support the core aspects of the suite:

- Business rules engine
- Business events engine
- Process engine
- User task engine
- Business Resource Planner engine

The business rules engine in this layer's responsible for rule parsing, evaluation, and execution. It's called by the application code and it's supplied with the gathered data, and the rule set used to test the data. If data matches a rule it triggers predefined actions. For example, a person's age's passed to a validation rule set that determines if the person's a minor or an adult. If the data puts the age under eighteen years old, the person's labeled a minor, otherwise they're labeled an adult. A second way to use a rule engine's through a business process. A business process can have a rule task, which is supplied with details for calling a rule set with data from the process. Often the outcome of the rule's used in a decision later in the process to take a specific path over another.

A business events engine's used to keep track of business events, the temporal aspects, as they occur and trigger rule executions as needed. It's used by the same applications when they're setup to monitor events. It must watch a flow of events for any matches with a set of rules within a given time period as defined. When an event's triggered, the event engine uses the rules engine to execute the defined actions.

The process engine in this layer manages the state of a process, keeping track of which task a process instance's in, and the data involved in that process instance, managing resources by releasing on wait states and rehydration of a process instance when needed. It also manages interactions with core rule engines, task servers, services and more. The process engine's the runtime integration component at the heart of a BPM suite.

A separate engine's used to manage tasks. These are the user tasks that require forms to display data, and to provide fields for input by users in order to complete the defined task. This engine's used to parse the form templates generated from user tasks defined in processes. These form templates can be embedded into other applications and be tied to the task engine for processing.

The final engine's the Business Resource Planner, an engine used to determine the best way to allocate resources, to plan a roster, to determine the most effective route to travel, to optimize bin packing, and much more. It works with a limited set of constrained resources, such as employees, assets, time and money, to do more business with less resources.

### **1.4.2 Modeling tools for all your BPM needs**

To facilitate business users and developers with their rule, event, and process designing, there are several business modeling tools that allow guided help in the creation of artifacts the above runtime engines use for process execution. These modeling tools are integrated into a central web interface known as Business Central. They're available in

the developer tooling that Red Hat offers as an integrated solution called JBoss Developer Studio (<https://www.redhat.com/en/technologies/jboss-middleware/developer-studio>). It's based on the popular Open Source Integrated Development Environment (IDE) called Eclipse (<https://eclipse.org>).

Various types of rules exist such as:

- guided rules
- technical rules
- decision tables
- score cards
- guided decision tables
- guided score cards

Each of these has a modeling tool provided in the web interface to make the development of specific rule artifacts easier than coding them by hand. Most of the modelers guide the rule developer through the steps needed to design and implement specific rule types. For example, the guided rule modeler, as shown in figure 1.6, has an interface to guide the developer from data imports, to the layout of required conditions that trigger actions. The modeler also has views that allows you to inspect the source code generated by the rule being designed and allows the rules to be stored in the development repository as it displays change history. It gives the developer a chance to select a past version and rewind to that point in time of developing the rule. Various rule modelers are discussed in more detail in later chapters.

Free Shipping Promotion DSL.rdslr - Guided Rules

EXTENDS None selected

WHEN

1. If customer spends \$ 75

THEN

1. Apply Free Shipping  
(show options...)

**Figure 1.6** An example of the Guided Rule Modeler where a developer's guided through designing a rule to apply free shipping (shipping value set to zero dollars) if the total value of the customer purchases exceeds seventy-five dollars.

The process modeler's a rich web-based modeler that provides the process developer with basic and advanced elements for process design. It uses drag and drop from a list of tasks, including property editors for each task type. This provides further detail for integration with back-end systems, and provides features covered in later chapters.

A data modeler provides developers the ability to define data objects that are part of the process. It provides a simple to use front-end, and allows the user to view the generated Java source code. This modeler's covered in later chapters.

The user form modeler gives the developer a drag-and-drop based tool where task forms are created or generated task forms are modified. It manages data imports to bring in models that tie fields in a form to data passed between user tasks and the form. It has lists of HTML based elements to enhance forms being designed, and includes specific property editors to allow for customization of elements. Form design's covered in a later chapter.

An asset manager provides help with importing external data models, and managing assets that make up a process project. This also includes deployment management at runtime as well as development assets.

### **1.4.3 Looking at the BPM analysis tooling**

Imagine a retail process for selling products online and a few user tasks that require manager approval on larger orders. Normally two employees are tasked eight hours per day to process these approvals, and this works. During the Christmas holidays the retail process is expecting a surge of product orders, and by doubling the number of employees working on the user tasks they can manage the expected increase of orders.

This is an example of how, during development, you can examine or test how a process reacts to severe loads or special situations before being put into production. It might also be desirable to examine the effects of a process change before they're put into production. The tooling provided here's found in the process designer. It allows you to set properties for each task to denote simulated costs and time aspects throughout the process. For example, the user task above takes, at most, fifteen minutes to complete, and at least five minutes. Furthermore, the task costs, on average, twenty dollars to complete based on the employee salary and task completion times. The user tasks can be adjusted to set the simulated number of work hours spent, and how many users are assigned to completing incoming tasks during the defined work hours. These properties are then used for input to calculate the time spent on each task, the cost of the tasks in the path taken through the process, to count the user tasks completed, and provide an overview of the totals in each category. The results can also be displayed in a variety of output styles, such as shown in figure 1.7.

After defining simulation details, you can set the number of process instances you want to run. Each simulation load you run gives you a chance to watch the server log output and upon completion, produces a simulation report.

Back to the retail process and hiring decisions that management think might help with the Christmas rush. The process developers and testers were able to simulate the process with two extra employees on the user task and determined three more employees were required on the user task. They were also able to determine the user tasks need to be worked on for sixteen hours a day, double the norm, for the expected work load to be processed in time for the holiday. They were able to make the right



**Figure 1.7** The simulation output report can be a bar chart, like shown here, or changed into one of several other types to give a visual representation of the simulation data.

hiring decisions and adjust the process workflow by extending the user task working hours based on valid test data provided by the process simulation tools.

#### **1.4.4 Execution management made easy**

After completing process development and deploying the process project, there are a few tools integrated into JBoss BPM Suite that help start and run through a process instance. A process manager provides a list of available process deployments from which you can select one to start. To start a process, process data must be submitted, and the task manager component provides a user form for entering data. The task manager keeps track of any user tasks that might be in the process, providing tasks lists and their status for users to claim, work on, and complete during the process instance lifecycle.

#### **1.4.5 Providing the necessary reporting and monitoring tools**

A key element of any BPM suite's the ability to monitor and generate reports based on what's happened during process execution. JBoss BPM Suite has an extensive set of tooling to provide Business Activity Monitoring (BAM). Out of the box there are reports designed to show historical data on process and task execution. Along with these reports there's a dashboard that allow reports to be designed based on any data source, both internal and external to the BPM suite. The dashboard modeler allows drag-and-drop creation of web based reports, allows for business owners to measure based on their own Key Performance Indicators (KPI) and ensure their process audit trails are shown according to their own business defined needs. More on this component's available in chapter 9.

### 1.4.6 The supporting components

Several components shown in figure 1.5 support the development, deployment, and execution of BPM projects. These are repositories, developer IDE tooling and system administration components that round out the suite and are based on industry wide standard components. The system administration tooling's used to configure various aspects behind the scenes when using the suite, such as:

- Users
- Roles
- repository locations
- and more

The development repository's where the BPM project artifacts reside and is Git (<https://git-scm.com>) based. The business central web console integrates the various modelers with the developer repository to preserve versions and history, as does the developer tooling. The deployment repository's a Maven (<https://maven.apache.org>) repository where the project build's deployed and can be tied into standard enterprise continuous build systems. Several programmable interfaces (APIs) are available, but these aren't covered here, instead see the product documentation.

## 1.5 Summary

- Business process management (BPM) orchestrates all manner of systems, people and services, while allowing you to structure the timeline of their execution.
- BPM starts with a process, where a piece of business value's automated either fully or partially to improve the business.
- A fully automated business process is referred to as Straight Through Processing (STP).
- When unable to fully automate a business process, there's a need to deal with wait states, such as human interaction through user tasks.
- Capturing and maintaining state in a process is a main characteristic of BPM that ensures efficient use of resources.
- Some processes aren't possible to automate, identified by the need for pure human intelligence for completion.
- Reporting and monitoring provides insights into the current and past conditions of processes, giving business owners insights into process improvements that might help their business.
- Rules, events and processes are the key building blocks for BPM projects.
- Rules are often found embedded in applications, making it difficult and costly to update. This is alleviated by using a business rules engine to capture and externalize business logic for application to call when needed.
- Business events are rules with a temporal element.

- Open Source products like JBoss BPM Suite use upstream community projects like Drools and jBPM.
- These community projects are created and worked on by a vast array of coders to deliver functionality that can be used together. It's often difficult to tie together all the components in the community to achieve a BPM suite.
- JBoss BPM Suite's a product from Red Hat that brings together a set of the community projects that are supported with official service level agreements.
- Various components make up the suite, such as runtimes, modeling tools, simulation tools, process and task execution tooling, business activity monitoring tooling and the background repositories.

# Processing first steps

---

## **This chapter covers**

- Installing JBoss BPM
- Starting a first project
- Touring JBoss BPM in the Cloud

As with any new technology, you can't wait to get your hands on your first installation of JBoss BPM Suite and start your first process project. You might have dug into the product documentation online and seen that the installation process requires a few steps to be done manually. Care needs to be taken on the order that these steps are done in and you must make sure the default settings are what you need for your project. If the default settings aren't what you need, then you can dig deeper to see what you need to do to change these default settings.

In this chapter, you're provided with an easy-to-use installation project that allows starting a project with JBoss BPM in minutes. This setup has been used in the workshops I provide online<sup>1</sup> and is an effective tool for many new users of JBoss BPM. You're installing a new and ready to use JBoss BPM Suite server in minutes.

As soon as you've an installation setup, the next thing's to start your first project. You walk through the steps needed to embark on any new JBoss BPM Suite process project. By the end of this chapter you're ready to start designing your project's artifacts, such as rules, events and processes.

---

<sup>1</sup> If you're interested in this online workshop, it's freely available for you to play with online at <https://bpm-workshop.github.io/> and you can install the entire workshop locally through this repository; follow the instructions included in the project at <https://github.com/eschabell/presentation-bpmworkshop>.



To wrap up this chapter, you're taken on a tour of the possibilities of using JBoss BPM in the Cloud. It includes both a look at the OpenShift Online cloud offering and the possibility for you to experiment with containerized JBoss BPM in a local private Cloud setup that uses the Red Hat OpenShift Container Platform. I provide links to more resources to help you explore further Cloud usage, but I'm not going any deeper in this book.

## **2.1 *Installing JBoss BPM***

The easiest way to get started with a JBoss BPM Suite installation's to make use of the easy install project. This project's a standardized installation of JBoss BPM Suite project which is used in all the examples found in this book.

This section ensures that you've a working basic installation of JBoss BPM Suite and that you're ready to start your first project. It explains the various configuration decisions that the project has chosen to use, but doesn't explain all the available configuration options within the JBoss BPM Suite product. Should you be interested enough to want to investigate more options available to you, I'd suggest digging into the available product documentation found at <http://developers.redhat.com/products/bpmsuite>.

### **2.1.1 *Meet the JBoss BPM Suite Easy Install project***

The JBoss BPM Suite Easy Install project's a self-contained project available to you free online at <https://github.com/effectivebpmwithjbossbpm/chapter-3-easy-install-demo>. This project gets you up and running in minutes without the hassle of reading lots of product documentation. It also provides you with sane defaults for your initial project explorations in the world of JBoss BPM. The project supports Java 7 or Java 8 and it assumes you've one of these installed on your machine.

To get started you need to follow the project's provided readme document, which contains all the installation instructions. This chapter is going to teach you how to generate a containerized installation based on the *Docker platform*<sup>1</sup>. It isn't possible to provide an example installation project as a completed container because you can't distribute the JBoss BPM Suite or JBoss EAP products in containers. Let's look at what it takes to generate a containerized installation of JBoss BPM Suite, one which is ready for you to start fresh with your first project.

The steps to generate a containerized installation are found in the project's main readme file and are slightly different than installing locally:

- 1 Download and unzip (<https://github.com/effectivebpmwithjbossbpm/chapter-3-easy-install-demo/archive/master.zip>).
- 2 Download JBoss EAP & JBoss BPM Suite, add to installs directory (see [installs/README](#)).

---

<sup>1</sup> Docker tooling's outside the scope of this book, but you can find all you need to get started online at [https://www.docker.com/products/overview#install\\_the\\_platform](https://www.docker.com/products/overview#install_the_platform).

3 Build the demo image from the root of the project directory:

```
docker build -t effectivebpmwithjbosssbpm/chapter-3-easy-install-demo .
```

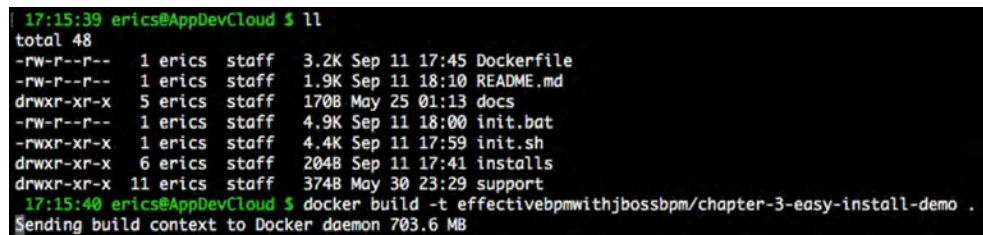
4 Start demo container:

```
docker run -it -p 8080:8080 -p 9990:9990 effectivebpmwithjbosssbpm/chapter-3-
easy-install-demo
```

5 Login to <http://localhost:8080/business-central> (u:erics / p:bpmsuite1!)

Now let's walk through this together from the start to see what a containerized installation looks like. It's always nice to have an example of how to get to the starting point where you can begin to work with JBoss BPM Suite.

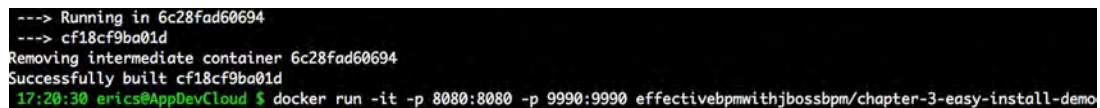
Starting at the point that you've started Docker and can enter the command found in step number three, which you see running in figure 2.1.



```
17:15:39 erics@AppDevCloud $ ll
total 48
-rw-r--r--  1 erics  staff   3.2K Sep 11 17:45 Dockerfile
-rw-r--r--  1 erics  staff   1.9K Sep 11 18:10 README.md
drwxr-xr-x  5 erics  staff  170B May 25 01:13 docs
-rw-r--r--  1 erics  staff   4.9K Sep 11 18:00 init.bat
-rwxr-xr-x  1 erics  staff   4.4K Sep 11 17:59 init.sh
drwxr-xr-x  6 erics  staff  204B Sep 11 17:41 installs
drwxr-xr-x 11 erics  staff  374B May 30 23:29 support
17:15:40 erics@AppDevCloud $ docker build -t effectivebpmwithjbosssbpm/chapter-3-easy-install-demo .
Sending build context to Docker daemon 703.6 MB
```

**Figure 2.1** Once the example project for this chapter's unzipped and products added, run the docker build command from the root of the project directory.

Once the container has been built, you can start it with the command from step number four as shown in figure 2.2. You see the container starting which looks exactly like a local machine starting up the JBoss BPM Suite.



```
---> Running in 6c28fad60694
---> cf18cf9ba01d
Removing intermediate container 6c28fad60694
Successfully built cf18cf9ba01d
17:20:30 erics@AppDevCloud $ docker run -it -p 8080:8080 -p 9990:9990 effectivebpmwithjbosssbpm/chapter-3-easy-install-demo
```

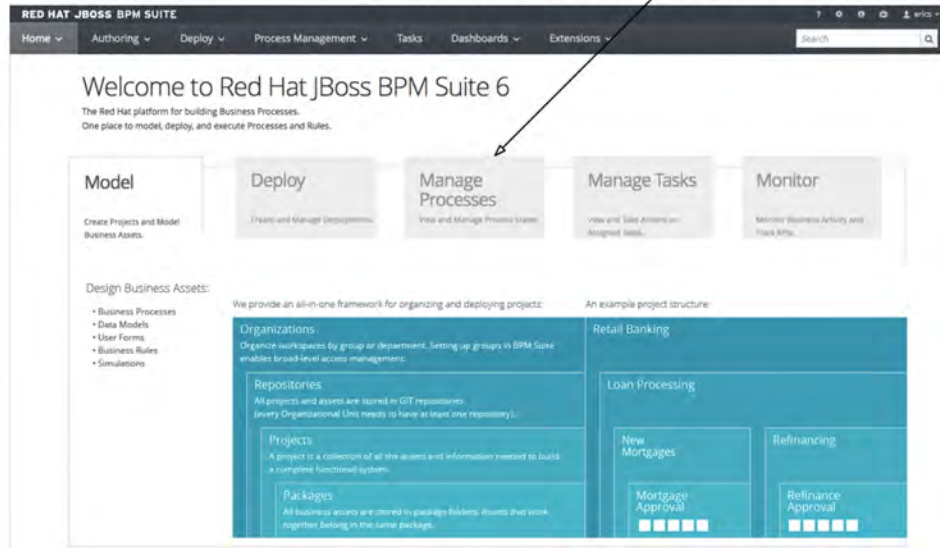
**Figure 2.2** After the container has been built, you run the container with the command shown.

To access the container installation, type it into your browser and you're given a login to JBoss BPM Suite. Visually there's no difference between running JBoss BPM Suite locally and running it in a container.

<http://localhost:8080/business-central>

Log in with user *erics* and password *bpmsuite1!* to see the home screen as shown in figure 2.3.

The documentation is available under the Model, Deploy, Manage Processes, Manage Tasks and Monitor tabs.



**Figure 2.3** After logging in to the Business Central you'll be at the home screen where you can browse the documentation. This home screen's where you start your work on BPM projects.

This completes the containerized installation of JBoss BPM Suite and you're now ready to start creating your first project. You're now able to install these example projects using container technologies.

Although JBoss BPM Suite Business Central's the collection of tools that enable you to create, manage, run and monitor everything needed for your BPM project, in this chapter you're limited to taking a path through Business Central that sets up your first project. In later chapters, you dive deeper into other areas of Business Central and the specific tools that you need to design a complete solution.

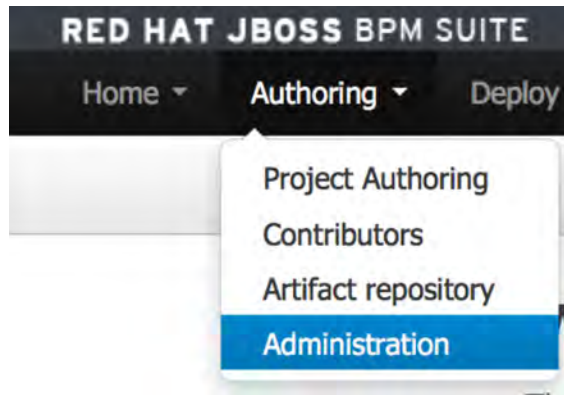
## 2.2 *Start a first project*

This section guides you through the *Administration* perspective which is where you start to get a project off the ground. It doesn't guide you through, nor explain every single aspect of the administration perspective. It isn't intended to be an exhaustive guide, but to give you a practical guide for getting hands on with a new project's organizational structure.

### 2.2.1 *Starting with Administration perspective*

The administration perspective's a view that takes you to the tools that an administrator of the JBoss BPM Suite might need. Tools exist to setup your project structure and to work with the repositories that holds your projects.

After logging into Business Central, you select the menu item *Administration* from the *Authoring* menu as shown in figure 2.4.



**Figure 2.4** The Authoring menu contains the Administration item that leads to the Administration perspective where you start your project setup.

The view you're presented with contains almost nothing, which might be expected when there are no projects defined. Let's take a closer look at how projects are organized within JBoss BPM Suite. The following structure from the top level to bottom levels look like this and is shown visually on figure 2.3:

- *Organizations* – allows you to structure high level workspaces organized by groups or departments in your enterprise.
- *Repositories* – the actual project assets are stored in a repository. Each organization must have at least one repository.
- *Projects* – a collection of all the assets and configuration information need to build a functioning system.
- *Packages* – business assets are stored in package folders and if they work together then they should be packaged together.

Next you're going to add the structure you need for your project one step at a time to get ready for a retail group process project.

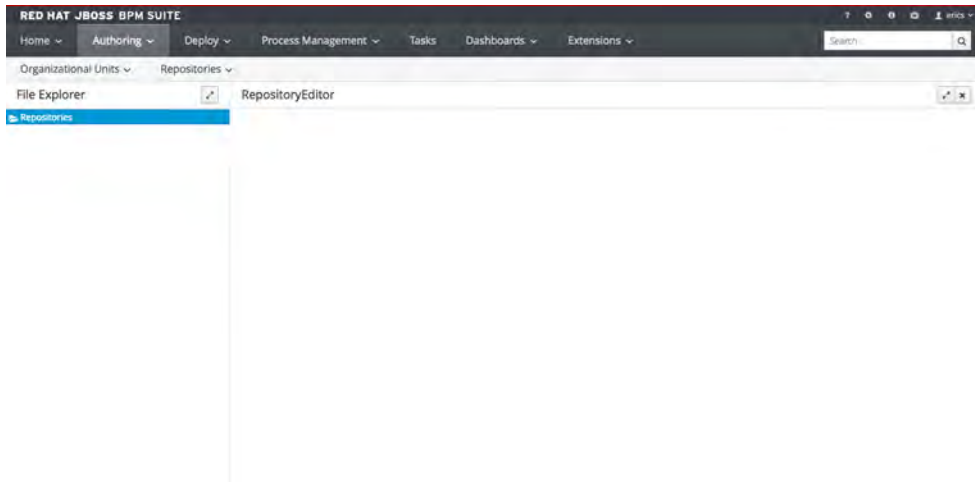
### **2.2.2 Adding an organizational unit**

Imagine you're working for a large central retail organization which is setting up process projects that eventually span the entire organization, but for now you're going to focus on human resources (HR) and the finance department (Finance). Later you're tasked with process projects relating to the remote stores, logistics and much more.

The organization structure you might setup to accommodate these first two targeted departments and projects could look like the following:

- Organizations: *Retail Group*
- Repositories: *Back Office*
- Projects: *HR, Finance*
- Packages
  - project HR: Employee Onboarding, Employee Rewards
  - project Finance: Credit Approval, Add Suppliers

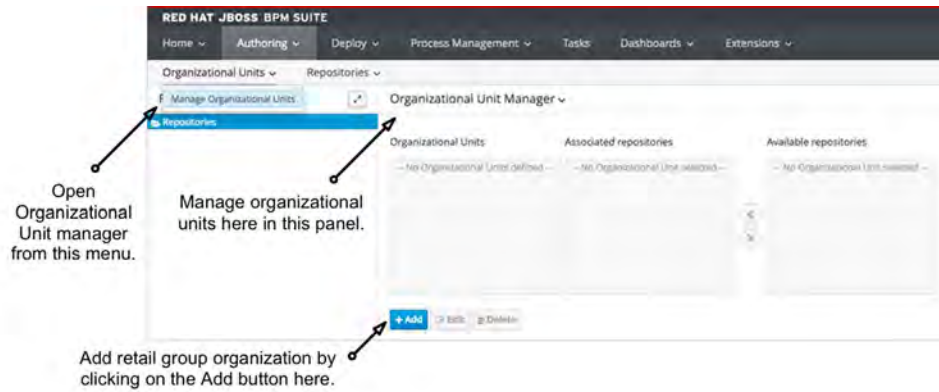
Our large retail company would be putting all BPM work under the *Retail Group* at the top organizational level. Next you'd define the first repository to be *Back Office* as you're going to tackle the internal human resources and financial processes. The projects can align with the naming of the departments, and they're put under an *HR* project and a separate *Finance* project. Finally, you define the first packages that focuses on the two project spaces. The first two processes are specific to human resources, namely *Employee Onboarding* and *Employee Rewards*. The other two belong with the financial processes and are called *Customer Credit Approval* and *New Suppliers*.



**Figure 2.5** The initial Administration perspective view's empty, nothing has been setup yet. You start by setting up your organization using the menu in the top right labeled **Organizational Units**.

The employee onboarding can be for processing all you need to get a new employee registered and working within the retail company. The employee rewards process is most likely about processing bonuses or awards given to individual employees. The credit approval process is used by the finance department for approving credit card accounts for customers at the stores in the retail group. When a new supplier's added to the logistics network of the retail group, a process is automated to setup billing and payment processing within the financial systems.

Let's setup the above structure in your new JBoss BPM Suite system. You're still logged in and looking at the administration perspective as shown in figure 2.5. First you need to setup the organization, which can be found in the menu labeled *Organizational Units*. Click on the drop-down menu item labeled *Manage Organizational Units*, which opens the *Organizational Unit Manager* and your screen should look like figure 2.6.



**Figure 2.6** In the administration perspective, you start by setting up your organizational structure for projects. Open the Organizational Unit Manager by accessing it through the menu. To add your retail group organization, you click on the Add button as shown.

Click on the *Add* button at the bottom and a pop-up appears for adding a new organizational unit. You can fill in the following information to create your new *Retail Group*:

- Name: Retail Group
- Default Group ID: *com.group.retail*
- Owner: (optional field to assign owner of the org, leave blank)

Figure 2.6 shows you what the information fields look like after you've completed the fields. You can click on the *+OK* button to finalize and add the organization. You should see *Retail Group* listed under Organization Units, with the remaining *Associated repositories* and *Available repositories* windows empty as there are no defined repositories yet. Once repositories are available, you can manage assignments to your organization here.

**Add New Organizational Unit**
✕

**Organizational Unit Information**

Name \*

Default Group ID \* i

Owner

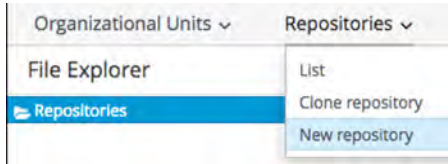
+ Ok
Cancel

**Figure 2.7** The organizational unit information pop-up provides you with the fields to add *Retail Group* and a default group ID of *com.group.retail* while leaving the owner field blank.

Now that you're done with adding an organizational unit, close the Organizational Unit Manager by clicking the X button in the top right of the editor. Next up, you add a repository for the back-office process projects.

### 2.2.3 Adding a repository

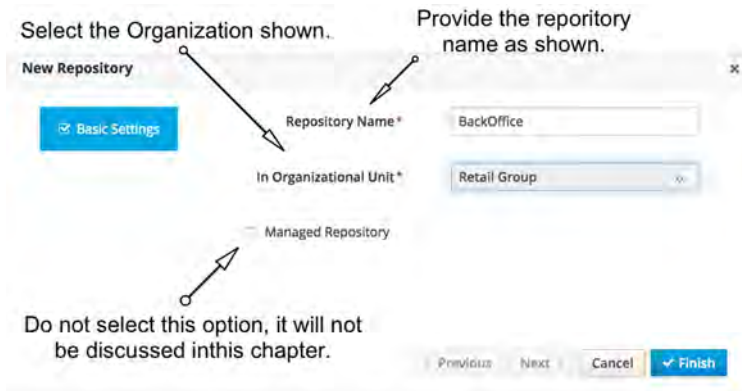
The focus of this imaginary retail group's initially on the back-office process projects centered around human resources and the finance department. Several options are available to you when you examine the *Repositories* menu as shown in figure 2.8.



**Figure 2.8** Several options are available to you when looking at the *Repositories* menu. You're creating a brand-new repository.

If you've a system setup with previously defined repositories, the *List* entry in this menu would provide an overview of all the repositories available to you. The second entry, *Clone repository*, is unfortunately a technical term related to the underlying technology used to manage a repository. Repositories are implemented in GIT (<https://git-scm.com>) and the activity by which you'd import an existing repository into JBoss BPM Suite, if using GIT commands, is called *cloning* a repository. If you'd an existing repository you could import it here with the menu entry *Clone repository*. The last entry, *New repository*, is the one you want as you need to create a brand-new repository for the retail group's back office process projects.

When you select that menu item you see a pop-up appear that asks you to fill in the information as shown in figure 2.9, namely the *BackOffice* repository name and then selecting the *Retail Group* organization from the list of organizational units. Ignore the *Managed Repository* check-box, this feature won't be discussed in this chapter. Finally, complete this action by clicking on the *Finish* button.



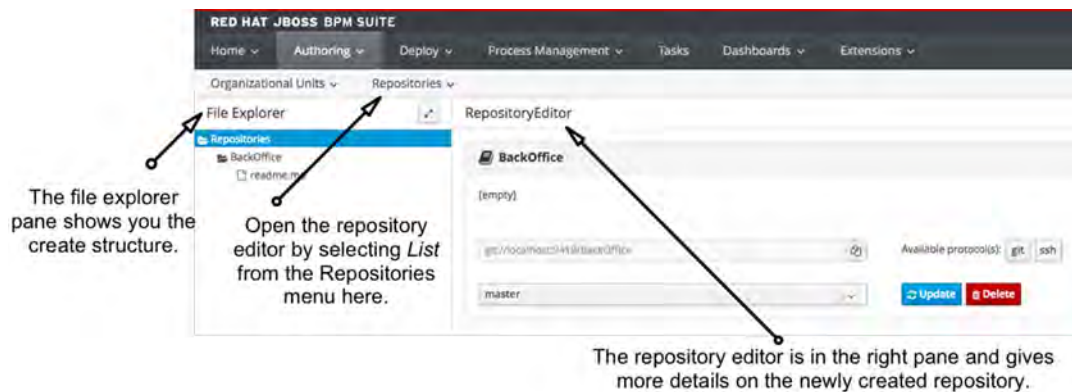
**Figure 2.9** When adding a new repository, you're shown this pop-up in which you enter a repository name and then select the organizational unit.

You should notice that a new *BackOffice* repository has appeared in the list on the left in the *File Explorer*; if not, then refresh your view. Now as with many things in the JBoss BPM Suite, there are several ways to view the details of this repository, such as:

- Click on the *Repositories* folder in the *File Explorer*
- Select from the *Repositories* menu the entry *List*

Both actions are opened in the right pane, the *Repository Editor* with your new empty *BackOffice* repository as shown in figure 2.10. Well, it isn't completely empty, there is a single *readme.md* file which is nothing but a placeholder. You might also notice that there's a single entry shown with the GIT repository address in the form of:

```
git://localhost:9418/BackOffice
```



**Figure 2.10** Once the *BackOffice* repository has been created it's visible in the left *File Explorer* pane. You can open it in the *Repository Editor* by clicking on the folder *Repositories* or by selecting the *Repositories* menu entry *List*.

This can be used by developers to check out a local copy of this repository<sup>1</sup>, shown here only to validate that a read only copy can be obtained of the repository with all artifacts contained therein. Developers want read-write access; click on the *ssh* button to show the ssh URL and clone it with:

```
$ git clone git://localhost:8001/BackOffice
```

```
The authenticity of host '[localhost]:8001 (:::1]:8001)' can't be
established.
```

```
DSA key fingerprint is SHA256:ok9uks2j16tEHxI9y2I13e8QhkXwIuLS0MytwfxEOo0.
Are you sure you want to continue connecting (yes/no)? yes
```

```
Warning: Permanently added '[localhost]:8001' (DSA) to the list of known
hosts.
```

<sup>1</sup> Here you're shown how to obtain a read-only version of this project's repository. For more details on how to obtain a read-write copy of a project's repository as a developer might desire to do see <http://www.schabell.org/2014/02/redhat-jboss-bpmsuite-access-git-using-ssh.html>.



```

Password authentication
Password: bpmsuite1!

Cloning into 'BackOffice'...
remote: Counting objects: 3, done
remote: Finding sources: 100% (3/3)
remote: Getting sizes: 100% (2/2)
remote: Compressing objects: 100% (116/116)
remote: Total 3 (delta 0), reused 0 (delta 0)
Receiving objects: 100% (3/3), done.
Checking connectivity... done.

$ ls -l BackOffice/

readme.md

```

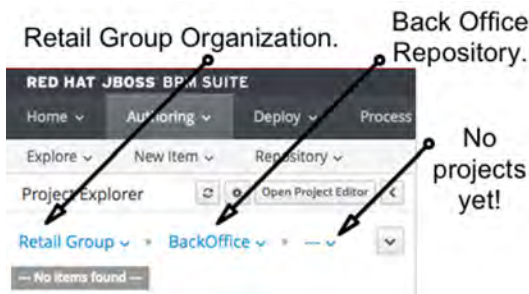
As you can see, the only artifact available's the readme file. I won't go into any more details on how developers can interact with this repository, but the focus remains how to interact through the Business Central interfaces. Next up, your task's adding projects for human resources and the financial department.

### 2.2.4 Adding new projects

If you remember, you've put together the Retail Group organization and Back Office repository. Now you create two projects for the human resource and financial departments.

Let's switch over to the *Project Authoring* perspective which you find in the *Authoring* menu at the top, remember? Figure 2.11 shows you the *Project Explorer*, the left panel on your screen where you see the menu structure in the form of:

ORGANIZATIONS - REPOSITORIES - PROJECTS

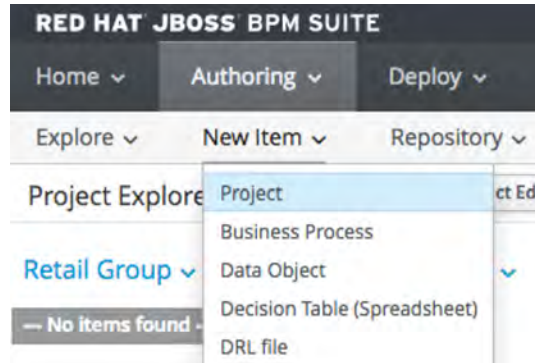


**Figure 2.11** After using the *Authoring* menu and selecting the *Project Authoring* perspective, you're shown the *Project Explorer* with your structure.

The menu shows one of the available organizations, repositories and projects. In your case having only create one organization, it's selected by default. With one repository, it's selected too. No projects have been created yet, and it's empty.

Should you have more organization, repositories and projects you'd be able to select your choices in these drop-down menus. At this point it's time to add your projects for the human resources and financial departments.

The *New Item* menu's where you can create any type of asset you might want to use in your project, but first you need a project. Therefore, in figure 2.12, all other available assets are greyed out and you can't select them. The only option available's the *Project* item.



**Figure 2.12** The *New Item* menu's used to add any of the assets you find listed. As you currently don't have a project defined, all the assets except *Project* are greyed out and unavailable. Select the *Project* to reach a pop-up labeled *Create new Project* where you can enter a project name.

Click on the *Project* entry to open a pop-up labeled *New Project*. Enter a project name and the rest of the project details as shown in figure 2.13. It contains the project name, a description and some details which are directly related to the fact that the

**Figure 2.13** This is the *New Project* pop-up where you can fill in all the information and details for your new project. Some fields are pre-filled based on previously entered information, and others can be filled or modified as desired. The last three are related to JBoss BPM Suite using Maven as a project build tool behind the scenes. These three fields are generated based on data previously entered about the project and you can either accept these defaults or modify any of the fields as you desire.

build process behind the scenes is based on Maven (<https://maven.apache.org>). The only thing you need to know is that projects which use Maven are defined with a group id, an artifact id and a version number. All three of these are auto-filled for you based on previously entered data and you can either accept these defaults or adjust as you desire before continuing by clicking on the *Finish* button at the bottom right.

This creates the project and open the *Project* editor in the right pane on your screen as shown in figure 2.14.

The screenshot shows the Project Editor interface. At the top, the project path is displayed as 'Project: [HREmployeeOnboarding:com.group.retail:1.0]'. Below this, there are several sections for configuring the project:

- Project Settings:** A dropdown menu currently set to 'Project General Settings'.
- Project General Settings:**
  - Project Name:** 'HR Employee Onboarding'
  - Project Description:** 'Human resources employee on-boarding process.'
- Group artifact version:**
  - Group ID:** 'com.group.retail' (with an example: 'com.myorganization.myprojects')
  - Artifact ID:** 'HREmployeeOnboarding' (with an example: 'MyProject')
  - Version:** '1.0' (with an example: '1.0.0')

**Figure 2.14** After creating a project, the Project Editor's displayed with project details.

Now you're going to create the three other projects by selecting from the *New Item* menu *Project* and filling in each of the following project names. The details for the *Create new Project* I leave up to you to come up with yourself:

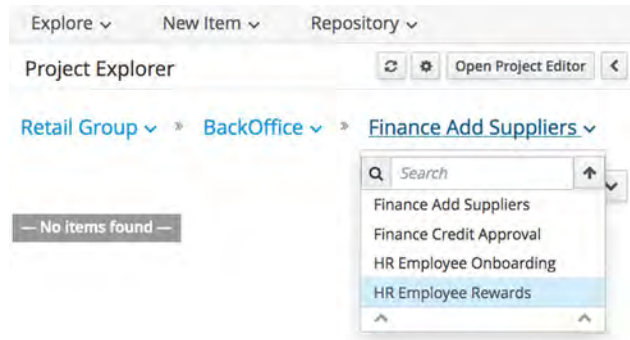
- HR Employee Rewards
- Finance Credit Approval
- Finance Add Suppliers

Did you notice that after the first project was added that all the new item asset types that were previously grey colored are now available for creation? This is because once a project's defined you can begin creating rules, events, models and processes.

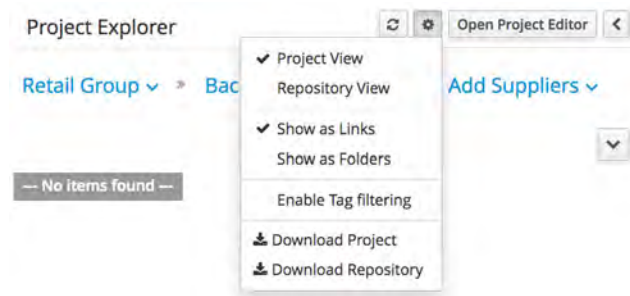
Your project's drop-down menu in the *Project Explorer* should look something like figure 2.15. All four projects are available for you to choose from and if you select one it opens that project's *Project* editor in the right pane for viewing along with all the assets, if any, in the left *Project Explorer* pane.

Did you notice that within the *Project Explorer* there are a few more options besides the organizations, repositories and project's drop-down menus? A small button's at the top in the middle of three with a sort of gear icon.

Start by putting your mouse pointer on it to expose the pop-up text labeling this button as *Customize view*. By clicking on it you expose the menu shown in figure 2.16 where you can determine how you want to view to look in the Project Explorer.



**Figure 2.15** From within the Project Explorer you can view all available projects from the drop-down menu. Here you see all four of the projects you've created. If you select one it opens, showing the Project editor in the right pane and the project assets listed in the Project Explorer pane on the left.

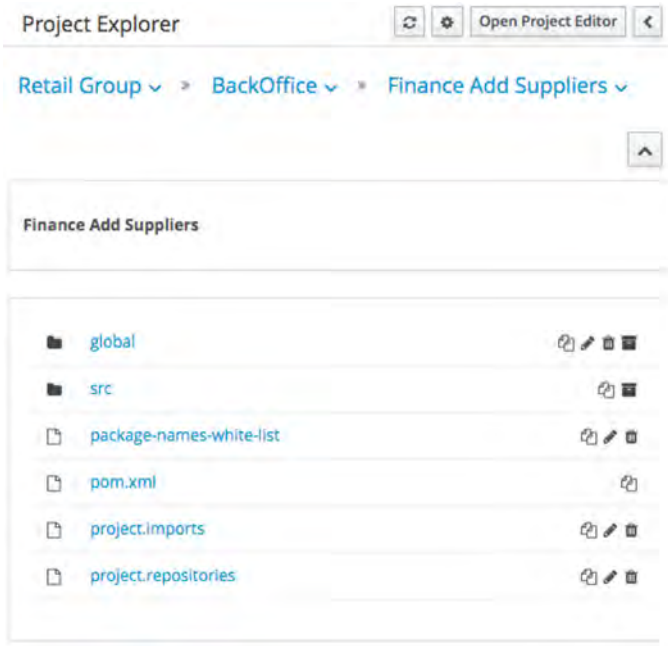


**Figure 2.16** The Customize View button produces a menu where you can adjust the view and how a project's shown. The Project View's the default, and the Repository View exposes project files as if you're viewing the file system.

By default, it's set to *Project view*, which means you're shown project assets in the format of categories with drop down menus. Right now, the only example you have's the category of *Work Item Definitions*, which drops down into a single item. If you select *Repository view*, it changes your view to that of all the files in the project as if you're looking directly at the file system as shown in figure 2.17.

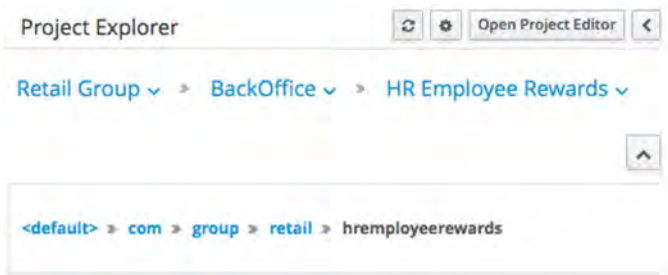
The *Project view* is meant to provide a more focused view of the BPM project assets that you can manage through the provided Business Central tooling. You can guess that the developer role in your team might feel better working with the repository view, and a process analyst and architect are comfortable with an asset focused project view.

The next two items in the customize view menu shown in figure 2.16 are *Show as Links* (the default) and *Show as Folders*. Selecting them appears to change nothing, as



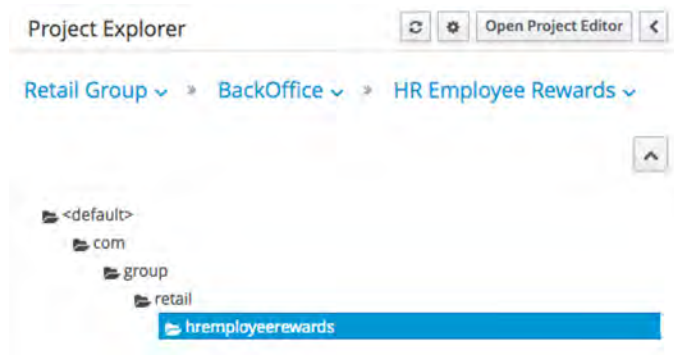
**Figure 2.17** This view in the Project Explorer's called *Repository view* and exposes all files in a project as if you're viewing the file system.

this is related to the view you first need to expand with the small '+' button next to your projects menu in the Project Explorer. If you click on this plus button it opens the currently selected project and shows you a linked view of the folder structure. For example, in figure 2.18 you see that I've the project *HR Employee Rewards* selected and I've already clicked on the plus button to expand the folder view. Clicking on the folders *com – group – retail – hremployeerewards*, in that order, exposes a linked view of the project.



**Figure 2.18** This is your view of a project in the Project Explorer after you configure your view to show the HR Employee Rewards project as links. You can click on the project structure from *com – group – retail – hremployeerewards* to expand the linked view down into the project structure. Any artifacts in a specific folder are displayed in the Project Explorer when you reach that depth. This image shows the deepest level of the project have selected.

If you click on the gear icon to customize your view, select the *Show as Folders*. This alters how folders are shown to better resemble your file system. Now you can click on each folder down into *hremployeerewards* until it resembles figure 2.19.



**Figure 2.19** This is your view of a project in the Project Explorer after you configure your view to show the HR Employee Rewards project as folders. You can click on the project structure from *com – group – retail – hremployeerewards* to expand the folder view down into the project structure. Any artifacts in a specific folder are shown in the Project Explorer when you reach that depth. This image shows the deepest level of the project selected.

In the customize view menu you can also *Enable Tag filtering*, which should allow you to filter the views by certain tags you can later assign to project assets. I won't demonstrate this but you're welcome to explore this later when you encounter the option to tag any project asset.

You can also download your project or repository, the ones selected in the drop-down menu in your Project Explorer, by selecting the menu items *Download Project* or *Download Repository* respectively. Depending on your browser settings you might be asked where to save the download or it might automatically save the file. These files are Zip archives of the requested repository or project.

The left button next to the configuration view gear icon is the *Refresh* button, a sort of circular icon used to refresh the static view of the Project Explorer at any time. This can be necessary from time to time as you're using a browser based view which isn't always able to detect changes to the view as fast as you'd like; click on the refresh button to pick up changes.

A button labeled *Open Project Editor* opens the project details in the panel on the right, as you've already seen. This completes your setup of the Retail Group's repositories and projects, and you're now ready to start designing the process artifacts that are part of the project solution.

### **2.2.5 Where are the packages?**

The sharp reader might have noticed that the concept of organizations, repositories, projects and packages has been covered in the previous sections. You explored in depth examples of setting up a project that included the organization, the repository and several projects.

What happened to the concept of packages and where are you going to learn about them? This is a concept that you don't use to define a project from the start, it evolves as the project assets are designed and created. For example, when designing rule assets, you can group these rules in packages. The product documentation ([https://access.redhat.com/documentation/en-US/Red\\_Hat\\_JBoss\\_BPM\\_Suite](https://access.redhat.com/documentation/en-US/Red_Hat_JBoss_BPM_Suite)) talks about this.

*A package is a collection of rules and other related constructs, such as imports and globals. The package members are typically related to each other, such as HR rules. A package represents a namespace, which ideally's kept unique for a given grouping of rules. The package name's the namespace, and isn't related to files or folders in any way.*

You can find references to packages, where artifacts are grouped within projects into packages. This is visible in the Project Explorer where you encounter the packages such as *Work Item Handlers* or *Business Processes*; menu items where assets that work together are grouped. This isn't the same stringent structure as the above package namespaces are for rules. It can be a confusing concept within JBoss BPM Suite as it depends on whether you're looking at packages as a developer or at as a higher-level business concept like architects or process analysts would.

Now you've covered all the concepts that you need to understand for setting up your first project, it's time to take a tour of the options available to you for exploring JBoss BPM in the Cloud.

## **2.3 Touring JBoss BPM in the Cloud**

This section provides a short tour of JBoss BPM as it can be used in the Cloud. It points you to materials that you can explore further at your convenience. You're provided informational sites and example demo collections, but this topic's too extensive to dive into deeply here. You won't be walking you through any of the example projects or diving deeply into the technical details of how you can deploy your JBoss BPM applications into the Cloud. I leave this to you to explore at your leisure.

### **2.3.1 What exactly is JBoss BPM in the Cloud?**

The concept of JBoss BPM in this book has been to focus on local installations and interaction through Business Central in your browser. An astute reader can easily imagine that when working in a browser, it doesn't matter where your JBoss BPM Suite's hosted.

The first experience you can create's to have JBoss BPM Suite running in the Cloud and available to you anywhere instead of hosting it locally. Another's to deploy your solutions into the Cloud for others to use. At the time of this writing there are a few solutions which are officially supported with more scheduled to be delivered into

the market soon. You can follow the availability and learn more at the OpenShift xPaaS topic (<https://www.redhat.com/en/about/blog/xpaas>).

Let's look at getting the same experience in the OpenShift Cloud with JBoss BPM Suite as you've seen in this book.

### 2.3.2 Getting the Cloud experience

To get a basic OpenShift Cloud setup, you'd normally need to have access to the Red Hat Customer Portal (<https://access.redhat.com>) where you could download the OpenShift components for installation in a data center somewhere. Luckily, there's also a way to obtain these components for free at the Red Hat Developers site (<http://developers.redhat.com>) and install them locally for the same Cloud experience.

To make this entire experience easier, I've setup an organization called the Red Hat Demo Central (<http://github.com/redhatdemocentral>) where you can browse a collection of easy-to-use JBoss example projects that run on the OpenShift Cloud. The first one you need to start with is the OpenShift Container Platform (OCP) installation demo. The OCP is meant to provide you with a ready to use local installation of OpenShift. Once it's installed you're ready to start developing your applications for deployment into the Cloud.

After you've installed the OCP project, there's a version of the JBoss BPM easy installation that can be used to install JBoss BPM Suite into your personal OpenShift Cloud. This enables you to log in to the Business Central console and start working on your BPM projects as explained to you in this chapter.

I want to provide you with a few steps in the format of a plan that you can follow should you be interested in trying this out. This plan only covers high level steps you need to take; refer to the individual project readme files for detailed steps.<sup>1</sup>

- 1 Install the *OpenShift Container Platform Install Demo* from project named *ocp-install-demo* (<https://github.com/redhatdemocentral/ocp-install-demo>)
- 2 Install the *App Dev Cloud with JBoss BPM Suite Install Demo* from project named *rhcs-bpms-install-demo* (<https://github.com/redhatdemocentral/rhcs-bpms-install-demo>) *being sure to pay attention to the OCP installation instructions*
- 3 Log in to Business Central as instructed in the installation directions and begin to develop BPM process projects

Now that you've a basic installation in the OpenShift Cloud I provided, you can move on to explore more of the example projects hosted at Red Hat Demo Central. Let's take a closer look at one of these and provide you with the plan to get it installed.

### 2.3.3 Installing the JBoss BPM Travel Agency in the Cloud

After installing your Cloud through the CP install demo project, you've been given the plan to install the JBoss BPM Suite to start with your BPM projects.

---

<sup>1</sup> Everything mentioned in the steps listed here can be found in the Red Hat Demo Central at <http://github.com/redhatdemocentral>.



The JBoss BPM Travel Agency isn't only available for local installation, but also for use on the OpenShift Cloud. The following high-level steps are needed to complete a full installation in the OpenShift Cloud, see the individual project readme files for detailed instruction.

- 1 Install the OCP Install Demo from project named ocp-install-demo (<https://github.com/redhatdemocentral/ocp-install-demo>)
- 2 Install the App Dev Cloud with JBoss Travel Agency Demo from project named rhcs-travel-agency-demo (<https://github.com/redhatdemocentral/rhcs-travel-agency-demo>)
- 3 Log in to Business Central as instructed in installation directions and you can explore the JBoss Travel Agency project

This is but one example project, there are many more you can explore in the Red Hat Demo Central collection that showcases BPM application development in a Cloud stack.

## **2.4 Summary**

- Installing JBoss BPM Suite can be done easily on a machine to get started with BPM process projects.
- Containerized installation of JBoss BPM Suite's as easy as installing locally and the container can run on any supporting container engine.
- Before embarking on a first BPM project it's necessary to define organizations, repositories, projects and packages.
- Organizations allow you to structure high level workspaces organized by groups or departments in your enterprise.
- Repositories are the concept of where the project assets are to be stored in an actual repository.
- Projects are a collection of all the assets and configuration information need to build a functioning system.
- Packages contain business assets are stored in package folders and if they work together then they should be packaged together.
- JBoss BPM can be used in the OpenShift Cloud either to develop BPM applications or for deploying BPM applications.
- Multiple example BPM cloud projects can be found on Red Hat Demo Central.

# Modeling process data

---

## **This chapter covers**

- Implementing a data model in JBoss BPM
- Using JBoss BPM data modeling tool
- Examples of data modeling in JBoss BPM

One of the most important building blocks for a process project's data. It could be argued that this is a fact for any application development project. It applies even more for a process project which, by definition, are receiving, acting on, moving, manipulating and modifying data constantly during its lifecycle. This data needs to be in a form that can be easily understood by all involved with designing and building the process project. The form the data's put into's called a *Data Model*, which is a part of every computer science student's education, learning how to formally model data.

The discovery of data is part of the process you're implementing, as well as the structuring of this into a form that can be considered a data model, is outside the scope of this book. I start when you've completed a data model which has been delivered to you for implementation, and you need to make this model available to applications and processes from within the JBoss BPM Suite.

Let's imagine you're part of a project team which is automating a process to determine if a customer of your financial institution has the right income and age to be considered for a car loan. There has already been a round of discovery workshops to uncover the process steps, which uncovers the data needed to complete the process. This data's modeled by someone in the project team and delivered to you as the project team member responsible for implementing this data model.

At this point, with your data model in hand, you'll start the journey of implementing data objects for your process projects.

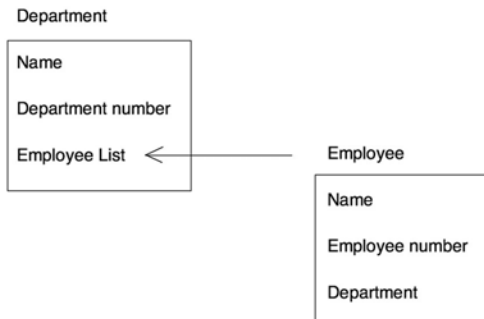
### 3.1 **Data modeling tooling overview**

Traditionally the art of data modeling referred to how data was structured in an organization. You can find detailed definitions of data modeling<sup>1</sup>, but for our purposes I'm focusing on being able to create a simple set of data objects that represent the data being manipulated in our process project. I don't provide details provided on how to design data models, nor do I discuss the actual code generated behind the data modeler tooling when you create a data object.

The data objects used here are the following:

- Employee
- Department

An employee has a name, employee number and a department that she works in. The department data object has a name, department number, and list of employees that belong to that department. This is a simple and fictitious data model, shown in figure 3.1, from the example project described above and I'm going to walk you through implementing it in JBoss BPM Suite.

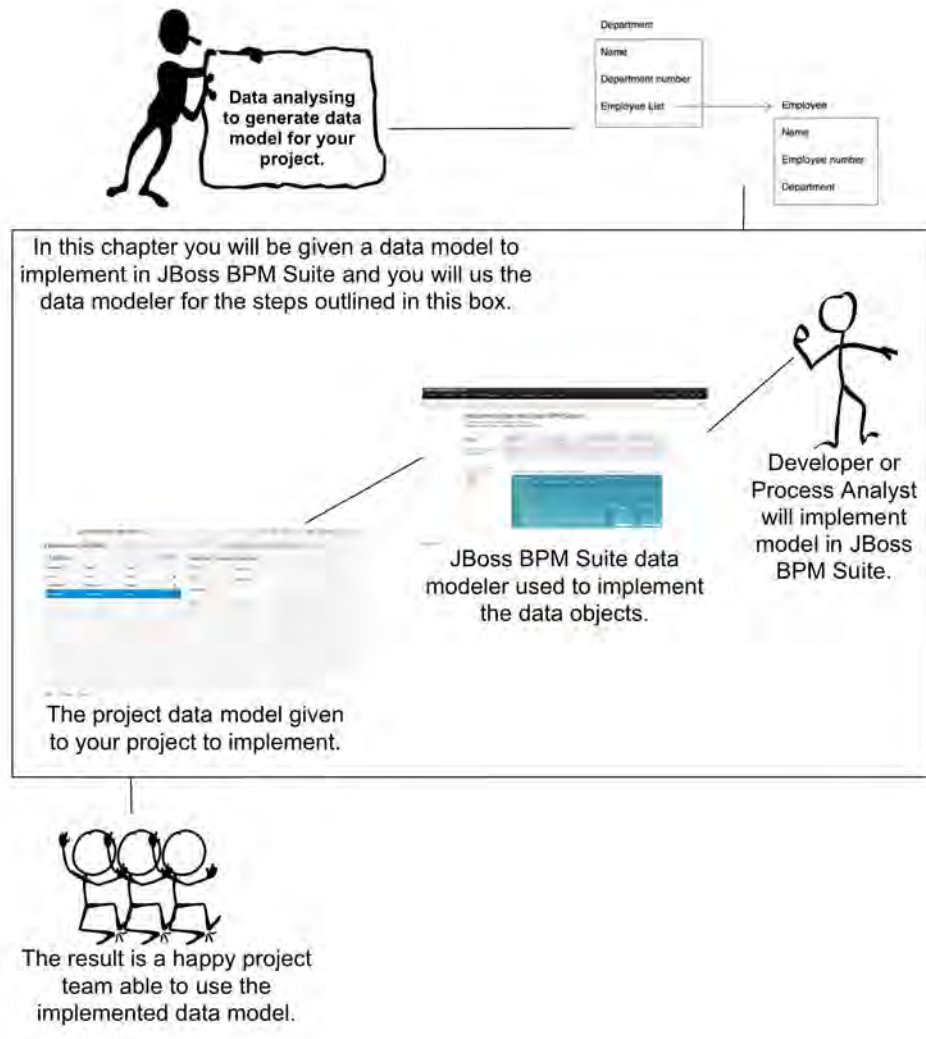


**Figure 3.1** This example data model has been given to you for implementation in JBoss BPM Suite. It consists of two data objects; Department and Employee.

#### 3.1.1 **Getting started with data modeling**

The first thing to do with a new design for your projects data model's to start implementing that model. This can be done by developers in the Java programming language using their favorite editors or integrated development environments, and given to your project, or you can use the data modeler provided by JBoss BPM Suite. I'm going to show you how to use the data modeler within the JBoss BPM Suite tooling as shown in figure 3.2.

<sup>1</sup> A formal definition can be found at [http://www.webopedia.com/TERM/D/data\\_modeling.html](http://www.webopedia.com/TERM/D/data_modeling.html), which states "Data modeling is often the first step in database design and object-oriented programming as the designers first create a conceptual model of how data items relate to each other. Data modeling involves a progression from conceptual model to logical model to physical schema."

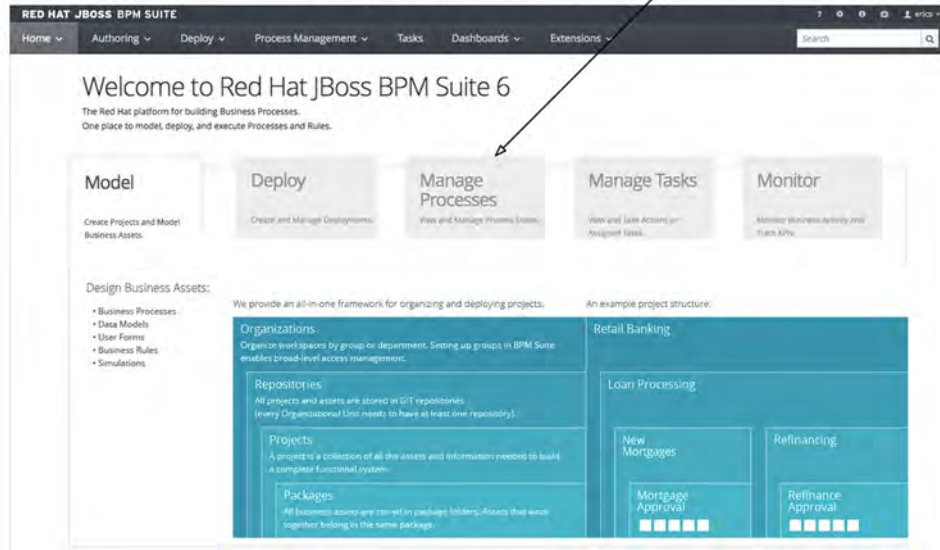


**Figure 3.2** An overview of the path you'll take in this chapter's outlined. Inside the box are the steps to be covered, from the moment you receive a data model design, through implementation in JBoss BPM Suite, to the final data objects in your project for use by the rest of your team.

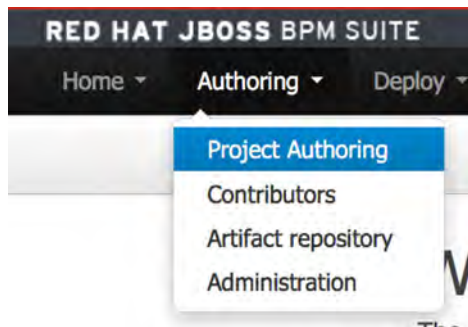
Let's get started modeling the data model we've been given. First, you need to login to the Business Central console as shown in figure 3.3. If you haven't previously done it, browse the documentation found in the tabs indicated by the arrows in the figure to get a feel for what's available in JBoss BPM Suite.

Next, you open the project authoring perspective to begin accessing and creating your data model. This is found in the *Authoring* menu as shown in figure 3.4; select *Project Authoring* to open the perspective where you see the data modeler.

The documentation is available under the Model, Deploy, Manage Processes, Manage Tasks and Monitor tabs.



**Figure 3.3** After logging into the Business Central you'll be at the home screen where you can browse the documentation. This home screen's where you start your work on data modeling.

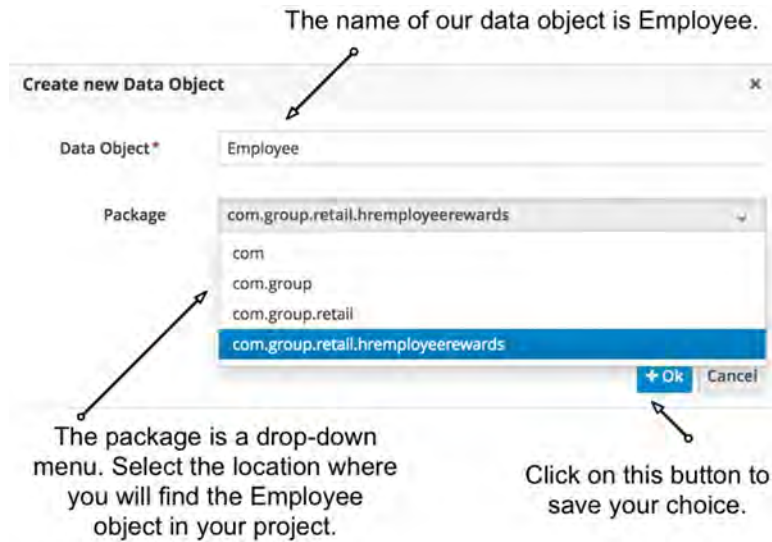


**Figure 3.4** The Project Authoring perspective can be found in the Authoring menu and opened by clicking on the Project Authoring entry. This'll get you to the various authoring tools for any BPM projects you're working on.

Now that your project's in front of you, open the data modeler by selecting from the *New Item* menu a new *Data Object*. This produces a pop-up, shown in figure 3.5, where you can get started on your first data object, the Employee. You can provide the following details to get started with the Employee data object:

- Data Object : Employee
- Package : com.group.retail.hremployeerewards
- Persistable : (leave check box empty)

The first's the name of the data object, Employee, as we're going to create an object to hold our employees for this project. The package name's selected from a drop-down menu and specifies where the data object's to be stored in your project. The final item,



**Figure 3.5** Use this pop-up to create a new data object.

*Persistable*, is a check box that allows you to indicate that you want to save this object to a database table, and provides special configuration details that need to be generated for you. That sort of mapping of data to a database table's outside of our scope as we're managing our data objects in memory, and you can leave that box empty.

You finish creating the Employee data object by clicking on the +OK button, which opens the data modeler with the provided Employee details already inserted in some of the fields you see.

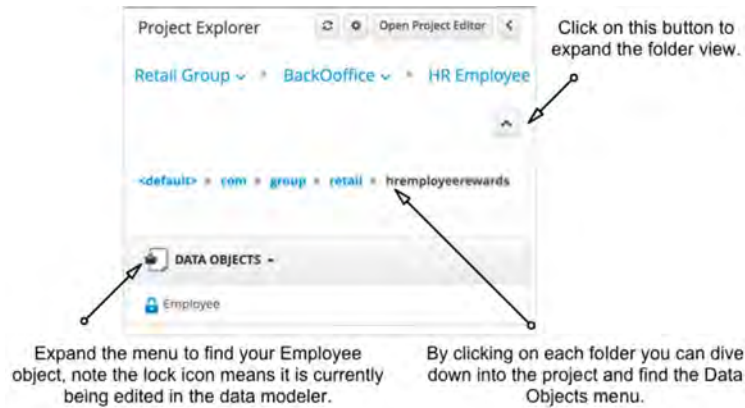
Now the Employee object appears in the data modeler within your Business Central console and you're ready to interact with the data modeler to start adding data fields like name, department and employee number.

Now you expect to see the newly created Employee data object in your Project Explorer pane on the left side of your screen, right? The reason that it isn't there's because you're viewing the *default* level of your project, which contains all the assets you created, except for assets given package names that are deeper into the project folder structure.

If you remember, you created the Employee data object with the package set to *com.group.retail.hremployeerewards* which translates to the folder structure of *com/group/retail/hremployeerewards/Employee.java* in your project. Any time you wish to browse or select the data modeler for the Employee object, you need to first navigate down to this folder before it appears in the Project Explorer.

In figure 3.6, the Project Explorer's shown with the folder structure expanded.

If you click on the Employee object, it opens in the data modeler. Any time you're looking for the data objects available in a project, you can follow this process within the Project Explorer to traverse your project and explore data model assets.

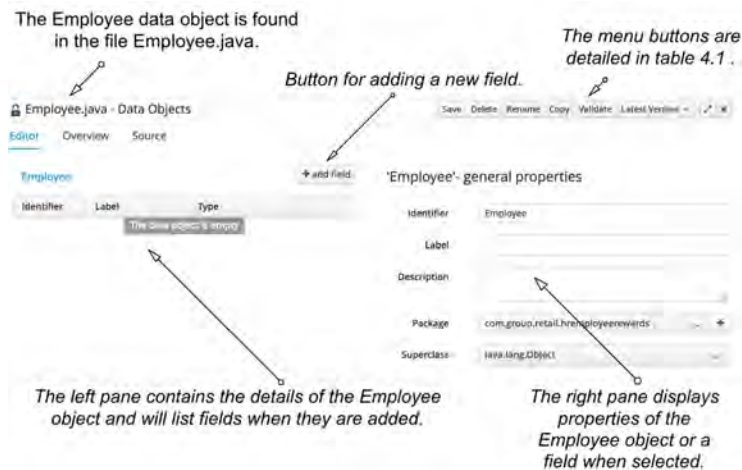


**Figure 3.6** Data objects are put into packages in which they appear hidden from the main Project Explorer default view. To find them you need to expand your folder structure.

### 3.1.2 Taking a close look at the data model editor

After initializing the Employee object, you're now at the point of adding all the identified fields like name, employee number and department where the employee's working. This section takes you through the data model editor, explaining what's available to complete the Employee object.

In figure 3.7 you see the Employee data object in its initial state after opening it in the data model editor. Nothing has yet been created outside of a name and assigning it to a location in the projects package structure. This is at the starting point for further implementing the details of this Employee object. To start working with the data modeler you need to understand the layout of this editor.



**Figure 3.7** The JBoss BPM Suite data model editor where you create, modify and view data objects.

You'll use a few menu bar buttons while creating and updating your data objects. Let's look at their actions in table 3.1.

**Table 3.1** An overview of the data modeler buttons as found on the top right of the screen. You can activate six actions by clicking on one of the buttons shown here.

Button name	Button action
Save	persists the contents of your data object to the file system
Delete	remove this data object from your project
Rename	change the name of the file holding this data object
Copy	create a copy of this data object, and provides a quick way to setup a new data object
Validate	validates that the data object's in a good state to be used
Latest Version	a menu that lets you select any existing version saved in the past should you want to go back to a previous version of the data object

Any time you're modifying your data objects, be sure to use the Save button and provide a short message as to what you've changed. It's easy to lose some of your work by forgetting you're working in a browser-based editor and pressing the back button on your browser.

### 3.1.3 Adding fields to a data object

Now that you've an idea of what the data model editor looks like, let's finish the Employee object by adding all the fields that were identified earlier. Let's start with the field for the employee name by clicking on the *+add field* button to generate a pop-up labeled *New Field* as shown in figure 3.8.

**Figure 3.8** The *New field* form pops-up when you click on the *+add field* button in the data model editor.



This pop-up's asking for the values that you can enter with the following values shown in figure 3.9:

New field

- Id : name
- Label : Name
- Type : String
- List : don't check this box

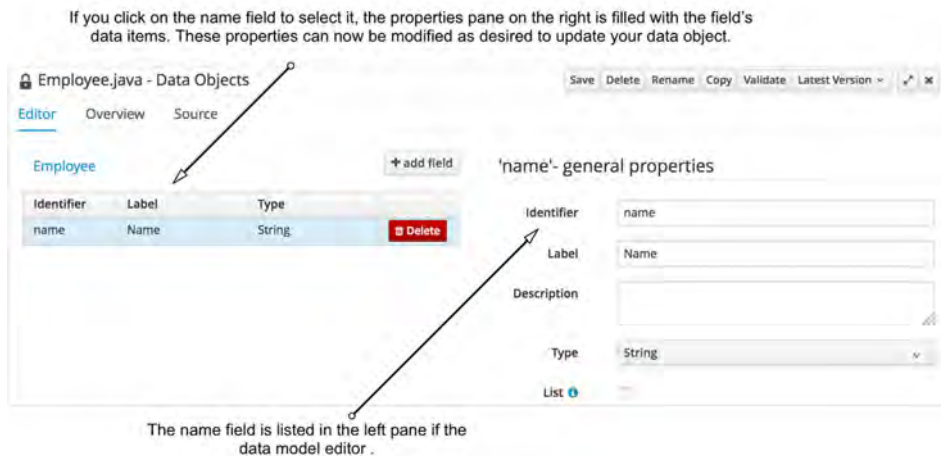
**Figure 3.9** This is the entry form called *New field*, used to add relevant information when adding a field to an existing data object. Here's a field with an *id*, *label* and *type* with the values *name*, *Name* and *String* are being created.

Once you've completed all the items for your field, you can submit them with the *Create* button or with *Create and continue*.

- The *Create* button submits and closes out the *New field* pop-up form.
- The *Create and continue* button submits your field and opens an empty *New field* form, allowing you to add multiple fields quickly.

Figure 3.9 also gives a good look at the item *Type*, which is a menu with many entries to pick from. Most are from the Java language, as this is the implementation language used to create JBoss BPM Suite. One interesting entry isn't a standard Java type; did you see it? Look closely and notice the entry entitled *com.group.retail.hremployees.Employee*. If you think back, when we created our initial *Employee* data object, it was put into a package with exactly that name. As you can now see, each object created's also available for use as a field type, even as a list of types if you should check the box making a field a list.

At this point use the *Complete* button to submit the field and close the form. This way you can examine your work as the data object editor now includes the name field in the list of fields in the left pane as shown in figure 3.10.



**Figure 3.10** After adding the first field *name*, it appears in the data modeler.

Now you're a real BPM data model editor expert, having successfully created the Employee data object and added a name field! It's time for you to fly solo and create the two remaining fields for Employee; the employee id and department fields. Create and verify them against figures 3.11 and 3.12. Otherwise follow along as I help you to create them.

Click on the *+add field* button in the data model editor to create the employee id field with the data here, it should look like figure 3.10 after you submit with the *Create* button:

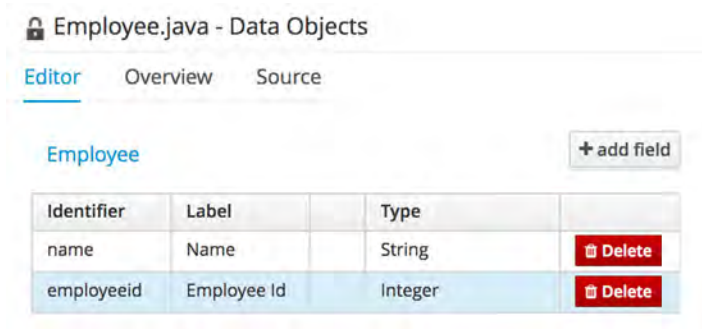
Employee id field

- Id : employeeid
- Label : Employee Id
- Type : Integer
- List : don't check this box

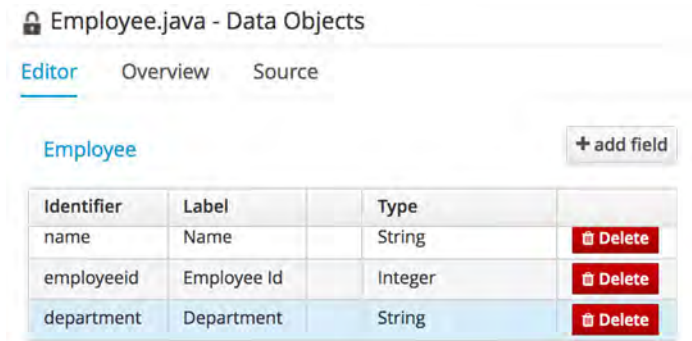
For the last field, you're adding to the Employee object, click on the *+add field* button in the data model editor to create the department field with the data here, submit it and ensure it looks like figure 3.12:

Employee id field

- Id : department
- Label : Department
- Type : String (for now, see why in figure 11 description)
- List : don't check this box



**Figure 3.11** The Employee object in the data model editor should look like this when you finish adding in the employee id field.



**Figure 3.12** The Employee object in the data model editor should look like this when you've added the last department field.

This completes the Employee data object with the fields name, employee id and department. You can now use the *Save* button in the data model editor found in the top right—it produces a pop-up labeled *Save this item*, in which you can enter a comment. For example, enter “*Created Employee Object*” and click on the *Save* button to save your data object.

Before you finish the entire data model by adding the Department object, here are a few more of the features available to explore and work with your Employee object. If you've been paying close attention to your screen you might notice two more tabs at the top of the editor, let's look at them.

### **3.1.4 More to the data modeler than meets the eye**

The data modeler offers a few tabs at the top, the first being the editor you've been using. The second is labeled *Overview* and it's the next one that shows what can be done with the Employee object you created.

You can open the *Overview* tab by clicking on it, which opens the view shown in figure 3.13. Note that this view contains the same main header and menu buttons on the top left or right as was covered in the editor.



**Figure 3.13** The data modeler Overview tab provides all the details around the currently selected object. In our case, it's the Employee object.

The section on the top left provides detailed information about the Employee object, such as its type, a field to add a description for this object, which projects use this object, the date this object was last modified, and when you created the Employee object. On the lower left, you see the versions of the Employee object that you've saved in the past. The current version's listed at the top and it's the one currently displayed in the data modeler. If you go back to a previous versions listed, select an entry and it'll replace the current version of the data modeler. On the right, you've a comments field where you can leave information for your fellow modelers.

If you click on the *Metadata* tab next to the *Version history* tab, you'll see something like that shown in figure 3.14. This shows you an editor to add some extra, or meta, information about this data object. You can insert a tag name, like *employee-docs* as shown here, and click on the *Add a new tag* button. The *Note* field shows you the last save message for this object and the actual path to the object's file in your project. The *URI* field points to the exact file for checking out the current data object from the GIT repository for this project. The *Subject*, *Type*, *External link* and *Source* fields are all free form text fields that let you put any information in them you deem pertinent for this data object.

These fields are left to you to determine how you want them to be used. For example, here we set the Type to Doc as we're relating all the information supplied here to the documentation-related employee-docs tag. All this extra information can be added to enrich the information supporting your data object, and is often detailed

The Tags field provides a search term with you are looking for assets in the project that are labeled with the tag 'employee-docs', clicking on 'Add a new tag/s' button to submit.

Version history **Metadata**

Tags: employee-docs Add new tag(s)

Note: Saved. (/HR Employee Rewards/src/main/java/com/group/retail/hremployeeerwards/Employee.java)

URI: git://master@BackOffice/HR%20Employee%20Rewards/src/main/java/com/group/retail/hremployeeerwards/Employee.java

Subject: \_\_\_\_\_

Type: \_\_\_\_\_

External link: \_\_\_\_\_

Source: \_\_\_\_\_

Lock status: Locked by you  Force unlock asset

URI field shows location of the file in the underlying code repository.

Note field shows the last saved message and file location in project.

These fields are shown with extra metadata or information about the data object. There is also a lock status where you can force the unlocking of the object if someone else has it open or left it open.

**Figure 3.14** The tab labeled **Metadata** provides you with the chance to add extra information about the current data object.

about how an organization works with data. The more details in your artifacts, the less chance of confusion later for anyone who revisits your data object.

The last field, labeled *Lock status*, shows if someone locked this data object by having it open in the data modeler editor, and provides a button that allows you to unlock it.

Valid reasons to take away a locked object exist. Imagine your colleague was working on the Employee object and left it open on his machine when he went away on an extended vacation. It might be nice to be able to access it after you've verified that the person isn't going to be working on the Employee object.

Now that we've opened the Employee object, it puts a lock on this behind the scenes. The next person, working on a different workstation and in a different browser, who attempts to open this Employee object will find it locked. It opens for them, but it's shown with a small lock icon next to the name in the Project Explorer, and is only readable. They could use the Metadata tab to force an unlocking of the data object, but be aware that they're possibly interrupting your work as you're unaware that they've taken the data object lock away. This is a light form of locking, which means it doesn't inform anyone about who forces the unlocking, and everyone using the system's responsible for behaving nicely when taking away a locked object from another user.

Remember to save any modifications you've made in the *Overview* tab by clicking on the *Save* button found in the top right menu bar; it produces a pop-up labeled *Save this item* that allows you to enter a comment. For example, enter “*Added metadata to Employee Object*” and click on the *Save* button to save your data object.

### 3.1.5 Using the data model source

The last tab in the data modeler's labeled *Source*, and it provides a learning tool for those interested in looking inside data models. If you click on this, it shows you the Java source code generated for the data object you're working on. In figure 3.15 you can see an example of a portion of the *Employee* object source code.

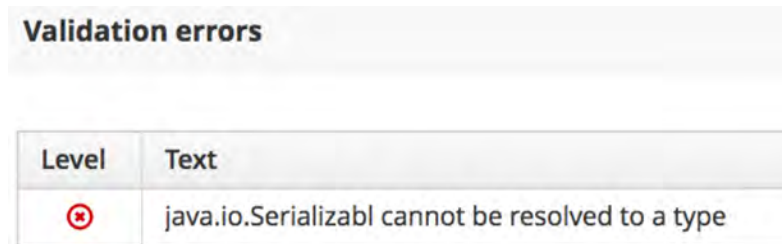


**Figure 3.15** The *Source* editor's the tab that provides a developer view of the actual Java source code that makes up your *Employee* data object.

This *Source* editor's a free text editor without any developer tooling such as code completion or other help. It's a view of what a Java developer would code in an integrated development environment (IDE) tooling, and could add to the project. The only point to note's that if a developer chooses to code their own data objects and add them to the project, they must implement the Java class `java.io.Serializable` to work with a JBoss BPM project. This is because everything in the project, behind the scenes during runtime, is *serialized* before being stored in the persistence layer (i.e. database) and *unserialized* when it's retrieved from the persistence layer.

Furthermore, each data object with fields is a Java serializable object with *getter* and *setter* methods. For example, the `name` field uses a `getName` method and a `setName` method. Be aware that this editor doesn't support code completion or other fancy tooling, it's a free text editor and you can easily break your data object if you aren't careful. Use the *Validate* button before saving to ensure your data object is working.

Let's look at what you can do if you decide to edit a data object by hand in the *Source* editor, to make sure you aren't saving a broken data object. If you click on the source code and remove the last 'e' in the word *Serializable* on line seven from figure 3.15, you can validate the data object before saving by clicking on the *Validate* button in the top right menu bar. After removing the 'e' a pop-up box with a validation error should appear, as in figure 3.16.



**Figure 3.16** If you're going to be editing a data object's *Source*, you'll need to use the *Validate* button to ensure you haven't broken your data object. This pop-up shows that you've broken your data object by removing the 'e' from line seven in figure 3.15. The only way to fix this is to click on the *+Ok* button and go back to the source code.

If you fix that error by putting the 'e' back on to the word *Serializable*, click again on the *Validate* button. You should see a green bar pop-up with a message stating that *Item successfully validated*. You can now save the data object without fear of breaking your project.

As you can see, the *Source* editor's a quick way to fix something in your data object, but requires discipline to constantly validate that your changes haven't broken anything. If you stick to the provided data modeler *Editor*, you'll never worry about these types of problems.

Let's move on and finish your data modeling task by adding the *Department* object, its various fields, and using the *Employee* object to create a field which is a list of employees within the *Department* object.

## 3.2 **Complete your data model**

In this section, you finish up the data model used as an example in the previous section to tour the tooling at your disposal in JBoss BPM Suite. The initial object was a simple *Employee* designed with three straightforward fields.

For the final object in your example model, you design a *Department* with two straightforward fields and an advanced field that consists of a list of employee objects. Instead of walking you through each field creation, I leave the first two for you to complete after you create the *Department* data object as follows:

- Data Object : *Department*
- Package : `com.group.retail.hremployeeerewards`
- Persistable : (leave check box empty)

The new Department object should look like figure 3.17.

'Department'- general properties

Identifier	Department
Label	
Description	
Package	com.group.retail.hremployeeerewards
Superclass	java.lang.Object

**Figure 3.17** The Department data object as shown in the right pane properties view.

Next you can create the *Name* and *Department Id* fields as follows:

Department name field

- Id : name
- Label : Name
- Type : String
- List : don't check box

Department id field

- Id : departmentid
- Label : Department Id
- Type : Integer
- List : don't check box

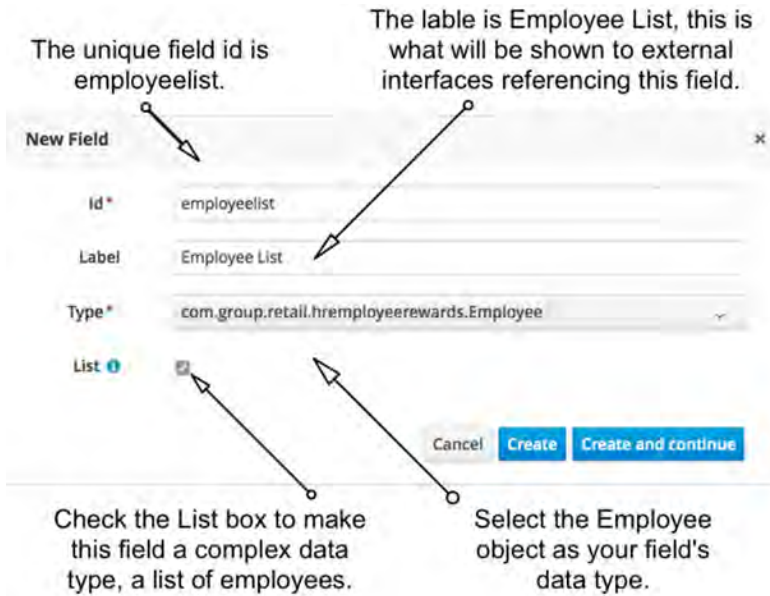
Now you need to use your previous work, the Employee data object, and create a field for the Department object that lists Employees, to keep track of the employees in a department. This can be done as follows:

Department employee list field

- Id : employeelist
- Label : Employee List
- Type : com.group.retail.hremployeeerewards.Employee (select in menu)
- List : check this box

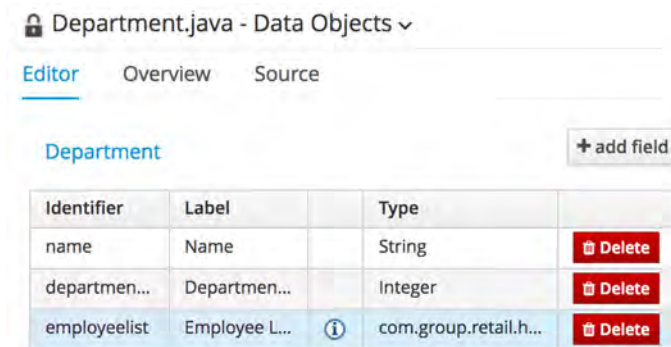
The advanced feature of this field's the ability to select an existing data object as the *Type* and check the *List* box as shown in figure 3.18.





**Figure 3.18** The final field for your Department object's an advanced type that uses the Employee object previously created, and it's part of a list. You'll need to check the box next to *List*.

This completes the Department data object with the fields name, department id and employee list. Save your Department object, it should look like figure 3.19. Congratulations, you've completed your entire example data model in this chapter and you've done it in time to make your project manager happy by meeting the tight project deadline!



**Figure 3.19** This is what your Department data object should look like (the order may differ for you) after completing all the fields you needed to create, including the advanced list of employee objects for the Employee List field.

Now that you can create data models, including usage of a data object as a list within another data object, the only question left's how can you use existing data models from your organization? Imagine you want to import work done in other projects that include data objects compatible with JBoss BPM projects, meaning they're standard Java objects and implement serialization.

In the next section, you discover how to do exactly that.

### **3.2.1 What to do with an external data model**

When you're working in a true enterprise organization, the chances of having the option to create all your project artifacts as new development are rare. For example, your enterprise probably has existing services that tie together its backend systems for your BPM project to use. What you most often encounter's the need to use data objects from existing enterprise data models within your current BPM project. This section shows you how to ensure that the data objects that haven't been created by you can be made available in JBoss BPM Suite for your entire project.

This section doesn't teach you to design or build an external data model, but it assumes you've ensured it's functionally ready to import into your JBoss BPM Suite project. I take you through a sample external data model, showing you what it means to be functionally ready for importing, import this external data model into the HR Employee Rewards project you've been using in this chapter, and I show you how to ensure it's available to the entire project.

### **3.2.2 Your external data model brought to you by ACME**

The problem's that you need to make an existing part of your enterprise's data model available to your current new BPM project. To simulate this, I use the existing HR Employee Rewards project you worked through with the Employee and Department data objects already created as shown in figure 3.19. You import an externally developed data model into this project to meet the requirements that you defined.

Imagine that the external data model's a piece of another project within your enterprise that provides for booking travel to a given destination. Within that travel booking project you can book a hotel, and for your current project you're expanding the employee rewards process to include a hotel stay as a possible reward. The framework of the travel booking project's data model works nicely for your current project, and you're interested in reusing it here.

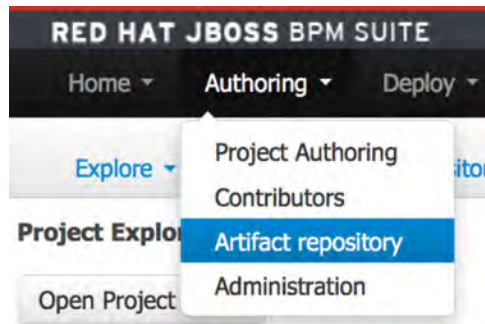
This travel booking data model's in the format of a Java archive, meaning that it was pre-built before you added it to your project. The *Java archive (JAR)* file discussed in the rest of this chapter's a compressed set of the Java objects ready for deployment and use in the JBoss BPM Suite server.

Any external data model can be added to your project if it's functionally ready, meaning it must be plain Java objects that implement the `Serializable` class as covered in section 3.1.5, and be built into a JAR file.

### 3.2.3 Using the artifact repository effectively

When you're working in the project authoring perspective, you're designing your data model, but you don't see a place to add an existing data model. The question forms in your mind, 'Where does one start importing an external data model once the model's provided?'

Within your JBoss BPM Suite Business Central console's a separate area reserved to provide a listing of the artifacts associated with your projects. This view's called the *Artifact Repository* and is found in the *Authoring* menu as shown in figure 3.20.



**Figure 3.20** The Artifact repository view can be found in the *Authoring* menu and it opens a list view of everything currently available to your BPM projects. Click on *Authoring* and then *Artifact repository* to open.

This opens the artifact repository with its default list of artifacts that BPM projects can depend upon or use. In the top right, you find the *Upload* button where you can add external data models. Next to the upload button's the icon refresh button, used to reload the list in case any uploaded artifacts haven't yet appeared in the list. Each entry in the list's an artifact that can be used in your BPM projects, and a few buttons to explore those artifacts in greater detail. Each artifact has a *Download* button, which provides you with a local copy of the listed artifact. The second button's labeled *Open* and provides a view of the artifact's details. Initially, this list contains several default entries, one being the JBoss Application Server CLI and several client libraries.

Let's upload our external data model, the *ACME Travel Data Model*, obtained from the provided project.<sup>1</sup> It's assumed you've downloaded the project, followed the provided instructions, and built your own JAR file. When you click on the *Upload* button in the top left, a pop-up appears asking you to choose a file to upload, as shown in figure 3.21.

This completes the import of your external data model, the *ACME Travel Data Model*, but you can't use it until you've added it to a specific project. Let's look at how you can do that.

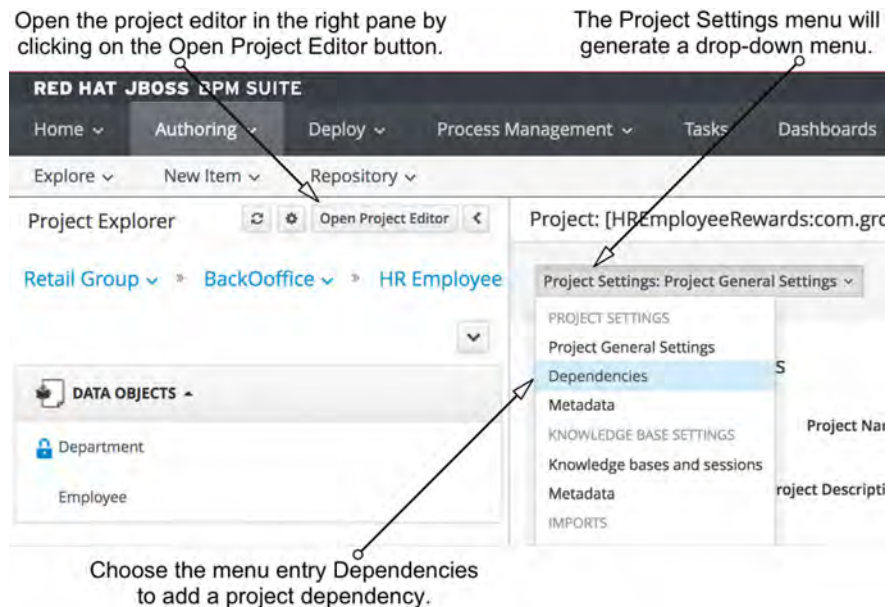
<sup>1</sup> The *ACME Travel Data Model Project* (<https://github.com/effectivebpmwithjbosssbpm/chapter-4-acme-travel-data-model>) provides you with instructions on how to install and build the data model as an external JAR file which you can then import into your project in JBoss BPM Suite.



**Figure 3.21** This is the pop-up you get when you click on the *Upload* button in the Artifact repository. You're asked to choose a file, which is the ACME data model in your case, for uploading into JBoss BPM Suite's artifact repository. Follow the instructions in the ACME Travel Data Model project and select the *target/acmeDataModel-1.0.jar*. Once you click on the *small upload arrow icon* button, the entry for the *acmeDataModel-1.0.jar* should appear in the artifact repository list along with the POM file. If not, refresh the view and you should now see the model artifact appear. The POM file provides you with more information on what the JAR file contains, viewable by using the *Open* button, as I previously described.

### 3.2.4 How to use an imported external data model

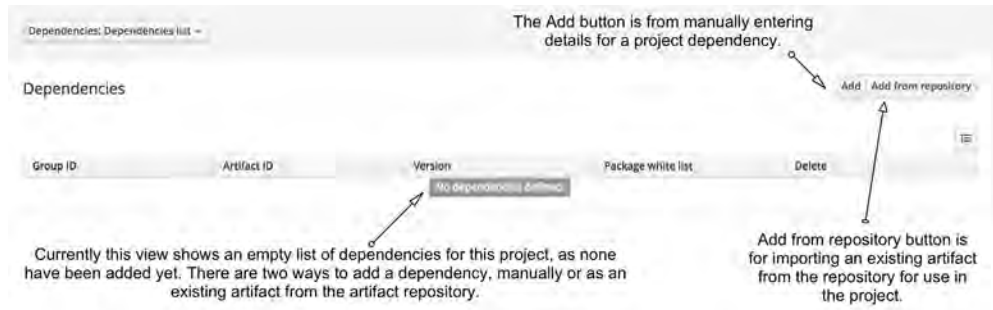
The previously imported ACME Travel Data Model's now listed in the artifact repository, but we need to add it to our BPM project before we can use it. Return to your HR Employee Rewards project by opening the Project Authoring perspective as shown in figure 3.4, then click on the *Open Project Editor* button and open the *Project Settings* menu to select the *Dependencies* view as shown in figure 3.22.



**Figure 3.22** From the Project Authoring perspective, you'll be able to add the dependency on our imported external data model.

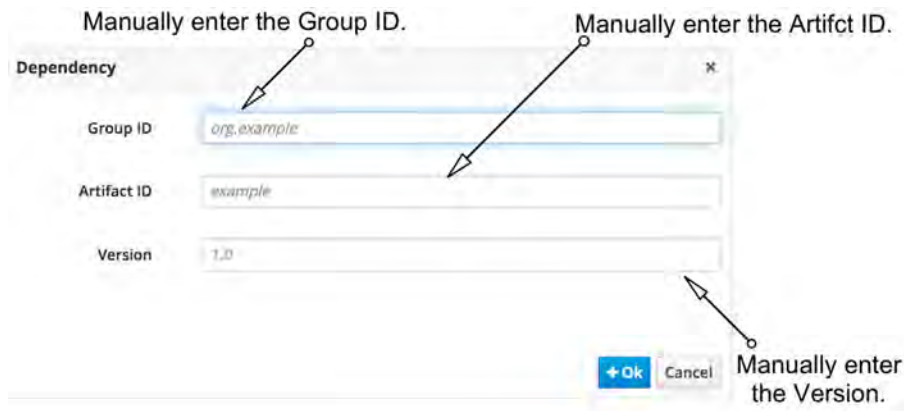
This opens the project editor's *dependency view*, which is currently empty but has options to add dependencies to this project. In figure 3.23 you can see an empty listing of dependencies which are currently part of this project, but this changes after you add in the ACME Data Model as this project's first dependency. You can add a dependency using one of two buttons:

- The *Add* button, which is for entering dependency information manually.
- The *Add from repository* button, which is for importing existing artifacts already in the JBoss BPM Suite maven repository for use in the project.



**Figure 3.23** The project editor's dependency view, used to view and add dependencies to your project.

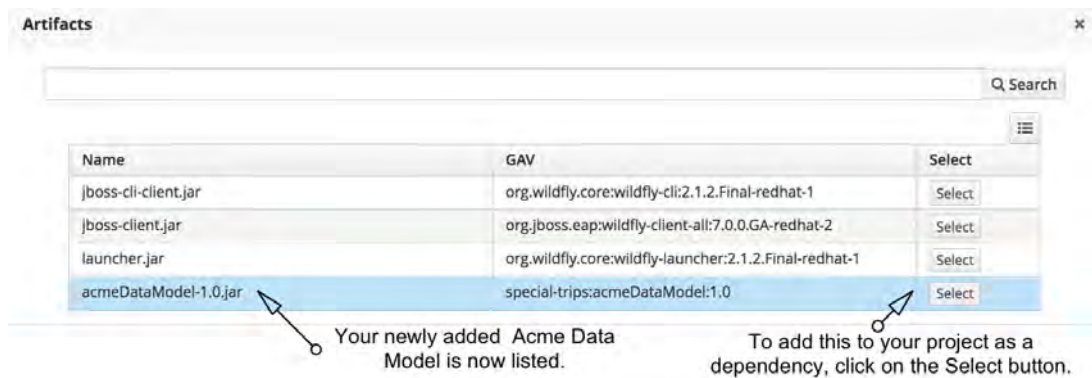
Let's first look at the manual entry option before we use the add from repository. Start by clicking on the *Add* button. This causes a pop-up to appear where you need to manually fill in your dependencies details. It requires all the details a developer recognizes as necessary to make your data object available to your BPM project, as shown in figure 3.24. These are technical details added to your project to find the data objects contained in the artifact you're referencing. These are related to how the artifact's stored in the JBoss BPM Suite internal maven repository.



**Figure 3.24** The *Add* button generates this pop-up where you'll need to manually enter each item asked for to complete the entry.

If you're certain of the details for a dependency in your organization, you can input all the fields needed, and your project has access to the data objects. Be careful though, this data entry isn't validated for you, and it's easy to make a mistake when manually entering this type of information. It's recommended that you first add your artifact to the internal repository and then add an existing entry from the artifact repository. For example, let's add the ACME Data Model you imported and see how you can add this as a dependency.

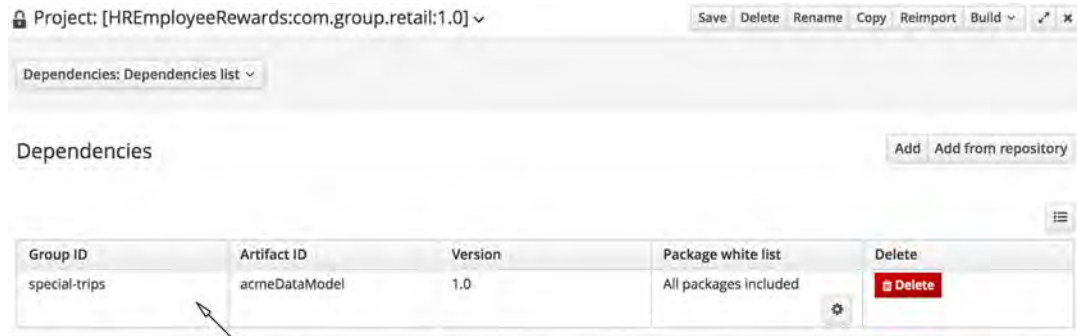
Click on the *Add from repository* button and a pop-up appears with a list of available artifacts in the current artifact repository. In figure 3.25 we see the list of artifacts that includes our previously added *acmeDataModel-1.0.jar*. You can add it to your project as a dependency by clicking on the *Select* button.



**Figure 3.25** The list of artifacts available in your repository are shown here with your newly imported *acmeDataModel-1.0.jar* waiting to be selected.

After selecting the external dependency, you see that the *ACME Data Model* entry listed with the various details filled in for you, as shown in figure 3.26.

Later when you're ready to use this data model, it's available for you to reference from rules, processes, and other artifacts you generate as you complete your BPM project. The way to use this model's to point to the models exact package location, for example importing it from the package *com.jboss.soap.service.acmedemo.Flight* to reference the *Flight* data object.



With the new dependency added to your project, you can now reference the data objects it provides in your BPM project as you design rules, processes and other artifacts for your solution.

**Figure 3.26** The Acme Data Model dependency has been added into the list of dependencies for your project with all the details automatically filled in for you.

### 3.3 Summary

- Every process project makes use of data provided by a data model.
- It's easy to implement a data model object in JBoss BPM Suite data modeler tooling.
- Data object creation's a few menu-clicks away and you're adding fields to data objects.
- The source code editor provided in the data modeler can be a valuable learning tool for viewing your data model internals as you build them.
- Taking notes and other meta data can easily be added in the data modeler overview to flush out a data objects details.
- Within the data modeler overview's a version history that can easily be used to jump back to a previous version of your data object; select a version and continue editing.
- External data models can be imported by uploading into JBoss BPM Suite for use in a project.
- Imported data models are stored in the JBoss BPM Suite internal maven repository, making them available for your project.
- Complex data objects are quickly created by using other data objects in your current data object as fields.
- Fields in a data object can be simple Java data types, other data objects, or lists of data objects to provide flexibility in creating your data models.
- Developers can use web tooling to generate data objects and expand them with persistence annotations, saving both time and tedious coding chores.

# *Starting with business rules*

---

## ***This chapter covers***

- Understanding the basics of business rules
- Using JBoss BPM rule editors to design business rules
- Implementing technical rules, guided rules and test scenarios

Business logic guides the flow of work in a specific business domain to allow it to function in a consistent, repeatable and predictable manner. It's at the core of how things are decided within the projects, within applications being built, and in processes which are captured. It's hard to imagine any sort of BPM suite of tools without the capabilities to support the creation, management, and execution of business logic within a BPM project. JBoss BPM Suite has these capabilities with the supporting tools for you to create, manage, and deploy business logic.

To implement business logic in an organization often means creating, in some technical language, an implementation of that logic. This implementation turns the business logic into application logic in the form of rules. These rules can be written directly into each of your applications, but this leads to repeated use of the same rules in many applications. It also means any change to a rule's a change in the application, and therefore requires a new release to use it. Finally, the rules encapsulate business knowledge, and the best person to manage this knowledge is the business user from that knowledge domain. If you take business logic and code it into rules within applications, developers are now responsible for interpreting, managing and maintaining the rules.



The solution for these problems is to externalize business rules into JBoss BPM Suite<sup>1</sup> where you've tools to manage and maintain the lifecycle of rules outside of the lifecycle of the applications using them. The goal of this chapter's to put business rules into the hands of the business owners in an easy to understand form, not as application code. In this chapter I take you on a path to implement business rules using guided tooling provided in JBoss BPM Suite. I walk you through technical and guided rules, which are the first steps you take on the road to implementing your application business logic. This requires more than implementing rules; you need to ensure that the rules pass tests you set for them to ensure that the rules do what you need them to do. Therefore, I teach you how to add test scenarios that exercise the rules you've created. By centralizing the business rules and test scenarios in JBoss BPM Suite, it becomes clear they're maintainable by the knowledge owners who're best suited for this task.

What's beyond the scope of this chapter's the exact syntax used for JBoss BPM Suite rules, which can be found online<sup>2</sup>. The focus here's to teach you what the tooling does to help you easily implement business logic into business rules by using examples I provide. These examples are presented in a pseudo-code format which is easy to follow.

Now imagine yourself as part of a team that implements the core business rules for an organization that wants to capture business logic in an external system using JBoss BPM Suite. When finished, all the processes and applications use a central managed set of rules governing how your organizations business works. To kick off this project, small tasks are assigned in which you take the given business logic and implement it in each of the rule types provided for by JBoss BPM Suite. Figure 4.1 shows the path taken to learn how to use the various guided editors and tooling to implement externalized business logic for your organization.

## **4.1 Business logic central to your process**

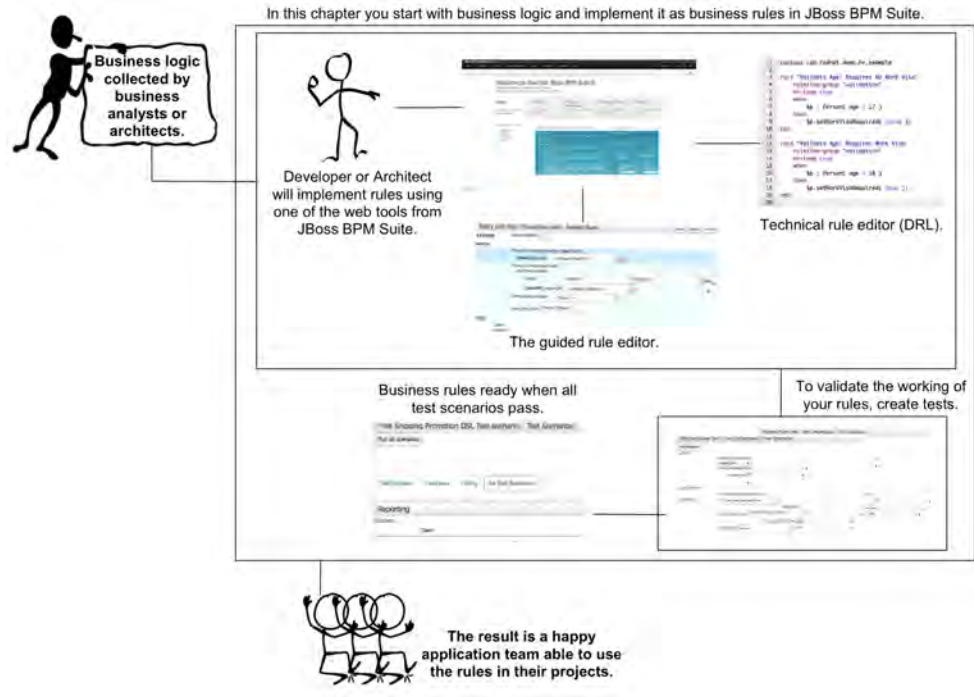
Business logic in any organization can be found almost everywhere. It's in the daily tasks that employees perform, it's found in the processes used to complete tasks, and not surprisingly, it can be found in many of the applications being created.

Let's look at a simple example in an imaginary Human Resources (HR) department. During discovery phases of a project that attempts to streamline the onboarding of new hires, discussions arise around the task of validating a new hires employment status. The input from the HR employee sounds something like this, "When I examine the new hires paperwork, I'm looking for a copy of his/her identification document such as a passport. If the passport's from the US, as we're a US com-

---

<sup>1</sup> JBoss BPM Suite's a super set product that includes the rules and events tooling that can also be found in the JBoss Business Rules Management System (BRMS) product. In this chapter I only reference the JBoss BPM Suite when discussing business rules.

<sup>2</sup> The syntax for constructing rules using MVEL language can be found in the free online documentation; here's the section on rule syntax supported by the version used in this book: [https://access.redhat.com/documentation/en/red-hat-jboss-bpm-suite/6.4/single/development-guide/#all\\_about\\_rules](https://access.redhat.com/documentation/en/red-hat-jboss-bpm-suite/6.4/single/development-guide/#all_about_rules)



**Figure 4.1** The overview of the path you take in this chapter's outlined here. With the provided business logic, you implement business rules leveraging JBoss BPM Suite.

pany, I know that the new hire can work if his/her age's over 18. If I find a foreign passport as identification, then more digging into the employment requirements in the US is required for that specific country.”

A lot of business logic's found in this discussion. It's easy to program this directly into our process application using the company standard Java programming language. The problem with this appears down the road in the future, when applications or processes need to use some of the same business logic contained here. The strategic decision to externalize business logic from new applications is our starting point; let's see what business rules you find from the discussion above.

### 4.1.1 From logic to rules

A key indicator of business logic and where you start to identify rules in code or discovery discussions as shown above, is to look for *if-then*, or *when-then* constructions. In these cases, you test for a condition to be met and, when it's met, it results in some sort of action. This is a rule.

The easiest way to think about a rule's that there's a premise and a conclusion. The premise is the set of one or more *conditions* and the conclusion's one or more *actions* that result from the conditions being met.

Now, without thinking too hard about the formal rule syntax, let's extract some of the business logic you see in the previous discussion with our HR employee. Let's start by looking at each sentence first, then extract the logic found into simple rules using natural language.

- “If the passport is from the US, as we're a US company, I know that the new hire can work if his/her age is over 18.”

The rule here'd be based on checking some of the employee data, which is a hint of the data objects which are available to you in this domain:

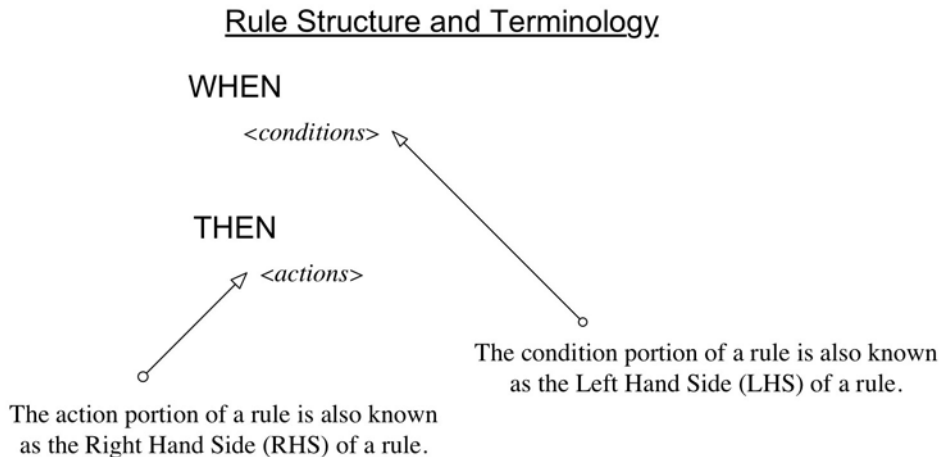
```
WHEN
  Employee age is 18 or higher
  Document is US passport
THEN
  Do nothing
```

- “If I find a foreign passport as identification, then more digging into the employment requirements the US is needed for that specific country.”

The rule here'd be a little different:

```
WHEN
  Employee age is 18 or higher
  Document is not US passport
THEN
  Set Employee needs work visa
```

When you look closer at these rules, you notice that there's a specific structure to a rule as shown in figure 4.2.

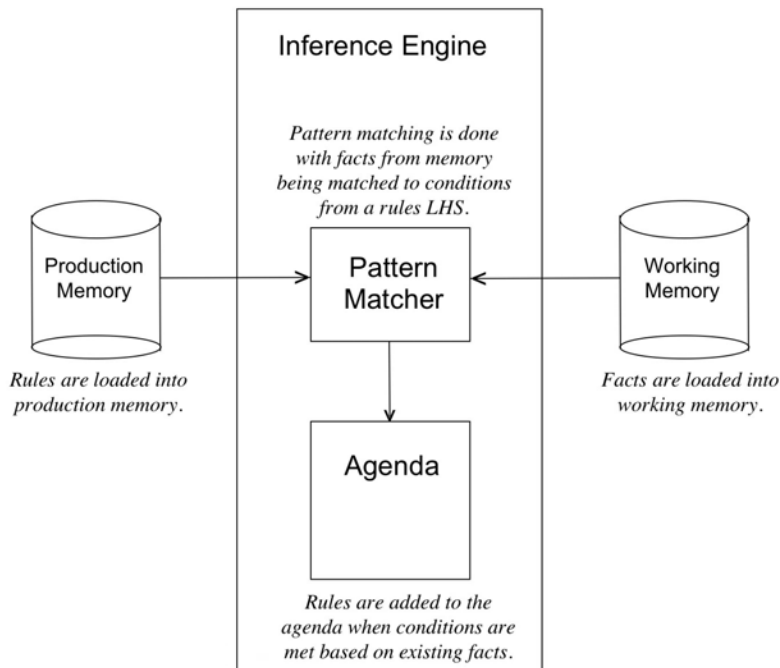


**Figure 4.2** The basic structure to a rule includes new terminology such as LHS and RHS.

The term *left-hand side (LHS)* is used when talking about the conditions of a rule. The rule operates on facts which are data objects placed into memory from your applications. When the conditions or LHS are met, then the rule *fires*. When a rule *fires*, that means that the *right-hand side (RHS)* or *actions* are executed.

#### 4.1.2 Inside the rule engine's brain

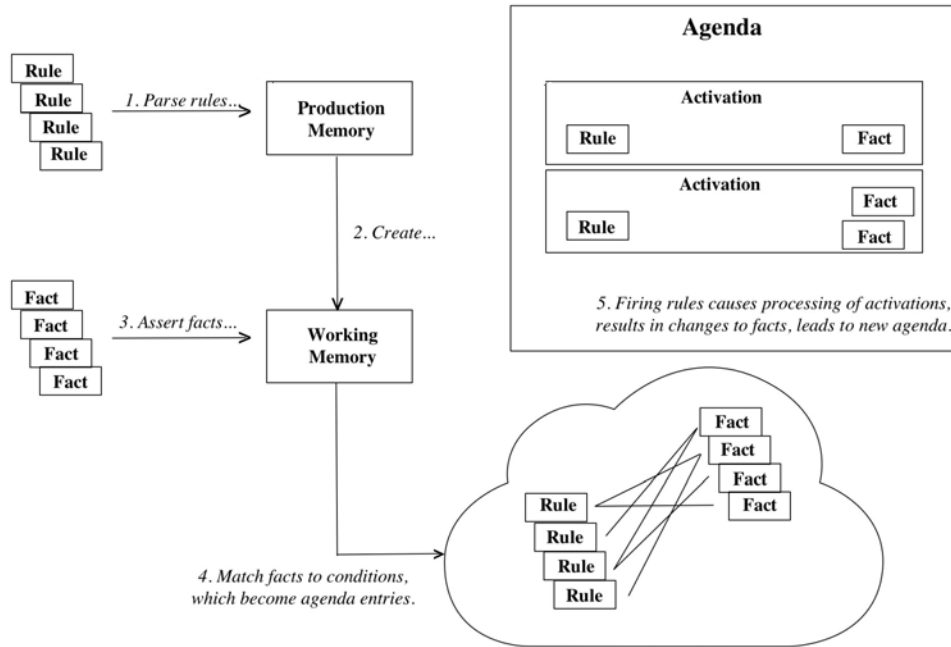
These rules with LHS and RHS need some sort of brain or engine to process the matching of facts to their conditions. This process uses an *Inference Engine*, the brain behind our rule processing system within JBoss BPM Suite. This inference engine matches facts to conditions in your rules and when they match, those rules are fired. Fired rules execute the actions contained within, which either performs an action external to the application or modifies one or more facts within your system. Figure 4.3 shows you an architectural overview of the inference engine as it is used in JBoss BPM Suite.



**Figure 4.3** The inference engine in architecture used by JBoss BPM Suite.

The rules are loaded into *production memory* and the facts are put into *working memory*. Production memory doesn't change, but working memory's constantly being updated when facts are added, removed or modified as rules are fired. The *pattern matcher* is at the center of the inference engine and uses pattern matching to find rules with conditions that can be met. When the conditions are met, the rule's added to an

*agenda*, which is a listing of the actions to be fired. Once all the pattern matching's done and the agenda's complete, the rules on the agenda fire one at a time. Figure 4.4 illustrates this process and the order of execution. If a rule firing causes any change in the facts in working memory, the agenda's erased and the pattern matching starts building a new agenda based on the new facts being applied to the same rules. This continues until there are no more conditions being met and the agenda's empty.



**Figure 4.4** The overview of the JBoss BPM Suite processing of rules, facts and agenda to determine which rules conditions are met and how they're activated.

These are the basics rules you need to understand; what an inference engine is, and how it processes your rules. Now let's look at taking a few sample rules and implement them in the available JBoss BPM Suite tooling, starting with technical and guided rule editors.

## 4.2 Considering technical and guided rules

It doesn't matter which of the editors are used, they simplify the coding of rules in the JBoss BPM Suite's rule language. This language's known as the *Drools Rule Language (DRL)* and can be directly coded into files that developers can produce in their development tools they use for coding. These files end with a \*.drl postfix notation, indicating their rule contents.

In the early days of JBoss rules development, this DRL coding by developers was the standard way to implement business rules. This is a powerful way to extract busi-

ness logic, as we've discussed, but it doesn't liberate the rules from the domain of the developers. To bring DRL to the business users or architects that might not want to dive into the details of DRL syntax, JBoss BPM Suite brings you several web based editors and wizards to help. Let's look at the first two, the technical rule editor and the guided rule editor.

### 4.2.1 Technical rules for developers

The first option available's for developers in that it's an editor that exposes the DRL syntax directly to the user. This isn't for everyone and, in all honesty, should be avoided if possible. The idea that you'd want to code your rules in the online web editor when, as a developer, you've your own development environment<sup>1</sup> isn't realistic. That being said, it can be quite handy to quickly edit a technical rule that a developer has added to your project by using the technical rule editor.

Let's look at what creating a technical rule entails with our HR example from above. The first logic we extracted in section 4.1.1 was:

```
WHEN
    Employee age is 18 or higer
    Document is US passport
THEN
    Do nothing
```

This rule can be implemented as several technical rules by splitting it down into smaller pieces. This is a good practice as it allows each rule you create to capture one specific piece of logic, and when you insert fact you can apply them to a set of rules, each one testing a small bit of business logic.

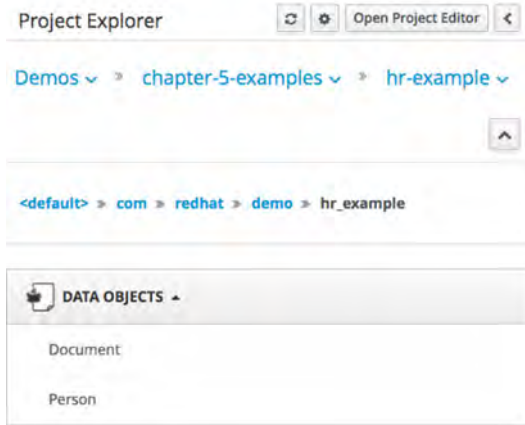
To get started, I provide you with an example project for this chapter that contains a data model, rule artifacts, and all the test scenarios created in this chapter for you to reference. You can examine the completed project artifacts as you read or learn by building them yourself. Install the provided code project (<https://github.com/effectivebpmwithjbosspbm/chapter-5-rules-demo>) for this chapter and note, as shown in figure 4.5, the data model already set up.

If you need to review how to open the Project Authoring perspective, the please review chapter two. The first data object you can review's the Document object, which opens when you click it, looking like figure 4.6.

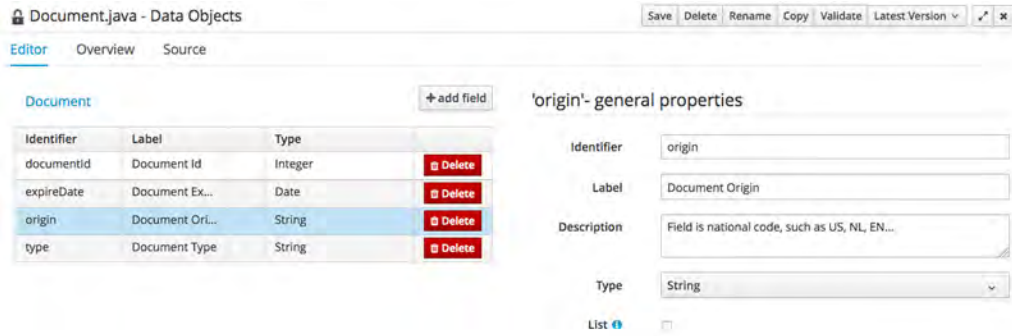
Review the fields available and pay particular attention to the description added in the properties panel for each type to understand their usage. This is going to be a simple data object for our example and contains fields like *Document.orgin* which is an open String. I provide information on the field restrictions in the description to allow you can design rules to validate field data later.

---

<sup>1</sup> When working with JBoss BPM Suite, there's a backing code repository based on Git which is accessed from outside the web console with a developer IDE like Eclipse. Having code completion and other tooling to support you in rule creation from scratch is invaluable. Most developers won't be using the technical rule editor provided by JBoss BPM Suite.

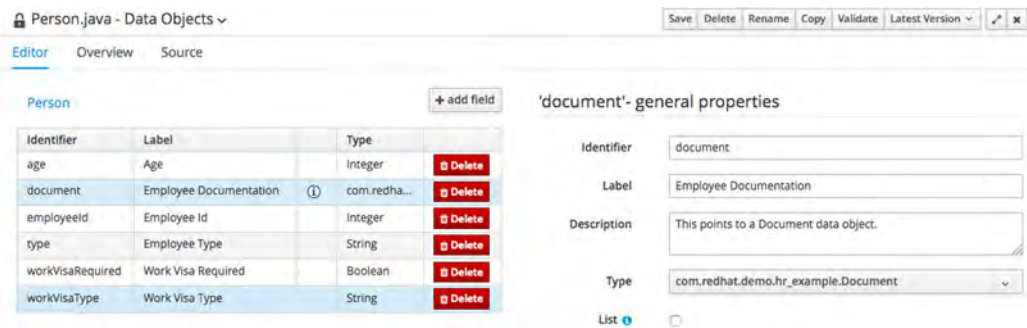


**Figure 4.5** The project explorer here shows you the hr-example project data model found in `com.redhat.demo.hr_example` with a Document and Person object defined.



**Figure 4.6** The Document data object in the HR example.

The second data object's Person and you can open by clicking to review the data fields. Here again, pay attention to the properties panel with the description being used to clarify expected field contents. In figure 4.7 you see the Person data object with the *document* field in the properties panel on the right. This field's of the type Document from figure 4.6 which allows us to embed the Document object inside the Person object.



**Figure 4.7** The Person data object in the HR example.

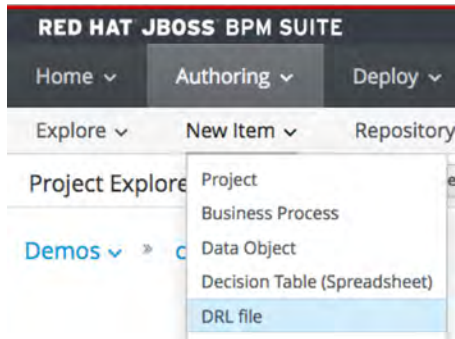
Now I go back to the *default* package level in our project using the link in the Project Explorer on the left. At this level you no longer see the data objects as they're down at the package level `com.redhat.demo.hr_example`.

At this point, before starting to create your first technical rule you might be wondering where to begin? By looking at the two conditions you find the facts you expect and can define what to test in your first rule.

1. Employee age is 18 or higher
2. Document is US passport

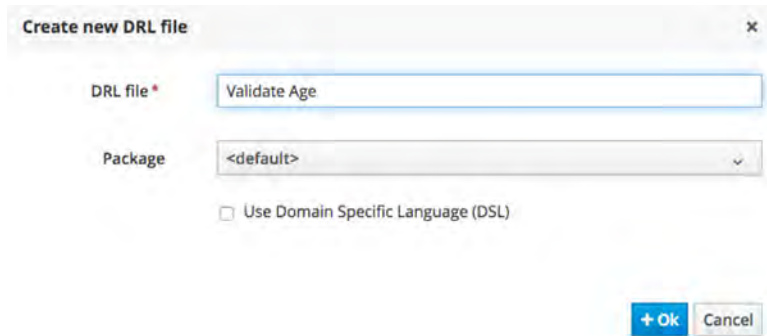
I always want to validate the data given; name the group of rules *validation*. The focus here's on *age* of the person in line one, and on the *origin* of the document in line two. Let's create a few technical rules for the age validation.

The first technical rule or DRL file you create's a field validation rule to ensure that the person object's age field's 18 or higher. If the age's 17 or under, you know right away that that person needs a special visa to work. Click on *New Item*, selecting *DRL file* as shown in figure 4.8.



**Figure 4.8** In your project, select the DRL file from New Item menu to create a new technical rule.

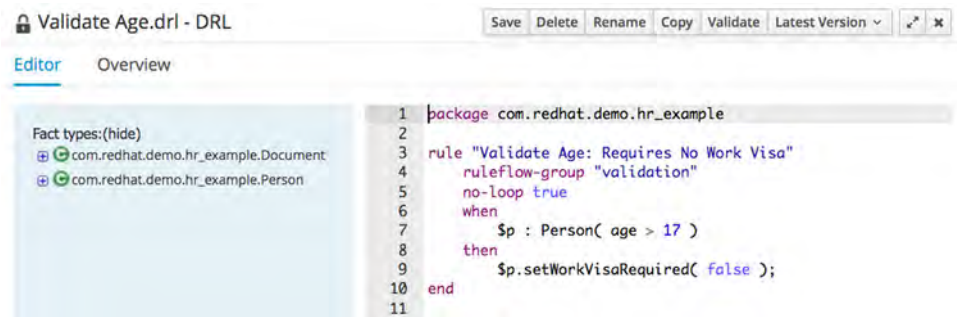
The pop-up that appears asks you to name this new technical rule and which package level to place it in. Enter the details as shown in figure 4.9 and click on the OK button to open the rule in the DRL editor.



**Figure 4.9** The pop-up where you create the new DRL rule file, providing the name *Validate Age* and package set to *<default>*.



You’ve an empty DRL file editor in front of you, which accepts any text you add and provides you with almost no help beyond a single *Validate* button. Enter the package name at the top to ensure your data objects are available. This way you don’t have to use the long package names shown on the left side of the editor. Then enter the rule as shown in figure 4.10.



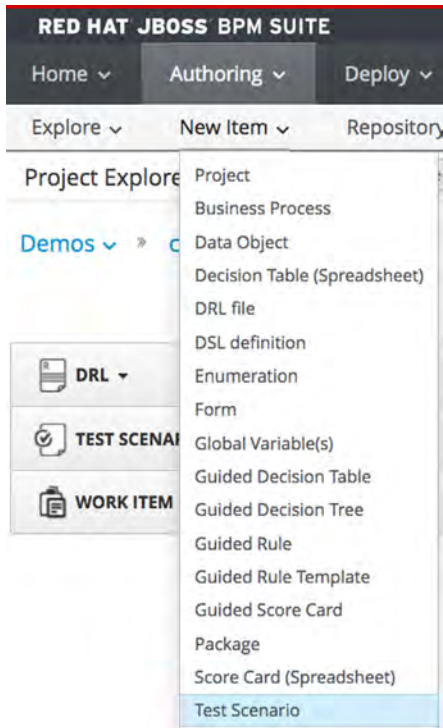
**Figure 4.10** The DRL file editor with the *Validate Age* rule for when the age is 18 or older, no work visa required.

The rule has a name *attribute* and this appears in all reporting such as in a test, which create to validate the rule logic. This rule’s called, “*Validate Age: Requires No Work Visa*”, which might seem wordy but a good practice’s to be expressive in your rule names to ensure that others understand what your rule’s doing. This rule belongs to a group called *validation*. Grouping your rules with the attribute *ruleflow-group* identifies the *validation* rules as the ones you supply with data objects to determine if they’re correct. The attribute *no-loop* is needed to prevent your rule from being re-activated due to changes made to the facts in the condition part of the rule. You can see the person object has a field that gets modified, which would cause our rule to activate again on the changed facts. Finally, you arrive at the heart of the rule, your condition’s that a person needs to have an age of 18 or higher, causing the actions of setting the *work visa required* field to false.

Finally, a good piece of advice when using any of the editors you find in this Business Central console: save your work often. Do this now by clicking on the *Save* button in the DRL editor and filling out the *check in comment* to state that you’ve created a new DRL rule. A pop-up at the top of the screen verifies that your work’s been saved when you click on the *SAVE* button.

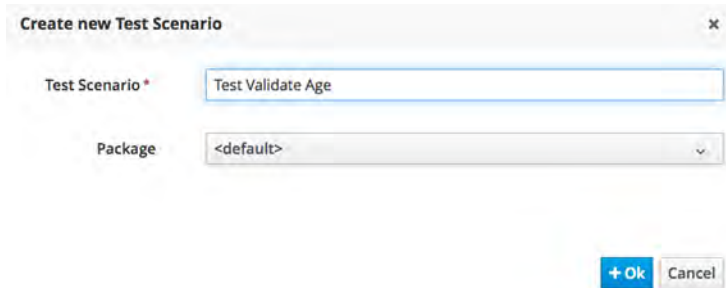
### 4.2.2 Testing a technical rule

Now you’ve your first technical rule, but are you sure it works? To ensure it does, you can create a test scenario by clicking on *New Item*, selecting *Test Scenario* as shown in figure 4.11.



**Figure 4.11** In your project, select the Test Scenario from New Item menu to create a new guided test.

The pop-up that appears asks you to name this new test scenario and which package level to place it in. Enter the details as shown in figure 4.12 and click on the OK button to open the guided test scenario editor.



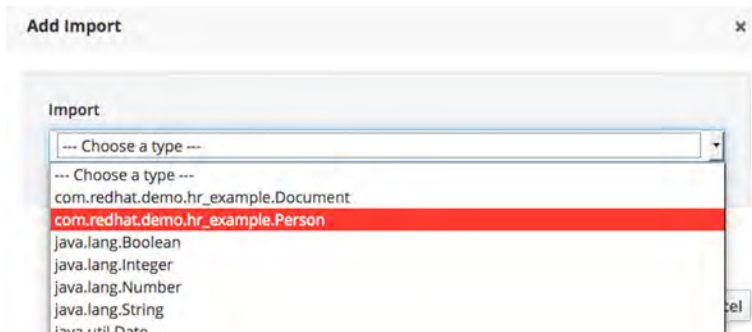
**Figure 4.12** The pop-up where you create the new test scenario, providing the name *Test Validate Age* and package set to *<default>*.

The guided test scenario editor is now ready for you to start implementing the test, populate it with data and setting your expectations for the outcome based on the rule you want to test. You need to start by importing data objects you need for your test; click on the *Data Object* tab to access the import screen as shown in figure 4.13.



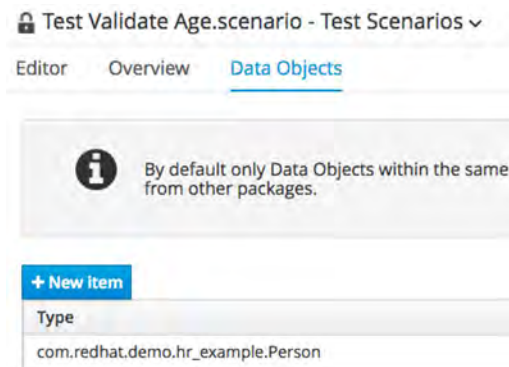
**Figure 4.13** To import data objects from your model to use them in a test scenario, click on Data Objects tab, adding a new item places it in the list that was initially empty.

Click on the *+New item* button. Select the person object from the *Import* menu in the *Add Import* pop-up as shown in figure 4.14.



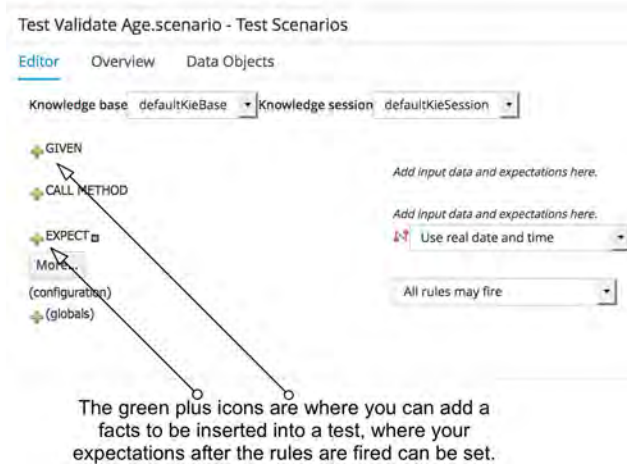
**Figure 4.14** The add import pop-up contains a list of data objects available to you. Select the Person object for use in your test scenario.

After clicking on the *OK* button, the Person object listed as an import's shown in figure 4.15. Now start defining your person fact by returning to the *Editor* by clicking on that tab.



**Figure 4.15** The Person data object's now listed as imported for use in this test scenario.

Test if the employees age's 18 or older, if it is, set that employees *workVisaRequired* to false. The test's only focused on the positive test case where you validate the desired outcome. On the left side of the guided test editor you see green plus icons next to *GIVEN* and *EXPECT* as shown in figure 4.16.



**Figure 4.16** The test scenario waiting for you to define facts to be inserted and the expectations after any specified rules you create have fired.

You start with the *GIVEN* plus icon, clicking it to open a pop-up that indicates what rule flow group you want to be tested. In this case you type in *validation* as shown in figure 4.17.

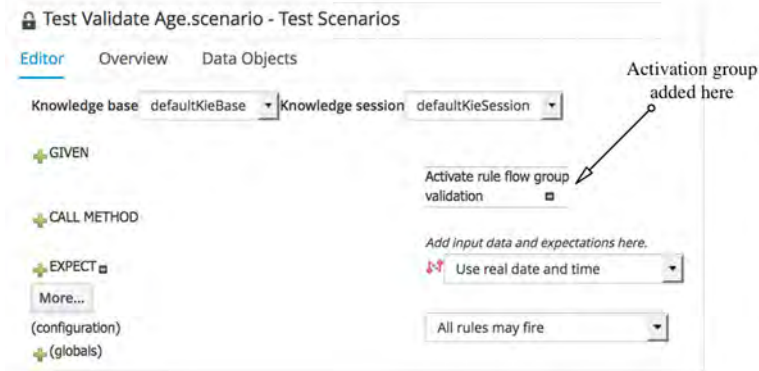


**Figure 4.17** The test needs to be given a rule flow group to provide you with a set of rules to apply your facts to, here *validation*.

When you click on the *Add* button next to the *Activate rule flow group*, you see it added to your test scenario as shown in figure 4.18.

Remember the advice to save your work often, this is a good time to save it with an appropriate commit message using the *SAVE* button at the top of the test scenario editor.

Now you can start defining the data in the form of a *Person* object with the age field set to 18. Clicking on the same green plus icon gives you the same pop-up as before, but now you use the *Insert a new fact* section to select *Person* and give it the *Fact name* as shown in figure 4.19.

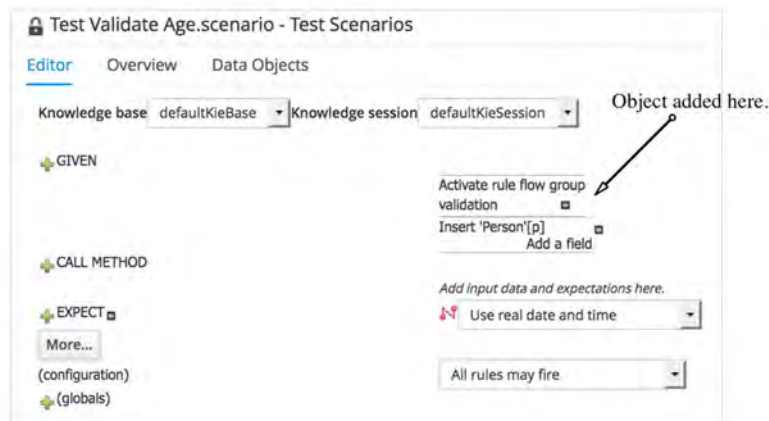


**Figure 4.18** The activation rule flow group's added to the test scenario in the *GIVEN* section.



**Figure 4.19** Inserting a new Person fact with the name 'p' to allow you to reference it later in the test scenario.

The use of a fact name acts like defining a variable. Later in the test scenario, when you want to check if the person's been modified, you reference the fact you inserted. You might be getting the feeling that it's now possible to define multiple person facts and give them different names, allowing you to cover more test cases in your scenario. To insert the person fact, click on the *Add* button and it's inserted into your test scenario as shown in figure 4.20.



**Figure 4.20** The test scenario has the Person object named 'p' added as the fact to be inserted in this test.

Now you add a field by clicking on the *Add a field* text, which displays a pop-up as shown in figure 4.21. You need to find the *age* field in the drop-down menu and click on the *OK* button.

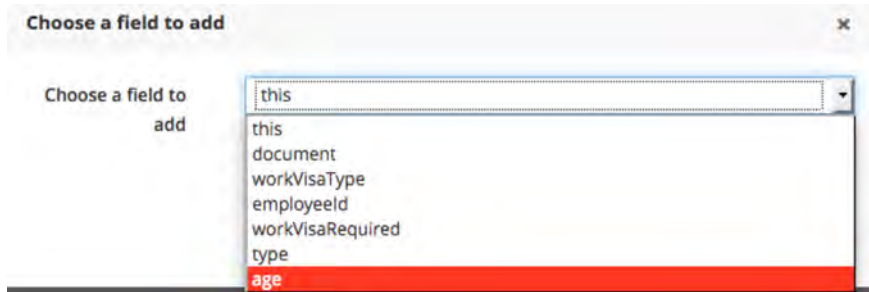
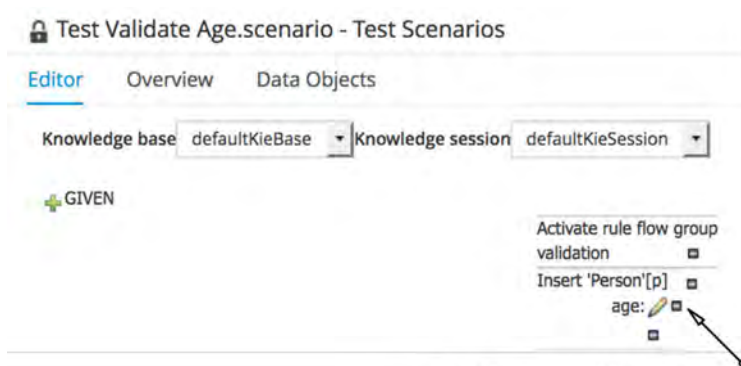


Figure 4.21 When adding a field from the pop-up, select it from the drop-down menu.

This adds it to the person fact with a pencil icon ready for you to click to add a value to the *age* field. In figure 4.22 you see that after clicking on the pencil icon a pop-up appears to define that field.

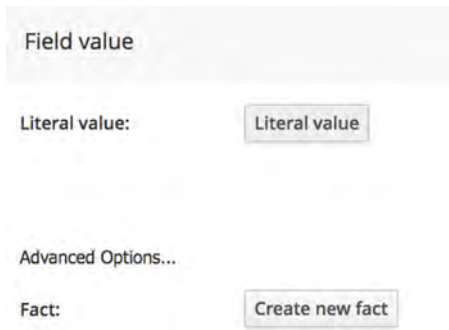


Click on pencile icon to set age value.

Figure 4.22 The age field has been added, now you can click on the pencil icon next to it to define what's assigned to the field.

In figure 4.23 you see that you can click on the *Literal value* button to be able to set a number value in that field.

Add the value 18 to the empty text box next to the *age* field to test this persons' legal working age. The test scenario now looks like figure 4.24 with the facts defined and the rule flow group activation.



**Figure 4.23** To define the age field value as a number, you need to click on the *Literal value* button in the pop-up.



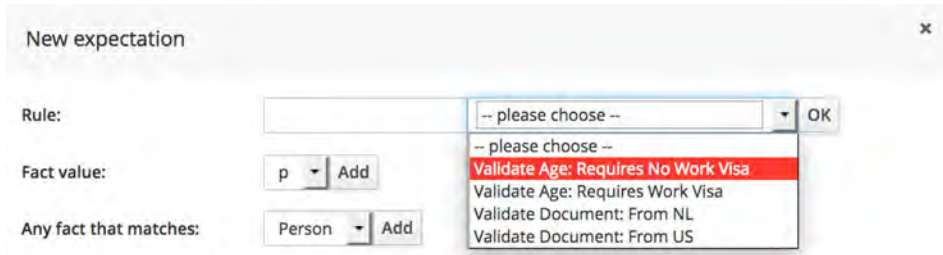
Set value of age to be 18.

**Figure 4.24** The age field's give a text box for you to add the literal value of 18.

Again, save your work with the button at the top of the test scenario editor as this completes the *GIVEN* section of this test scenario. Ignore the *CALL METHODS* for now, this is for advanced actions on facts which goes beyond the scope of our example. You can also leave the menu item *Use real date and time* as it's defined.

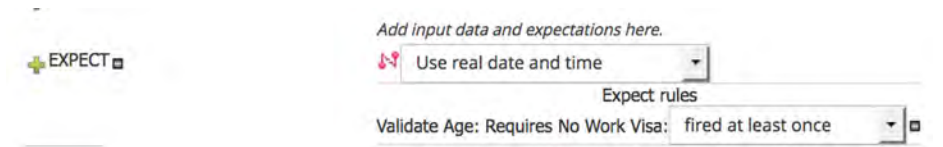
To finish the rule, you need to decide what your expectations are after the facts have been applied to the rule flow group you've set up above. If you remember, when the person given's 18 or older, then the *workVisaRequired* field for the person being evaluated's to be set to *false*. Let's do this by clicking on the green plus icon next to the *EXPECT* section. A new pop-up appears to add a new expectation. The first thing you expect's that your rule fires; select the validate age rule that requires no work visa as shown in figure 4.25 and click on the *OK* button to add it.

At this point you've yet to check if the person's been modified to indicate that no work visa's required, but you can run this test scenario. First save your work by clicking



**Figure 4.25** The new expectation pop-up where you can select the rule you expect to fire with the given facts in this test scenario.

on the *SAVE* button at the top of the test scenario editor, you can't do this often enough to ensure you don't lose work while in a web based editor. Figure 4.26 shows you the status of your test scenario.



**Figure 4.26** Add the rule you expect to fire when the given facts are applied and keep the default rule to fire at least once.

In figure 4.27, you can see the results of clicking on the *Run scenario* button, which runs the test, inserting facts, activating the rule flow group you defined and validating the expectation that the rule fires at least once. A green bar then appears in the *Reporting* panel at the bottom of your screen.

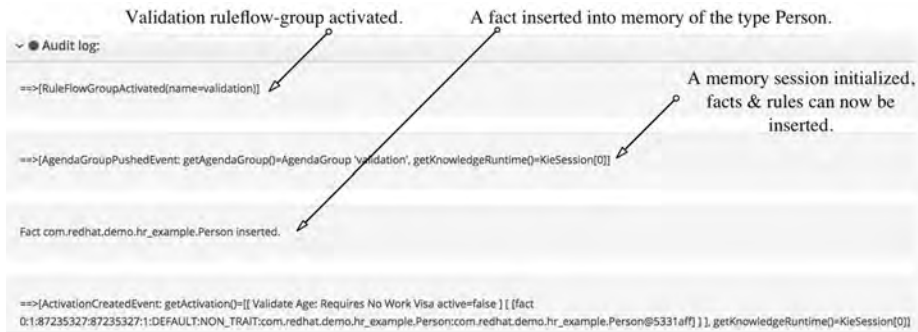


**Figure 4.27** A successful test run appears in the reporting panel at the bottom of your screen.

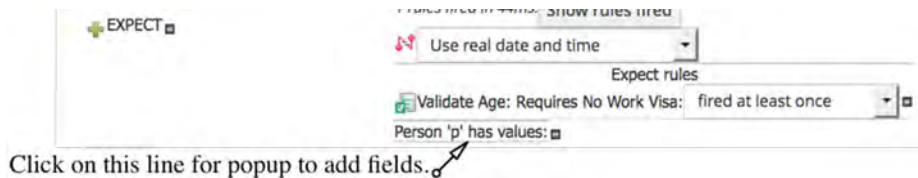
If you want to see what happened in detail, open the *Audit log* by clicking on it at the top of the test scenario. It expands to show technical details around activation of your rule flow group, inserting facts, deleting facts, rule activations and more. See figure 4.28 for an example of your test scenario's audit log after the above test run.

Now let's check if the person object you inserted with name 'p' has been modified to indicate that the work visa isn't needed. Click on the *EXPECT* green plus icon to add a new expectation, this one being a *fact value*. The fact you need to add's our person labeled 'p', which is the only fact available and selected by default in the provided menu. Click on the *Add* button to insert into the test scenario as shown in figure 4.29.





**Figure 4.28** The audit log for a test scenario run shows the rule flow group activation, session being setup, facts inserted, facts deleted, rule firing and more.



**Figure 4.29** The person 'p' has been added to the expectation, but still need to check if the field value for work visa required has been set properly by the rule.

Click on the line *Person 'p' has values* for the pop-up to choose a field to add. In figure 4.30 you see how to select the field *workVisaRequired* from the drop-down menu.



**Figure 4.30** The pop-up to choose a field to add's used now to select the *workVisaRequired* field.

By clicking on the *OK* button, you add the field to your test scenario with the defaults set to *equals true*. Be sure to save the work you've done and in figure 4.31 you can run the completed test scenario by clicking on the *Run scenario* button. Wait a minute, did you get the same red bar at the bottom in reporting and a message stating "*There were test failures*" when you ran your test? The test scenario shows green check

marks next to the parts of the expectations that passed, but yellow warning signs next to the failed expectations and even include what the *Actual* results were.



**Figure 4.31** The test scenario ran with errors. The work visa was expected to be true, but was false. The errors have yellow warning signs.

Fix the test by setting the expectation of work visa required to false and re-run the test by clicking on the *Run scenario* button. Now you can expect the test to produce a green bar in the reporting panel as in figure 4.27. If not, then double check that you didn't make any typing errors, removing any mistakes with the small minus icons next to the fields in the test scenario. Remember to save your work once you finish making changes.

### 4.2.3 Extending the technical rule and test scenario

The rule only covers persons 18 years or older and makes sure that the work visa required field's set to false. What happens if the person's under 18 years old?

What happens is that the person's age field doesn't match the current rule, therefore it doesn't fire and the work visa required field's not set to the needed value of true. To fix this you can add a second rule to the same DRL file as shown in figure 4.32. Once you've typed this into the DRL rule editor, be sure to save your work.

```

1 package com.redhat.demo.hr_example
2
3 rule "Validate Age: Requires No Work Visa"
4     ruleflow-group "validation"
5     no-loop true
6     when
7         $p : Person( age > 17 )
8     then
9         $p.setWorkVisaRequired( false );
10    end
11
12 rule "Validate Age: Requires Work Visa"
13     ruleflow-group "validation"
14     no-loop true
15     when
16         $p : Person( age < 18 )
17     then
18         $p.setWorkVisaRequired( true );
19    end
20

```

**Figure 4.32** Add the second rule to your DRL file as shown here to ensure a work visa's required if the person's under eighteen years old.

Now let's finalize the test scenario by including test facts that ensure a person under 18 years old needs a work visa. To do this you *insert a new fact*, a person, and give it a *fact* name of 'p2' and click on the *ADD* button. This is as you did for the person object 'p'. A second person column's added with 'p2' along with a pencil icon for clicking on to add a *literal value* which inserts a test box. Put the value of 17 in the text box. Save your work and notice that the test scenario can be run producing a green bar in the reporting pane.

What's missing's the expectations for the person object 'p2'; add them by selecting the rule for validate age that requires a work visa. Expect this rule to fire at least once, like the first time you added a rule firing expectation. Then add a fact value of 'p2' by selecting it in the drop-down menu and clicking on the *Add* button as shown in figure 4.33.

The image shows a 'New expectation' dialog box. It has three main sections: 'Rule:', 'Fact value:', and 'Any fact that matches:'. The 'Rule:' field is empty. The 'Fact value:' field has a dropdown menu with 'p2' selected and an 'Add' button. The 'Any fact that matches:' field has a dropdown menu with 'p' and 'p2' (highlighted in red) and an 'Add' button.

**Figure 4.33** Add new fact value p2 from drop down menu as our new expectation for the second person object.

Click on the *Person 'p2'* to choose a field to add to your second person. The *workVisaRequired* field can be found in the drop-down menu, click on the *OK* button to add it. By default, it's set to expect the value true. This completes the test scenario for the *Validate Age* rule, and you should save your work and validate that it now has test coverage for both over and under the age limit for needing a work visa by running the scenario. Figure 4.34 shows you the completed test and scenario run.

Now let's move on to creating guided rules for validating the second part, if you remember, which was that the document needed to be a US passport or a work visa.

#### 4.2.4 **Guided rules for everyone**

Although technical rules using the DRL file editor are possible, they tend to be troublesome when you can easily make a mistake while typing out the rules. Rule design can be achieved by less technically inclined with the *guided rule* editor. It provides you with the same experience for rule design as you encountered when designing your test scenario in the previous section, that of a guided path through your rule creation.

Let's look at what creating a guided rule looks like, but first a reminder of the logic you extracted in section 4.1.1:

```
WHEN
    Employee age is 18 or higer
    Document is US passport
THEN
    Do nothing
```

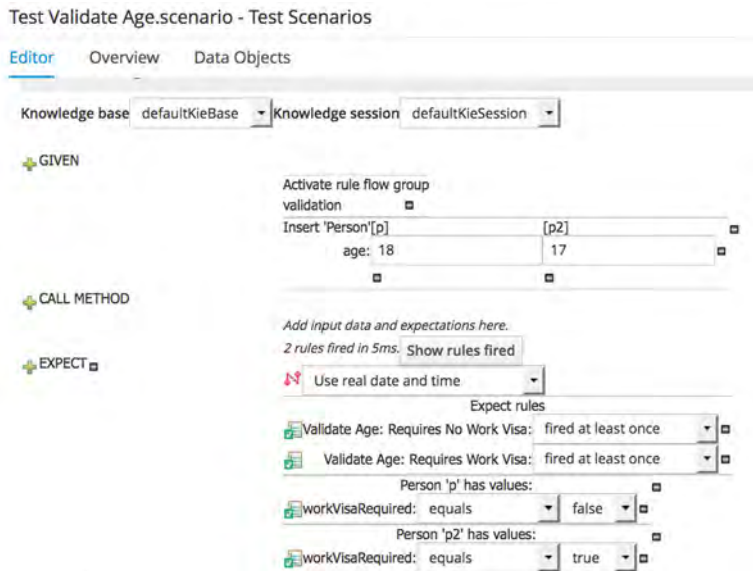


Figure 4.34 Test validate age scenario completed and running successfully.

This rule's to ensure that the document's a US passport, otherwise the employee needs a special work visa. You can continue where you left off in the project and create a guided rule from the *New Item* menu as shown in figure 4.35.

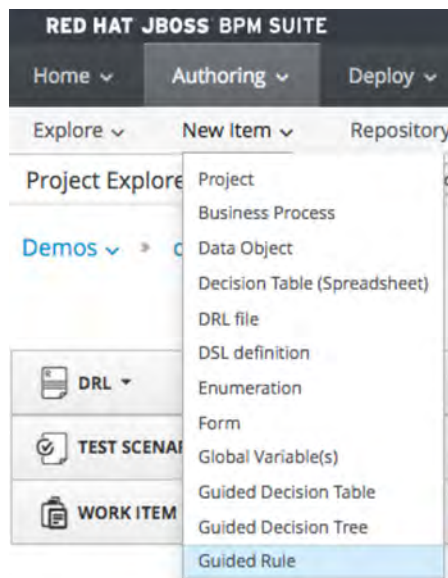
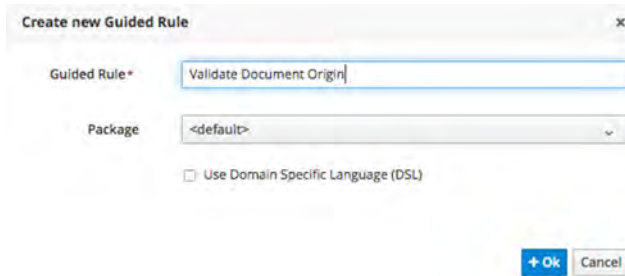


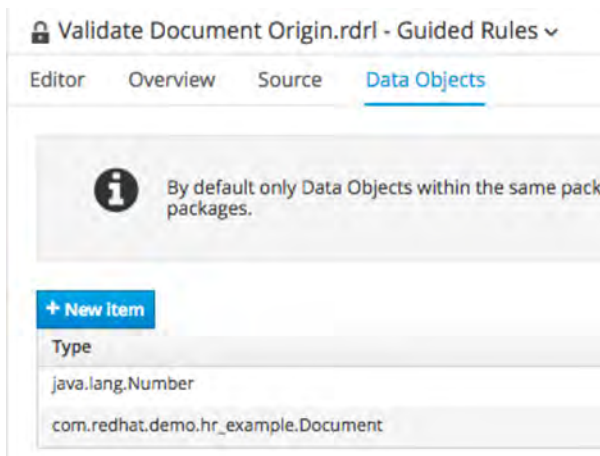
Figure 4.35 Guided rule creation starts in the new item menu in project authoring perspective.

A pop-up appears for you to provide the details to create a new guided rule. Fill in the name for the guided rule as *Validate Document Origin* and leave the package in the default setting as shown in figure 4.36.



**Figure 4.36** The pop-up for creating a new guided rule.

When finished, click on the *OK* button to open the guided rule editor with a blank rule. The first thing you need's to import a Document object for this rule; click on the *Data Objects* tab. Do you recognize it? Yes, it's the same interface you saw while creating the test scenario for your age validation rule. Add an import for the Document object by using the new item button as shown in figure 4.37 and then switch back to the guided rule editor by clicking on the *Editor* tab.



**Figure 4.37** Import a Document object using the new item button like you did in the test scenario.

Save your guided rule and then look at the guided rule which is in front of you. The editor presents you with a *WHEN - THEN* construction and on the far right they both have the same green plus icons you saw in the test scenario editor. Start with the *WHEN* section of your rule by clicking on the green plus icon to generate the pop-up to add a condition to this rule. In the pop-up, select *Document* and click on the *OK* button as shown in figure 4.38.

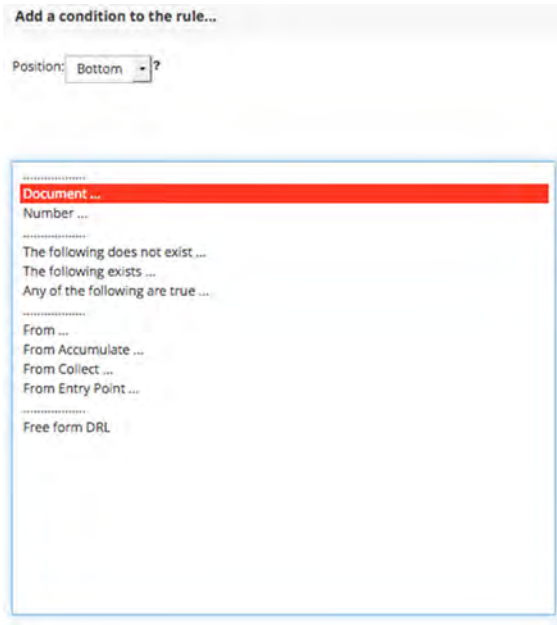


Figure 4.38 Add a condition to your rule by selecting the Document object and click on the OK button.

The guided rule editor has now added a first condition line labeled *I*. Click on the document to open a pop-up for you to modify a constraint. You want to apply a constraint to the origin field of this document which allows you to validate that it originates from the US. Figure 4.39 shows how to select the origin field to start adding a constraint. In the guided rule editor, you see now the origin field added with a drop-down menu next to it in the condition of the rule. When you open the drop-down menu next to the *origin* field it provides you the options for how you can constrain this field.

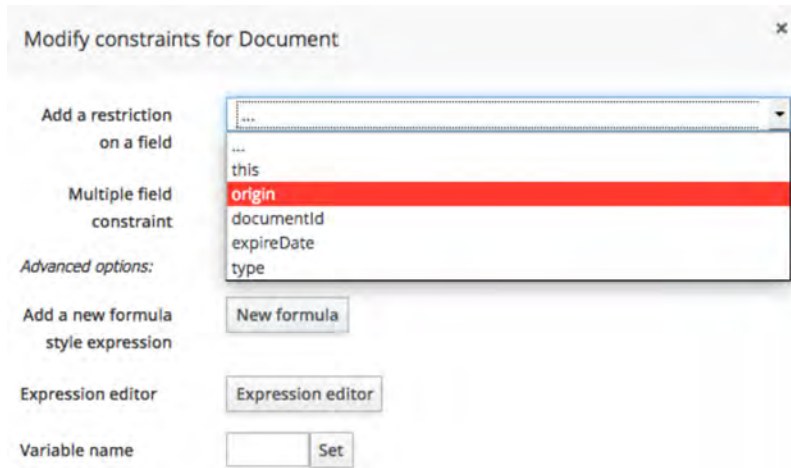
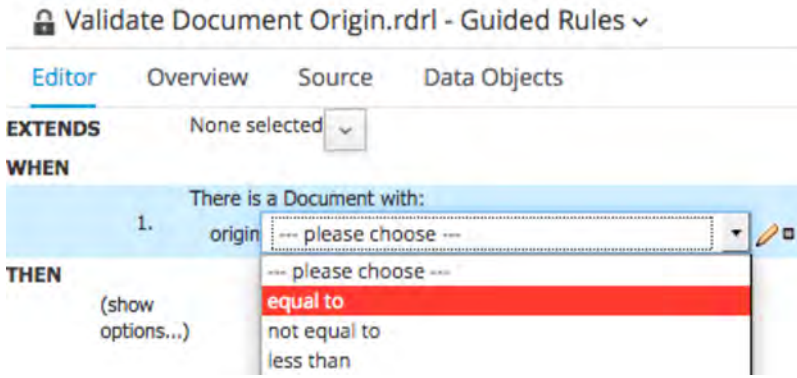


Figure 4.39 The pop-up that you can use to select the field you want to constrain. Selecting the origin field for your Document and the pop-up closes, leaving the origin field added to your conditions section of this rule.

As shown in figure 4.40, select *equal to* from the drop-down menu to constrain the origin field to be exactly the value for a US passport. The final piece of your constrained document origin field's to specify what value represents a US passport.



**Figure 4.40** The document object's origin field now has a pull-down menu. Select as shown the *equal to* constraint.

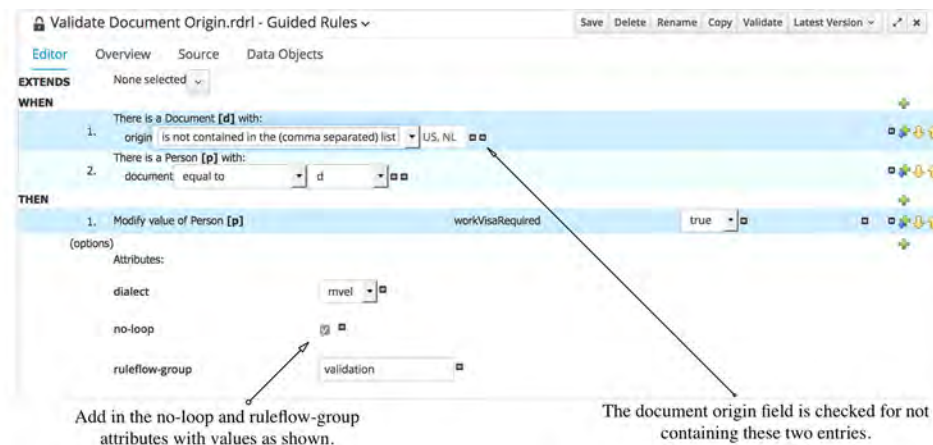
Once the constraint's selected, an empty text box appears for entering the value of *US* as shown in figure 4.41. Now to look for document objects with the origin field set to *US*.



**Figure 4.41** Adding the final value of *US* to the text box means that the origin field of any document needs to be equal to that value.

Now let's take a step back as the requirements have changed a bit to include the Netherlands (NL) as a document origin that also doesn't require a work visa. The only change needed's to modify the document origin constraint by creating a comma separated list that contains only "US, NL" in it. You might think that this rule needs to constrain the origin field such that it's *only contained in the comma separated list*, but you cover more of your domain by doing the opposite as shown in figure 4.42.

Instead of walking you through the exact steps, you've now enough knowledge to make the change yourself. Not to be forgotten, you need to click on the green plus



**Figure 4.42** The adjusted rule where you've constrained the document origin field excluding all values except what's in the comma separated list of US and NL. It's also important to set the options as shown.

icon in the *options* section at the bottom of the rule to add the *no-loop* and *ruleflow-group* with validation options. The rule's now completed and ready for testing to ensure your rule looks like figure 4.42.

### 4.2.5 Testing a guided rule

Nothing different from creating a test for a guided rule exists than the technical rule as shown in section 4.2.2. Because you've been given the ability to create and populate a test scenario previously, I want to provide you with a list of steps to be completed and let you create the test scenario yourself.

Start by adding two sets of facts. The first's to test if a document in the set of US or NL doesn't change the work visa required field value. The second set of facts test if a document outside the set of US or NL causes the rule to fire that changes the work visa required field value.

Steps to create guided rule test

- 1 Use new item menu to create a new test scenario called *Test Validate Document*.
- 2 In the Data Objects tab, add a document and person object.
- 3 In the *Editor tab*, in *Given* section of the test add the following:
  - a Add a document with fact name *d*.
    - i Add the field *origin* to document *d*, set its literal value to *US*.
  - b Add a person with fact name *p*.
    - i Add the field *document* with the value *=d*.
    - ii Add the field *workVisaRequired*, set its literal value to *true*.
  - c In the *Given* section of the test add a second document with the fact name *d2*.
    - i The *origin* field is set to *GB*.



- d Add a person with the fact name *p2*.
  - i The *document* field's set to the value *=d2*.
  - ii The field *workVisaRequired* is set to value *false*.
- e Add a rule flow group activation set to value *validation*.
- 4 In the *EXPECT* section of the test scenario add the following:
  - a Expect *Validate Document Origin* and set this as *fired this many times: 1*.
  - b Expect person named *p* to have the field *workVisaRequired* set to *true*.
  - c Expect person named *p2* to have the field *workVisaRequired* set to *true*.
- 5 Ensure that *all rules may fire*.
- 6 Don't forget to save your test scenario!

Compare your results with the test scenario in figure 4.43 and click on the *Run scenario* button to ensure you get a green test run report.

Test Validate Document.scenario - Test Scenarios Save Delete Rename C

Editor Overview Data Objects

Knowledge base defaultKieBase Knowledge session defaultKieSession

**GIVEN**

Insert 'Document'[d] [d2]  
 origin: US GB

Insert 'Person' [p] [p2]  
 document: =d =d2  
 workVisaRequired: true false

Activate rule flow group  
 validation

**CALL METHOD**

Add input data and expectations here.

**EXPECT**

Use real date and time

Expect rules

Validate Document Origin: fired this many times: 1

Person 'p' has values:  
 workVisaRequired: equals true

Person 'p2' has values:  
 workVisaRequired: equals true

More...  
 (configuration)  
 (globals)

All rules may fire

**Figure 4.43** The final test scenario to validate document origins. It includes two sets of facts which are tested for the positive inclusion of the origin in the expected set of data and the negative.

This chapter introduced you to creating technical rules, guided rules, and showed you how to verify them by writing effective test scenarios. These aren't yet used in a process, that comes later when you start to design your business process. For now, the test scenarios validate that the rules are ready for use by your project.

It hasn't covered all the possibilities with regards to rule creation in JBoss BPM Suite as there remains more to explore after reading this chapter. I hope you continue to use the basic skills taught here to further expand your ability to use rules in the many forms provided by JBoss BPM Suite.

### **4.3 Summary**

- You always need to capture business logic in rules for a BPM project.
- Business logic's used to extract business rules that can be implemented in one of the available rule types.
- Externalizing business logic's done by collecting rules in JBoss BPM Suite that a process can then use.
- Rules need to define the condition that must be met, before the corresponding actions are taken.
- Conditions are defined on the data model based facts, which attempt to match certain fields and values.
- Actions that result from conditions that match a given set of facts can be almost anything, from modifying existing facts to printing a message to a log file.
- The JBoss BPM rule engine's the brain behind business rules execution.
- The rule engine matches facts to conditions, creates an activation agenda and processes the rule actions that need to be taken.
- Technical rules are the rawest form of a rule where you see the actual rule syntax in the DRL editor.
- Technical rule editor's used to write rules in free text editor using the DRL syntax and is mainly targeting a developer.
- Guided rules provide a systematic approach to creating rules without having to be exposed to the underlying rule syntax.
- Guided rules provide an easy way to quickly create business rules, targeting the business level user.
- Test scenarios are used to validate that the rule logic's implemented as desired.

# 5

## *Creating complex business rules*

---

### ***This chapter covers***

- Creating complex business rules
- Implementing domain specific language for use in creating guided rules
- Implementing business rules in decision tables and validating with test scenarios

The previous chapter introduced business logic, rules, and how to implement a few basic examples. The next step's to expand your capabilities with more complex rules and look into how more advanced rule implementation can make your life easier. In this chapter, I take you farther into the world of rules by introducing you to the uses of the following complex rule solutions:

- A domain specific language (DSL)
- A rule designed using a DSL
- A decision table
- Test scenarios for each of the rules to validate correctness.

Although you don't have to read chapter five before this one, it's recommended because I won't be explaining test scenarios or the guided rule editor in as much detail this time around. For example, I focus instead on sharing the knowledge of what the differences are between using the guided rule editor with a domain specific language instead of repeating basic guided rule editor usage.

What's not covered in this chapter are the remaining rule implementation choices found in the *New Item* menu, such as a *Guided Decision Tree*, *Guided Rule Template*, *Guided Score Card* and *Score Card*. It isn't that these lack value, but I chose to highlight the most common rule implementations in this book to cover most situations encountered in your BPM projects. The rest go beyond the scope of this book and are left to the reader to research.

Before you get started on this chapter, I've provided you with an example project for this chapter that contains a data model, rule artifacts, and all the test scenarios created in this chapter as a reference. You can examine the completed project artifacts as you read, or you can learn by building them yourself.

Install the provided code project (<https://github.com/effectivebpmwithjbos-sbpm/chapter-6-complex-rules-demo>) for this chapter and note, the data model's already set up. Feel free to create the rules and test scenarios as I teach you how, or follow along with the completed artifacts as examples in the demo project.

Now let's pick up where you left off working on the same project requirements that are part of your process improvement project around the human resources (HR) department employee onboarding, see chapter four section 4.1 for details. In Figure 5.1 shows the path taken to expand your knowledge of the rules tooling for complex rule implementation. I teach you how to implement a DSL, then how to apply the DSL in the guided rule editor to create a rule and finally how to implement rules in a decision table.

## 5.1 Complex domains as natural language rules

Your journey to create business rules for more complex domains involves, in this section, something known as a DSL. The definition of a DSL's, "*A machine-processable language whose terms are derived from a domain model and that is used for the definition of components or software architectures supporting that domain.*"<sup>1</sup> This definition can be simplified as you design a language which is easy to translate into rules, but remains human readable for experts in the domain. You design a DSL in this chapter for our human resources project that allows a human resource domain expert to design rules in almost natural language, yet the rules can be parsed into a programming language that the rule engine can evaluate during runtime.

The given example requires the use of a DSL to allow a guided rule, using this DSL, to act as a structured design experience for knowledge workers in that domain. Let's look at your example and use the DSL editor to create a DSL based on this chapter's employee onboarding example. This DSL forms the foundations for validating various aspects of data being processed.

---

<sup>1</sup> This definition's as found online (<http://www.yourdictionary.com/domain-specific-language>) and is much broader and a more simplistic translation's presented in this section.



In this chapter you take complex business logic and implement it as business rules in JBoss BPM Suite using the more advanced web tools.

Developer or Architect will implement rules using one of the web tools from JBoss BPM Suite.

Guided rule editor with DSL.

Guided Decision Table Wizard

- Summary
- Inputs
- Add Fact Processors
- Add Conditions
- Add Actions to create Fact
- Add Actions to insert Fact
- Columns to expand

The decision table wizard.

```
Promotions DSL.dsl - Domain Specific Language Definitions
((YRule))? customer age($YAR) <= 30 && $YAR <= 30 && $YAR <= 30 && $YAR <= 30 && $YAR <= 30
((YRule))? Apply Free Shipping($YAR) <= $YAR && $YAR <= $YAR && $YAR <= $YAR && $YAR <= $YAR && $YAR <= $YAR
```

Domain Specific Language (DSL).

Business rules ready when all test scenarios pass.

Free Shipping Promotion DSL Test.scenario - Test Scenarios

Run all scenarios

Test Scenario Overview Config All Test Scenarios

Reporting

Success

Text

To validate the working of your rules, create tests.

Test scenarios are created for each rule to validate logic.



The result is a happy application team able to use the rules in their projects.

**Figure 5.1** Business logic's provided to your project for designing rules. In this chapter, you design complex rules using a domain specific language (DSL), create a rule using the DSL and create a decision table.

### 5.1.1 Domain specific languages to ease rule design

The existence of a DSL can be attributed to the desire of capturing the way a person, in their daily work, talks about their world. DSLs are used in JBoss BPM Suite to create a rule language that captures the problem domain in the words used by people operating in that domain.

Most often, you find the need for an DSL when attempting to abstract away from Java or MVEL<sup>1</sup> based rules syntax. The desire's to present rule construction to a business user in such a way that formulating specific rules happens in almost natural language constructs.

The rule engine takes a set of DSL *definitions*, which are *sentences* that can be converted to DRL<sup>2</sup> rule *constructs*, which is the technical rule language syntax needed by the rule engine. It's another way of saying, let's put a layer of simplicity between the rule syntax and the business user. This is due to business users being the best to understand their own domain language when talking about business rules related to their work.

Most often a DSL's created by a developer, someone with a deeper understanding of the rule syntax to present to the domain experts, who use the natural language supplied to construct their business rules. This doesn't preclude another type of person from constructing a DSL, but it takes a greater understanding of the rule syntax than that person might already have.

A more detailed look at JBoss BPM Suite DSL documentation's found online,<sup>3</sup> but for now I dive right in and help you start creating a DSL.

### 5.1.2 Your first DSL

With the introduction to this section in mind, you've decided to apply a DSL to the following problem in your project. The current technical and guided rules you created have only validated the age in relation to the document origin for employment. It's possible in this project for an employee to be classified, purely on their age, into one of three categories.

- *Standard* – employees who receive the normal amount of vacation days.
- *Senior* – employees who receive extra vacation days after a certain age's reached.
- *Unknown* – employees who've yet to be classified in the system.

A DSL makes this rule one that the HR business owner can manage as the age brackets shift periodically when the laws around employment change. By making

---

<sup>1</sup> MVFELX Expression Language (MVEL) is a dynamically / static typed, embeddable expression language and runtime for the Java platform. As previously mentioned in the introduction to chapter four, the details and syntax for constructing rules using MVEL language can be found in the free online documentation; here's the section on rule syntax supported by the version used in this book: [https://access.redhat.com/documentation/en/red-hat-jboss-bpm-suite/5.4/single/development-guide/#all\\_about\\_rules](https://access.redhat.com/documentation/en/red-hat-jboss-bpm-suite/5.4/single/development-guide/#all_about_rules)

<sup>2</sup> Drools Rule Language (DRL) has been discussed in chapter four section 4.2, refer to that if you need to refresh your knowledge.

<sup>3</sup> Discover all the details behind DSL syntax in the product documentation, [https://access.redhat.com/documentation/en/red-hat-jboss-bpm-suite/5.4/single/development-guide/#sect\\_domain\\_specific\\_languages\\_dsls](https://access.redhat.com/documentation/en/red-hat-jboss-bpm-suite/5.4/single/development-guide/#sect_domain_specific_languages_dsls)

them almost natural language rules with a DSL, the rules can be used by applications as they're maintained by business owners in the form of guided rules.

Let's examine the rules which are applied to the age of an employee in this company to determine their seniority as follows:

- “If the employee's between 18-40 years old, then they're classified as standard.”

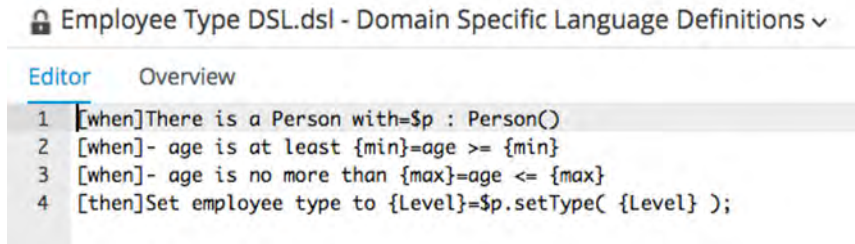
The rule here's based on checking and modifying employee data:

```
WHEN
    Employee age is between 18 and 40 years old
THEN
    Employee type is standard
```

The other age range's 41-65. This rule would look like the following:

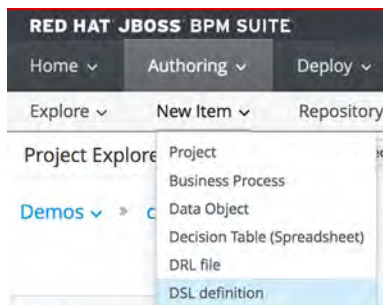
```
WHEN
    Employee age is between 41 and 65 years old
THEN
    Employee type is senior
```

In figure 5.2 you see the completed DSL with each of the rules outlined here represented to allow rule creation to be done using almost natural language. The rest of this section teaches you to create this DSL and explains what each line does.



**Figure 5.2** The completed DSL, consisting of four lines of language to provide the natural language for rule developers to design the rules discussed.

Start by creating the DSL for creating guided rules to cover the above business logic. Select from the menu *New Item* the entry *DSL definition*, as shown in figure 5.3.

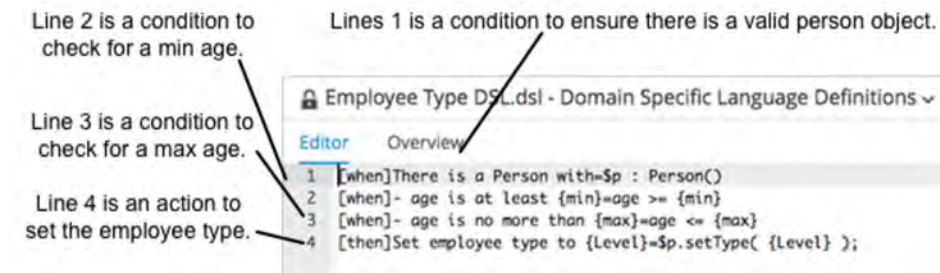


**Figure 5.3** Open a new DSL definition in the *New Item* menu from your project.

The pop-up's presented and you give the DSL definition a name as shown in figure 5.4. When you click on the *OK* button to submit your naming, the DSL editor opens on your screen for you to start creating your DSL.

**Figure 5.4** The popup to create a new DSL definition shown here for you to enter the name *Employee Type DSL* and submit by clicking on the *OK* button.

What you should notice first in figure 5.5 is that this editor reminds you of the technical rule editor in that there's little help in creating a DSL. Indeed, it's a free form text box that lets you type in any text you see fit to define a DSL. You can type these lines into your DSL editor, then using the validate button to ensure the syntax is correct before saving the DSL. Now let's examine each line with the above structure in mind to see how it validates the employee type information in this project.



**Figure 5.5** This is the Employee Type DSL with each line annotated for further discussion in this section.

The basic idea with a DSL's the same as a rule, it has a *WHEN* section with conditions that need to be matched by facts and *THEN* section with actions to be taken when the conditions are met. Each line needs to start with a *WHEN* or *THEN*, followed by a natural sentence that you expect the user of the DSL rule can select. This is followed by an equals sign (=), which starts the condition rule syntax represented by the previous sentence. You can create multiple *WHEN* lines to build more complex conditions and multiple *THEN* lines to cover more complex actions.



The first line in figure 5.5 is a condition, which you know from the *[when]* that starts the line. The following sentence's what you want the rule designer to see, followed by an equals sign that ties the end condition rule syntax to that sentence. The structure looks like this:

```
[when] SENTENCE=CONDITION

SENTENCE: There is a Person with
CONDITION: $p : Person()
```

The second line in figure 5.5 is also a condition, but because it contains a minus sign at the start of the sentence, the rule engine knows it's adding a condition to the line above. The structure can be broken down into this:

```
[when]- SENTENCE=CONDITION

SENTENCE: age is at lease {min}
CONDITION: age >= {min}
```

What's going on with the *{min}* in your sentence? This is how you can put a placeholder for the rule designer using this DSL to enter a value. This value gets assigned to the variable *min*. The variable names are displayed for the rule designer to either adjust or leave it as a default value. Knowing this, the rule's clearer when the age of a person's evaluated for being greater than or equal to the value entered for *min*.

The third line in figure 5.5 is also a condition, one which is added to the first line due to the minus sign at the start of the sentence portion. The structure can be broken down into this:

```
[when]- SENTENCE=CONDITION

SENTENCE: age is no more than {max}
CONDITION: age <= {max}
```

This sentence contains another variable placeholder called *max*, which is used in the rule to evaluate if the given persons' age's less than or equal to the value of *max*.

The final line in figure 5.5 contains the action, which is indicated by a *[then]* element at the start of the line. The structure of the action can be broken down as follows:

```
[then] SENTENCE=ACTION

SENTENCE: Set employee type to {Level}
ACTION: $p.setType( {Level} );
```

This sentence also contains a new variable placeholder called *Level*, which is used in the action portion to set the person's type value. You should have the feeling that it doesn't matter what you put in the *SENTENCE* portion of a DSL line and that you can create the basis of a rule domain in your countries' natural language, be that Dutch, French, Spanish or whatever you need. This is the power of a DSL, to abstract away the rule designing details and constrain your rule designers to only the rule elements you wish to expose.

This completes the DSL and you now have a language that can be used together with the guided rule designer to make it much easier for the HR domain experts to create and maintain employee type logic. Let's see how it's done by designing a couple of guided rules using your DSL.

### 5.1.3 Designing a rule using the DSL

The idea behind your DSL was to have a way to validate an employee based on their age, to determine if they fall into one of two categories. This requires two rules, one for each age range. The first one ensures that employees between 18 and 40 are classified as *Standard* employee types. The second ensures that employees between 41 and 65, which is retirement age, are classified as Senior types.

Creating a guided rule with a DSL's almost the same as creating a normal guided rule as shown in chapter four, refer to section 4.2.4 if you need to revisit how. You start with a new guided rule found in the *New Item* menu. In the popup, you need to fill in a rule name and check the box at the bottom to use a DSL as shown in figure 5.6.

**Figure 5.6** The popup to creating a new guided rule's the same as used before, but now you check the box at the bottom to include a DSL.

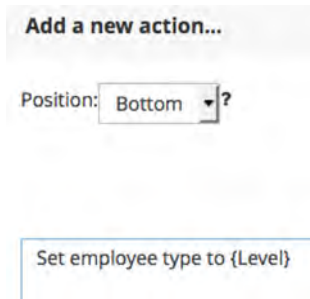
The guided rule editor opens as before; be sure to add a *Person* object in the *Data Objects* editor before returning to the *Editor* tab. Next, click on the WHEN section's green plus icon on the far right and view the DSL language sentences along with the rest of the conditions. In figure 5.7 you see by clicking on the box at the bottom

**Figure 5.7** The popup to add a condition now contains the three sentences from the DSL.

labeled, “*Only display DSL conditions*” that you can only choose from the three DSL sentences you previously created.

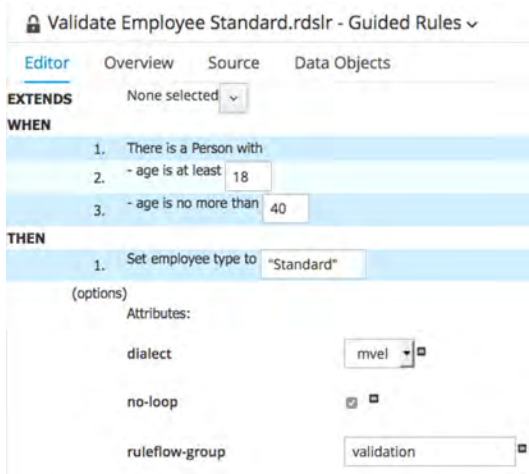
By adding each one in succession to the *WHEN* section of the guided rule you notice that the sentences can’t be edited except for the fields where you inserted variables. Be sure to set the minimum age to 18 and the max age to 40.

Now you can click on the green plus icon for the *THEN* section to generate the popup to create your actions. In figure 5.8 you see the DSL sentence to be added as an action. By selecting the sentence, it’s added to the actions with a field where the variable *Level* was inserted. Please modify this field to contain the string value “*Standard*” as this is the type of employee that fits the age range.



**Figure 5.8** The popup to add an action now contains the sentence from the DSL.

This almost completes the guided rule with DSL for the standard employee type, but before you validate and save the rule you need to add the options of *no-loop*<sup>1</sup> and *ruleflow-group*<sup>2</sup> set to *validation*. When you’ve completed the options, make sure it matches figure 5.9 before saving and moving on to create the second and final guided rule with DSL.



**Figure 5.9** The completed Validate Employee Standard guided rule using the DSL.

<sup>1</sup> If you need to refresh your knowledge of what *no-loop* attribute’s for, see chapter four section 4.2.1 where it was first introduced.

<sup>2</sup> If you need to refresh your knowledge of what *ruleflow-group* attribute’s for, see chapter four section 4.2.1 where it was first introduced.

The second guided rule with DSL looks a lot like this first one, but you need to change the age ranges and action variables. The age range needs to be 41 to 65 and the action sets the employee type to “*Senior*.” Let’s see if you can create this guided rule with DSL without my help, but make sure it matches the one shown in figure 5.10 before you validate and save it.

**Figure 5.10** The completed Validate Employee Senior guided rule using the DSL.

This completes the rules needed to cover validation of employee types based on their ages. To prove this, you need to create a test scenario that covers a few cases to ensure all the age ranges are satisfied. Having become an experienced rule maintainer, creating a test scenario should be almost second nature. I provide only a few hints; name the test scenario *Test Validate Employee Type*, ensure you import a *Person* data object from the *Data Objects* tab, create four-person facts to test ages 17, 18, 41 and 66. Activate the *validation* ruleflow-group. Also make sure you set a person type initially to *Unknown*, as you expect each one of the ages to result in employee type being set respectively to *Unknown*, *Standard*, *Senior* and *Unknown*. Finally, allow all rules to fire, but expect each of your *Validate Employee* rules to fire only once.

When finished you should get a green bar stating that the test scenario ran successfully. You can inspect the fired rules, as there are rules in your project that don’t apply to this test. Click on the *Show Rules Fired* button to expand the list.

Figure 5.11 shows you a completed and working test scenario to validate the rules around employee type checking. Don’t forget to save your test scenario before closing.

This completes the tour of DSL usage in guided rules. This isn’t an all-encompassing look at what’s possible with DSL’s, but it should give you a solid foundation of how they work and what you can achieve. Next’s a look at what you can do with guided decision tables and spreadsheets as a form of rule processing in your project.

Test Validate Employee Type.scenario - Test Scenarios Save Delete Rename Copy Run scenario Run all

Editor Overview Data Objects

▼ Audit log:

Knowledge base: defaultKieBase Knowledge session: defaultKieSession

➤ GIVEN

Insert 'Person'[p]	[p2]	[p3]	[p4]
age: 17	18	41	66
type: Unknown	Unknown	Unknown	Unknown

Activate rule flow group validation

➤ CALL METHOD

Add input data and expectations here.

6 rules fired in 12ms: Show rules fired

➤ EXPECT

Use real date and time

Expect rules

- Validate Employee Standard: fired this many times: 1
- Validate Employee Senior: fired this many times: 1

Person 'p1' has values:

- type: equals Unknown

Person 'p2' has values:

- type: equals Standard

Person 'p3' has values:

- type: equals Senior

Person 'p4' has values:

- type: equals Unknown

More... (configuration) (globals)

All rules may fire

Figure 5.11 The completed test scenario which ensures all ages are validated for employee type.

## 5.2 Complex rules made easy with decision tables

Sometimes you've a more complex set of business rules that you want organized in a way that allows you to oversee the rules in a single view. The problem with technical rules and guided rules is that you end up with a lot of rules in different files with a use case like this. The decision table's a web based spreadsheet format where each row of the table represents a rule, providing you with a single view of multiple and often complex rules.

It's often the case that in the business logic discovery phase, you find that the business owner uses a spreadsheet to keep track of pricing, products, ratings or some combination of logic that determines how they go about their daily work. It's also possible to plug in spreadsheets, when in the correct form, into your project to allow the business owner to continue working in the manner that she's accustomed to. It goes beyond this book to cover how to use existing spreadsheets in your projects, instead you can explore an example online.<sup>1</sup> I take you through an example in the HR example allows you to create a guided decision table in JBoss BPM Suite to implement a set of complex rules.

### 5.2.1 Guided decision table wizard for complex rules

Imagine that for each employee, based on age and the need for a work visa, you need to create a set of rules to determine which of the four categories an employee needing a work visa fits into.

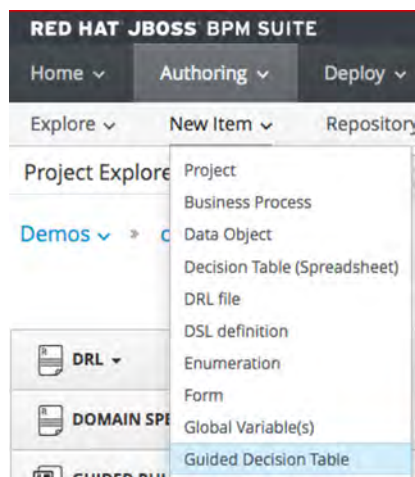
Work visa categories

- Type A – employee work visa needed and age's between 18 – 25
- Type B – employee work visa needed and age's between 26 – 35
- Type C – employee work visa needed and age's between 36 – 45
- Type D – employee work visa needed and age's between 46 – 65

Four rules are in the initial decision table. There might be even more categories in the future or the age patterns might change. To provide for flexibility in a large or possibly large range of data in a rule set you can put the rules into a decision table. The basic rule that needs to be represented in each row looks like this:

```
WHEN
    Employee age is between MIN and MAX
    Employee requires a work visa
THEN
    Set Employee work visa type to TYPE
```

The values for *MIN* and *MAX* are different for each row in the decision table and represent the conditions to be met along with the requirement for a work visa. The values for *TYPE* are different for each row and represent the action of setting the employee field work visa type. Let's get started by using a guided decision table wizard that you find in the *New Item* menu under *Guided Decision Table* as shown in figure 5.12.



**Figure 5.12** Select the Guided Decision Table's created from the New Item menu.

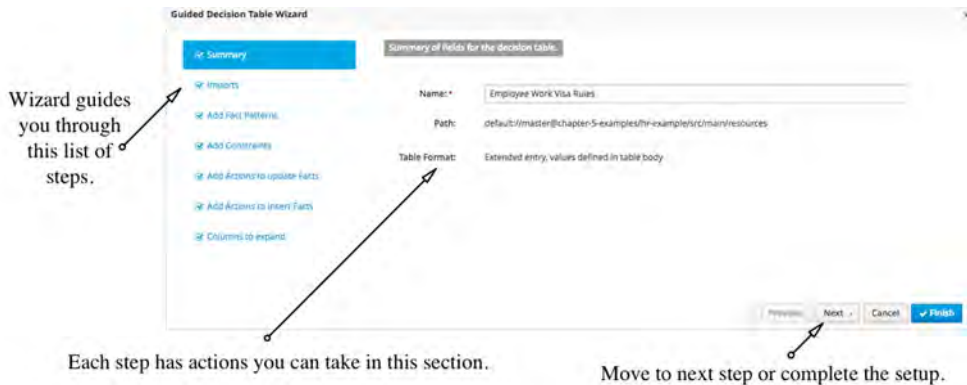
<sup>1</sup> A complete project example that used an external spreadsheet to validate zip codes, known as the JBoss BPM Baggage Delivery demo, can be found at (<https://github.com/effectivebpmwithjbossbpm/bpms-baggage-delivery-demo>).

A popup appears for you to name the decision table, select the default package as its location, ensure that both the *Use Wizard* box and the *extended entry* box are checked before clicking on *OK* as shown in figure 5.13.

**Figure 5.13** Create a new guided decision table starts with this popup.

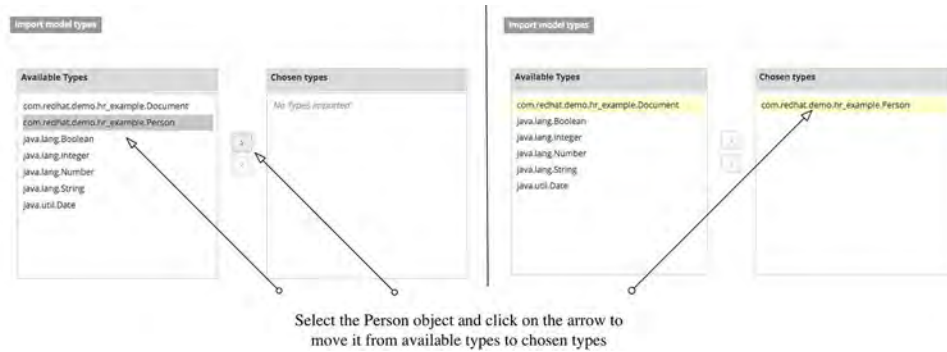
You're presented with the *Guided Decision Table Wizard* which includes a check list that guides you through the process of creating your decision table. The first step's labeled *Summary* and it's filled in with the values you selected from the previous popup. Note the *Next*, *Previous*, *Cancel* and *Finish* buttons on the bottom right. Be careful to use the next and previous buttons and don't accidentally click on the finish button. You can't get back to this guided wizard once you click on the finish button.

In figure 5.14 you see the initial summary page of the guided wizard. Click on the next button to move down the list to *Imports* and on the right side find the fields to create imports, conditions, actions and more.



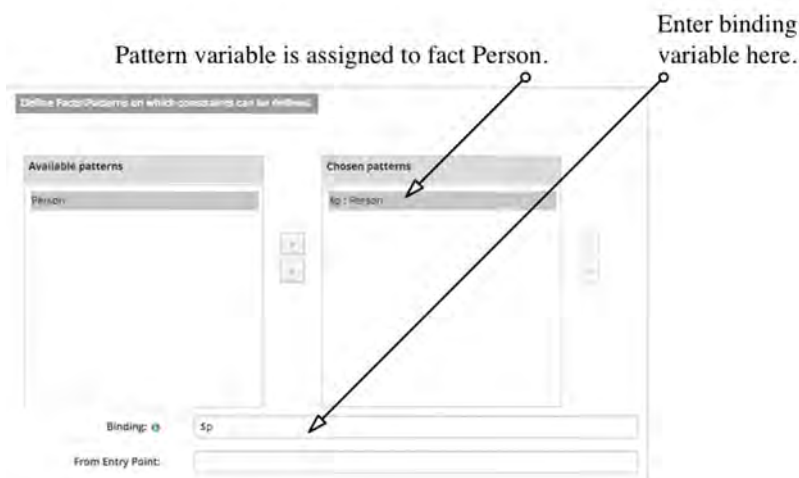
**Figure 5.14** The guided decision table wizard for step by step building of complex rule sets.

Starting with the imports, shown in figure 5.15, you see that you can select any of the available data objects for your project. You need the person object, and you can select this, click on the right arrow to insert it, and then click on the next button to add your facts.



**Figure 5.15** Adding imports step in guided decision table wizard, where you select the data object to be imported and use the arrows to move to the right column to make it available for your decision table rules.

The step to add fact patterns consists of you selecting your person object, using the arrow to move it to the right window, selecting the person object and binding it to the variable as show in figure 5.16. Note that after you type in the variable name, hit enter and it appears next to the person object in the chosen types window. Click on next to start adding constraints.



**Figure 5.16** Assigning a binding variable name to reference a fact.

You see three windows with your person object on the far left under *available patterns*. If you select this person object, all its available fields are shown in the middle window entitled *available fields*. When defining the incoming fact constraints for each



row in the decision table, you've a minimum age, a maximum age, and check that the work visa required field's marked true. To do that, select age field and click on the arrow to add it to the far-left window labeled *conditions*. Do this twice to create two age fields to assign conditions and for work visa required field.

Once added, you can select each field and underneath the windows the text fields are shown to provide a column header description. Select the operator that constrains that field from the drop-down menu. For the age fields, select each one, fill in the following two fields and ignore the rest.

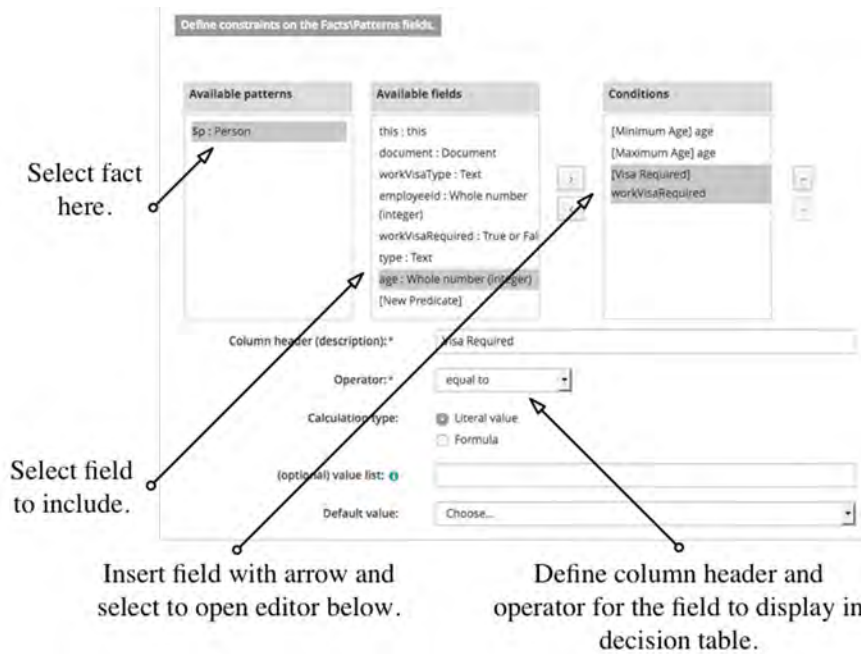
Min Age

- Column header: Minimum Age
- Operator: greater than or equal to

Max Age

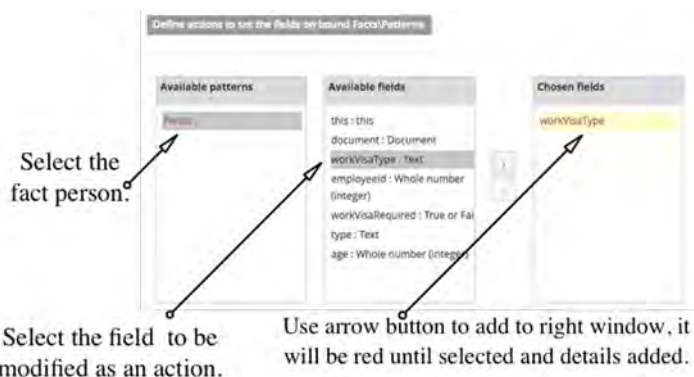
- Column header: Maximum Age
- Operator: less than or equal to

For the work visa required field see figure 5.17, which shows the end results for all three fields. Note that all fields added as conditions are red until you provide the necessary details, at which time they turn black. Click on next to start adding actions.



**Figure 5.17** Define constraints on the fact fields, these then appear in the final decision table to create rules.

Defining actions to be taken's the same process as you completed. Choose the person object to display its fields in the second window, choose the field *workVisaType* if your rule conditions are met, and use the arrow button to add it to the *chosen fields* window. As shown in figure 5.18, it's now red and you must select it to start filling in the column header, but leave the rest of the fields blank.



**Figure 5.18** Define actions by selecting the object, then the field you want the action to take place on. The chosen field, *workVisaType*, remains a red color until selected and the details are filled in.

Click on next to move to the add actions step, which you don't need, and click next to move to the last step in the guided decision table wizard. You want the table to be fully expanded to ensure that the provided box is checked, and click on the finish button to see your decision table appear in the editor.

### 5.2.2 Finalize decision table with rows of rules

Now that your decision table's in front of you, you can expand it to view the details. It's an empty decision table, as you've yet to define any rows which are your rules. Notice the *Condition columns* and *Action columns* are filled with your pre-defined values, ready for your rules to be added in the rows.

The rules you want to add can be found at the beginning of section 5.2.1. Fill in the rows and your decision table should look like the one shown in figure 5.19. You can do this by clicking on each field in turn, then filling in the text, number or checking the box.

One thing's left to do before you can start working on a test scenario to validate the workings of your decision table; you must add the options *no-loop* and define a *rule-flow-group*. To do this click on the green plus icon found under the *decision table* label which produces the *add a new column* pop up. Select the column type *add a new column* entry as shown in figure 5.20 and click on *OK* button.

Employee Work Visa Rules.gdst - Guided Decision Tables Save Delete

Editor Overview Source Data Objects

All the rules inherit: None selected

Decision table

+ New column

Condition columns

Person [\$p]

- Minimum Age
- Maximum Age
- Visa Required

Action columns

Work Visa Type

(options)

Add row... Otherwise Audit log

	#	Description	Minimum Age	Maximum Age	Visa Required	Work Visa Type
+ □	1	Youth Visa	18	25	☑	A
+ □	2	Adult Visa	26	35	☑	B
+ □	3	Mature Visa	36	45	☑	C
+ □	4	Senior Visa	46	65	☑	D

**Figure 5.19** The decision table with the four complex rules entered as rows, defining what age span requires which visa type.

Add a new column

Type of column:

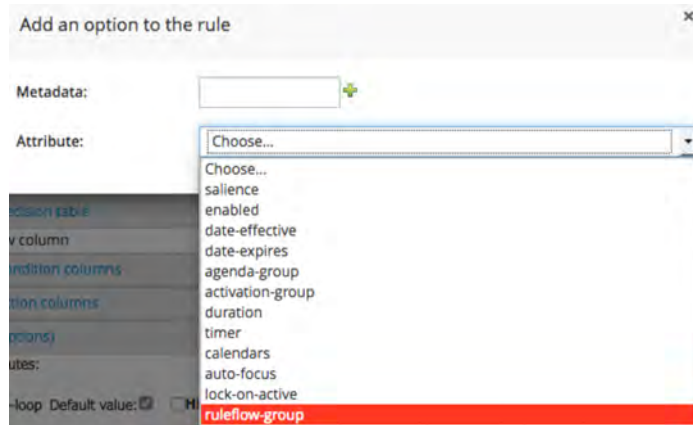
- Add a new MetadataAttribute column
- Add a simple Condition
- Set the value of a field
- Set the value of a field on a new fact
- Delete an existing fact

Include advanced options

+ Ok Cancel

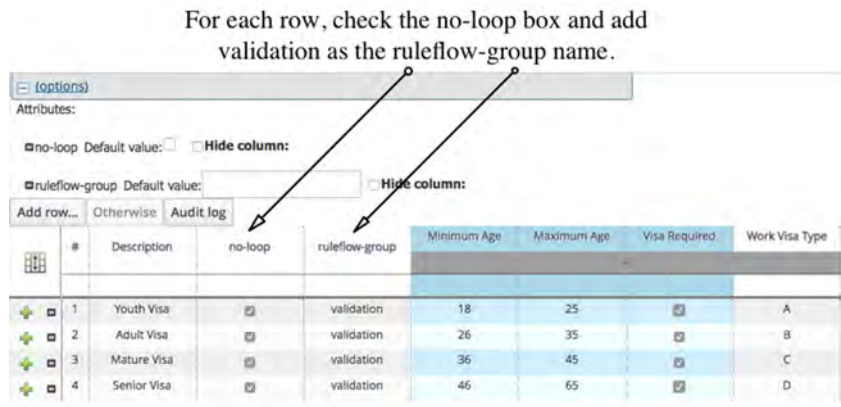
**Figure 5.20** Add a new column popup's used to add options to your guided decision table.

You want to add the two attributes; add the *no-loop* option before coming back around again to add the *ruleflow-group* option from the pull-down menu as shown in figure 5.21.



**Figure 5.21** Add attributes to your decision table from the drop-down menu, such as *ruleflow-group*.

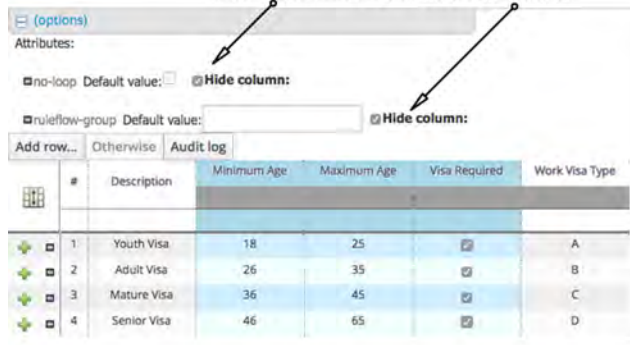
Back in the decision table editor you find the options added and displayed as column entries in your decision table. You can put in default values under the options tab, but for clarity you should put them in each rule row as shown in figure 5.22.



**Figure 5.22** There attribute values should be filled in for each row and for each rule in the decision table.

As the options aren't contributing to the readability of your decision table, you can make use of the option to check the boxes labeled *Hide Column* to remove the columns from your view as shown in figure 5.23.

Check the hide column box for no-loop and ruleflow-group to hide their columns in the decision table.



**Figure 5.23** To enhance readability of the decision table, hide the attribute columns.

Be sure to validate and save your decision table. Only one task's left; creating a test scenario to exercise the correctness of this set of rules. This test scenario's called *Validate Employee Work Visa Rules* and can be found in the chapter project. It's left to you to create this based on the knowledge you've acquired building previous test scenarios.

This chapter has taken you through creating a DSL, using a guided rule which uses that DSL and implementing rules in a guided decision table. It hasn't covered all the possibilities with regards to rule creation in JBoss BPM Suite, as there remains more to explore after reading this chapter. I hope you continue to use the basic skills taught to further expand your ability to use rules in the many forms provided by JBoss BPM Suite.

### 5.3 Summary

- DSLs simplifies a problem domain, making rule creation like designing natural language rules.
- Conditions and actions are the building blocks in a DSL like in simple guided rules.
- Rules become easier to understand when building them using a DSL.
- Decision tables help when rules contain data or ranges of data that need to be flexible or modified often.
- The overview of a large rule set's easier when the rules are captured in a single decision table.
- The need for rule validation remains no matter what type of rules are created; even complex rules should be tested using scenarios.

## **Symbols**

---

- \*.drl postfix notation 64
- +New item button, technical rule testing 70
- +OK button 25
  - creating new data object 41

## **A**

---

- ACME Travel Data Model, example project 54–57
- Action columns, decision table 101
- actions 61, 85
- Actual results, technical rule testing 77
- Add a new tag button 47
- Add button 56
- Add button, Organizational Unit Manager 25
- Add from repository button 56
- Administration item, Authoring menu 22
- administration perspective 22
  - project organization 23
  - setting up organizational structure 24–25
  - starting 22–23
- agenda, described 64
- Apache Software Foundation 10
- application, changes to business logic and 7
- Approve Reward, user task 6
- Artifact Repository view ??–54
- Artifact repository view 54
- artifacts, default list 54
- asset manager 15
- Associated repositories window 25
- Audit log, technical rule testing 75
- Authoring menu 22, 28, 54
  - data modeling 39

- automation
  - and potential to improve business 4
  - and process improvement 2
  - and removal of inconsistent behavior 4
  - and tradition human brain power 3
  - BMP suite and integrating services in organization 9
  - business value and 3
  - fully-automated business process 5
- Available repositories window 25

## **B**

---

- BAM (Business Activity Monitoring) 16
- behavior, inconsistent, and process automation 4
- BMP suite
  - when not to use 10
- BPM (Business Process Management) 1
  - basis of 2
  - introduction to BPM concepts 2–7
- BPM analysis tooling 15–16
- BPM project
  - basic building blocks for 7
  - supporting components 17
- BPMN (Business Process Modeling Notation) specification 4
- business activities, aspects that support integration of 1
- Business Activity Monitoring. *See* BAM
- business behaviors, process measurement and 4
- Business Central 13, 22
  - data modeling 39
  - home screen 40
  - logging into 22

- business events 1, 8
- business events engine 13
- business knowledge, and rules that
  - encapsulate 59
- business logic
  - externalization 61, 85
  - Human Resources (HR) department,
    - example 60
    - data object 65
  - Human Resources department, example ??–61
  - identifying rules in code 61
  - in organization 60
  - technical and guided rules 60, 64
  - traditional application development and 7
  - workflow 59
- business logic, complex 88
- Business Process Management. *See* BPM
- Business Process Modeling Notation. *See* BPMN  
(Business Process Modeling Notation)
- business processes 1
  - basic series of events for process definition 4
  - bottleneck, example of 3
  - different uses of 9–10
  - example of developing process definition 4–5
  - human involvement 5
  - human resource department, example of 4
  - ideal 5
  - in daily business 2
  - overview 9
  - process diagram 4
  - selection and business improvement 4
  - STP (Straight Through Processing) 5
  - when not to use BMP suite 10
- Business Processes package 34
- business resource planner engine 13
- business rule management systems 8
  - and powerful use of business events 9
- business rules 1
  - and indicators of logic that can be extracted
    - as 8
  - and maintaining consistency across
    - applications 8
  - application of 8
  - centralizing in JBoss BPM Suite 60
  - defined 7
  - evaluation 8
  - externalization 8
- business rules engine 13
- business rules, complex 86
  - decision table *See* decision table
  - example project 87
  - implementing a set of complex rules,
    - example 97–101

- business value
  - defining a process and 4
  - described 3
- business, driving forward and 2

## C

---

- Cancel button, guided decision table wizard 98
- case-statements 8
- certified and supported configurations 12
- Clone repository, Repositories menu 26
- cloud, JBoss BPM and running in 20, 34
- configuration decisions, various 20
- communities 1
- community projects 18
  - Apache Software Foundation 10
  - examples of 11
  - integration and maintenance 11
  - upstream 10
- competitors, organizations constantly tested by 2
- conclusion, defined 61
- Condition columns, decision table 101
- conditions 61, 85
  - building more complex 91
- containerized installation 36
  - Docker platform and 20
  - steps to generate 20–21
- Copy, data modeler button 43
- Create and continue button, adding fields to data
  - object 44
- Create button, adding fields to data object 44
- Create new Project pop-up 29–30
- credit card transactions business rules 8
- customer evaluation process, example of STP
  - process 6
- customers, shifting expectations of 2
- Customize view button 30

## D

---

- data model
  - described 37
  - example
    - Department data object 50–53
  - external 53
    - importing 54
    - Java archive 53
  - imported external
    - using 55–57
- data modeler 15
  - adding data fields 41

- Overview tab 47
  - providing necessary details 40
  - data modeling
    - adding fields to data object 43–46
    - adding identified fields 42
    - Business Central console 39
    - creating new data object 40–41
    - data model editor 42
      - data object's initial state 42
    - data model example 38
    - data model source 49–50
    - formal definition 38
    - getting started 38
    - menu bar buttons 43
    - overview 38
    - Project Authoring perspective 41
  - data object
    - +add field button 43
    - Complete button and field submission 45
    - creating new 40–41
    - detailed information in Overview tab 47
    - editing by hand 50
    - example project 38
    - fields in 58
    - getter and setter methods 49
    - Metadata tab and extra information about 48
    - New Field pop-up 44
    - put in packages 41
    - taking away locked object 48
  - Data Object, New Item menu 40
  - data Objects tab, guided rule editor 80
  - data, as building blocks for process project 37
  - decision table
    - adding options to 101–102
    - and flexibility in a large range of data 97
    - defining fact constraints 100
    - enhancing readability of 103
    - Extended entry box 98
    - finalization 101
    - guided decision table wizard 97–101
    - MIN and MAX values 97
    - overview 96
    - Use Wizard box 98
    - validation and saving 104
  - decision table editor, viewing added options 103
  - decision tables 14
  - default settings, JBoss BPM Suite installation 19
  - Delete, data modeler button 43
  - demo container, steps for generating
    - containerized installation 21
  - Department data object, example of data
    - modeling 50–53
  - Department data object, example project 38
  - dependencies
    - adding 56–57
    - details, filling in manually 56
  - Dependencies view 55
  - deployment management 15
  - deployment repository 17
  - developers
    - and DSL 89
    - technical rules for 65–68
  - development repository 14, 17
  - Docker platform 20
  - domain specific language *See* DSL
  - domains, complex, as natural language rules 87
  - Download button, Artifact repository 54
  - Download Project menu item 33
  - Download Repository menu item 33
  - DRL (Drools Rule Language) 64
  - DRL file *See* technical rules, for developers
  - DRL rule constructs 89
  - Drools projects 10
  - Drools Rule Language. *See* (DRL)
  - droolsjbpm-knowledge, community project 11
  - drools-website, community project 11
  - DSL (domain specific language)
    - and creating the basis of rule domain in
      - country's natural language 92
    - defined 87
    - designing a rule using 93
      - guided rule editor 93
      - no-loop and ruleflow group 94
      - validation and saving 94
    - example 89
      - completed DSL 90
      - creating new DSL definition 91
      - multiple WHEN lines 91
      - rules applied 90
      - variable names 92
      - when to use 89
  - DSL definition, New Item menu 90
  - DSL editor 91
    - similar to technical rule editor 91
- ## E
- 
- Eclipse 14
  - Editor tab 80
  - Employee data object, example project 38, 41
  - employees, hiring of, automated processes and 3
  - Enable Tag filtering, view configuration 33
  - end node, process diagram 4
  - equals sign (=), condition rule syntax 91
  - events *See* business events



execution details, business process completion and 4  
 execution management 16  
 external data model 58  
 External link, Metadata tab 47

## F

---

fact value 75  
 fields, available, guided decision table wizard 99  
 File Explorer, viewing repository details 27  
 Finish button  
   adding new projects 30  
   guided decision table wizard 98

## G

---

gateways, process diagram 4  
 getName method, name field 49  
 guided decision table wizard 97  
   adding fact patterns 99  
   adding imports 99  
   creating decision table step by step 98  
   defining actions 101  
   initial summary page 98  
 guided decision tables 14  
 guided rule editor 78–83  
 guided rule modeler 14  
 guided rules 14, 85  
   adding conditions to 80  
   comma separated list 82  
   creating with DSL 93  
   described 78  
   DSL (domain specific language) and 86  
   guided rule editor  
     applying constraint 81  
     field constraint 81  
   guided rule editor *See* guided rule editor  
   origin field 81–82  
   overview 64  
   pop-up for creating new 80  
   testing  
     steps to create guided rule test 83–84  
 guided score cards 14

## H

---

historical data, previous process instances and 3  
 human involvement  
   and example of STP process 5  
   and fully-automated processes 5

business processes and reducing human errors 9  
 defining a process and 4  
 Human Resources (HR) department, business logic example 60

## I

---

if-then construction 61  
 if-then-statements 8  
 imported external data model 55–57  
 Imports, step in creating decision table 98  
 inference engine 63  
 Insert a new fact, section, technical rule testing 71  
 install project, getting started with JBoss BPM Suite 20  
 installation process, JBoss BPM Suite 19  
 installation setup, JBoss BPM Suite 19  
 integrated development environment (IDE)  
   tooling 49  
 Intelligent Integrated Business Runtime 12  
 interoperability, JBoss BPM Suite and 12  
 Item successfully validated, message 50

## J

---

Java 44  
   business logic 61  
   data model implementations and 38  
   source code and Source editor 49  
 Java 7/Java 8, supported by JBoss BPM Suite project 20  
 Java archive (JAR) file 53  
 JBoss BPM  
   cloud experience 35  
   data modeling 37  
 JBoss BPM. *See* also BPM (Business Process Management)  
 JBoss BPM Suite  
   accessing container installation 21  
   artifact storage 56  
   as collection of components 12  
   BPM analysis tooling 15–16  
   Business Central 22  
   Business Central console  
     Artifact repository view 54  
   business complex rules  
     guided decision table 97–101  
   business logic 59  
   containerized installation 20, 36  
   data model editor 42

- data modeler 38
- described 11
- DRL (Drools Rule Language) 64
- example data model for implementation in 38
- external data model 53
- maven repository 56
- Maven, project build tool 29
- modeling tools 13–15
- packages and 34
- process testing 1
- project organization 23
- rules
  - inference engine 63–64
  - rule processing 64
- running locally vs. running in a container 21
- runtime engines 12–13
- simulation of process instances 3
- syntax for constructing rules 60
- JBoss BPM Suite architecture 1
- JBoss BPM Suite Easy Install project 20–22
- JBoss BPM Suite product, configuration options within 20
- JBoss Business Rules Management System (BRMS) product 60
- JBoss Developer Studio 14
- JBoss EAP (Enterprise Application Platform) 12
- JBoss Enterprise Application Platform. *See* JBoss EAP
- JBoss Travel Agency, example project 36
- jBPM projects 10
- jbpm, community project 11
- jbpm-designer, community project 11
- jbpmmigration, community project 11

## K

---

- Key Performance Indicators. *See* KPI
- knowledge workers. *See* also business value
- knowledge, business value and 3
- KPI (Key Performance Indicators) 16

## L

---

- language, human readable 87
- Latest Version, data modeler button 43
- layers, JBoss BPM Suite 12
- left-hand side *See* LHS
- Level, variable placeholder 92
- LHS (left-hand side) 63
- List, Repositories menu 26–27
- Literal value button, technical rule testing 73
- lock icon 48
- Lock status field, Metadata tab 48

## M

---

- Manage Organizational Units menu item 24
- Managed Repository check-box 26
- max, variable placeholder 92
- Metadata tab 47–48
- min variable 92
- modeling tools 13–15
- MVEL language 89

## N

---

- New Field entry form 44
- New Field pop-up 43
- New Item menu 79
  - adding new projects 29
  - rule implementation choices 87
- New Project pop-up, adding new projects 29
- New repository, Repositories menu 26
- Next button, guided decision table wizard 98
- no-loop attribute 68, 94
  - decision table 103
- Note field, Metadata tab 47

## O

---

- Object Management Group. *See* OMG
- OCP (OpenShift Container Platform) 35
- OMG (Object Management Group) 4
- Open button, Artifact repository 54
- Open Project Editor button 33, 55
- Open Source
  - projects, community of 1
  - software solutions 10
- Open Source Integrated Development Environment (IDE) 14
- OpenShift Cloud
  - basic setup 35
  - example projects 35
- OpenShift Container Platform. *See* OCP
- OpenShift Online cloud 20
- OpenShift xPaaS 35
- organization
  - and searching ways to constantly grow 2
  - automation in modern 9
  - business value 3
  - data structure 38
  - implementing business logic and 59
- organization structure, example of *See* project, starting first
- Organizational Unit Manager 24
- organizational unit, adding 23–26

Organizational Units menu 24  
 Overview tab 46–47  
 Overview tab, saving modifications 49

## P

---

package 34  
   default level 67  
   described 36  
   references to 34  
 pattern matcher 63  
 patterns, available, guided decision table wizard 99  
 Persistable check-box 41  
 pop-up, creating new data object 40  
 premise, defined 61  
 Previous button, guided decision table wizard 98  
 process instance  
   simulation 3  
 process development  
   JBoss BPM Suite 1  
 process development projects, aspects that support 1  
 process diagram  
   described 4  
   example of 5  
 process engine 13  
 process instance 16  
   rehydration 6  
 process manager 16  
 process modeler 14  
 process monitoring, business value and 3  
 process simulation, BPM analysis tooling 15  
 product documentation  
   available configuration options and 20  
   JBoss BPM Suite 19  
 production memory 63  
 project  
   adding new 28–33  
   business logic 59  
   data model implementation 38  
   external data model 53  
   multiple example cloud projects 36  
   starting first 22  
     adding organizational unit 23  
     adding repository 26  
     administration perspective 22  
     organization structure 23  
     Project editor 30  
   viewing default level of 41  
 Project Authoring perspective 28  
   adding dependencies to project 55

  project authoring perspective  
     inability to see where to add data model 54  
 Project Explorer, adding new projects 28, 30  
 Project Explorer, expanded folder structure 41  
 Project Settings menu 55  
 Project view 31  
 projects 1

## R

---

readme document, getting started with JBoss BPM Suite Easy Install project 20  
 Red Hat  
   EAP and 12  
   JBoss Developer Studio 14  
   Red Hat Customer Portal 35  
   Red Hat Developers site 35  
   Red Hat OpenShift Container Platform 20  
 Refresh button 33  
 rehydration, process instance and 6  
 Rename, data modeler button 43  
 Reporting panel 75  
 Repositories menu 26  
 repository  
   adding  
     Managed Repository check-box 26  
     Repositories menu 26  
     Repository Editor 27  
   and project assets 23  
   cloning 26  
   described 36  
 Repository Editor 27  
 Repository view 31  
 research and development, communities and 1  
 resource waste, reduction 4  
 RHS (right-hand side) 63  
 right-hand side *See* RHS  
 rule design and DSLs 89  
 rule engine 85  
   DSL definitions 89  
 rule engine *See also* inference engine  
 rule syntax, developers and 89  
 ruleflow-group attribute 68, 94  
   decision table 103  
 rules  
   agenda 64  
   basic structure 62  
   conclusion 61  
   design 78  
   designing with DSL 93–95  
   example of credit card transactions 8  
   externalizing into JBoss BPM Suite 60  
   fired 63

- grouping 68
- guided rule editor 78
- implementation of business logic 59
- implemented as several technical rules 65
- inference engine 63–64
- pattern matcher 63
- premise 61
- production memory 63
- the way they operate 63
- validation 67
- working memory 63
- rules *See* also business rules, technical rules,
  - guided rules
- Run scenario button, technical rule testing 75–76
- runtime engines 12–13

## S

---

- Save button, data model editor 46
- Save this item pop-up 46
- Save, data modeler button 43
- score cards 14
- Select button, adding dependency 57
- setName method, name field 49
- Show as Folders, customize view menu 31–33
- Show as Links menu item 31
- Show Rules Fired button 95
- simulation report 15
- source code editor 58
- Source editor 49
  - constant validation 50
  - described 49
- Source field, Metadata tab 47
- ssh button 27
- start node, process diagram 4
- state engine, wait states and 6
- static applicaton code, business logic in 7
- static diagram 5
- STP (Straight Through Processing) 5, 17
  - example of 5
- Straight Through Processing. *See* STP
- Subject field, Metadata tab 47
- suite
  - various components 10, 18
- Summary, step in creating decision table 98
- technical rule editor 65–68, 85
- technical rules 14, 85
  - completed test, example 78
  - extending 77–78
  - for developers
    - creating a rule 67–68
    - development environment 65
    - DRL syntax exposed directly to user 65
    - example project 65–68
    - field restrictions 65
    - properties panel 65
    - technical rule editor 65
    - technical rule editor. *See* also technical rule editor
  - inserting new fact 78
  - overview 64
  - testing
    - activation rule flow group 71
    - Add Import pop-up 70
    - adding a field 73
    - CALL METHODS 74
    - Data Object tab 69
    - EXPECT section 74
    - fact name 71–72
    - GIVEN section 71
    - positive test case 71
    - test scenario 68
    - test scenario guided editor 69
- temporal element, business events and 8
- terminology, basic rule structure and 63
- test scenario 60, 85
  - and designing a rule using DSL 95
  - correctness validation 86
  - errors in 76
  - extending 77–78
  - finalizing 78
- Test Scenario menu item 68
- Test Validate Document, test scenario 83
- testing
  - artifacts 1
  - guided rules, steps for 83–84
  - process reaction to severe loads 15
  - rule flow group to be tested 71
- testing *See* also technical rules
- THEN section 94
  - and DSL (domain specific language) 91
- transition arrow, process diagram 4
- Type field, Metadata tab 47

## T

---

- task manager 16
- task node, process diagram 4
- task, process diagram 4
- technical previews 12

## U

---

- Upload button, Artifact repository 54
- upstream community projects 18

URI field, Metadata tab 47  
Use real date and time, menu item 74  
user form modeler 15  
user task engine 13

## V

---

Validate button 43, 49, 68  
    editing data object's Source 50  
validation 71  
    correctness 86  
    rules 68  
Version history tab 47, 58  
view, customizing 31–33

## W

---

wait states 4  
    described 6  
    rewards process 7  
web tooling, generating data objects and 58  
WHEN section, DSL (domain specific  
    language) 91  
when-then construction 61, 80  
Work Item Definitions 31  
Work Item Handlers package 34  
working memory 63  
workshop, business process and its execution 4  
workshops, online 19

