Compliments of

RED HAT®
DEVELOPER
PROGRAM

# Effective Business Process Management with JBoss BPM

by Eric D. Schabell

# *Effective Business Process Management with JBoss BPM*

Eric D. Schabell

For online information and ordering of these and other Manning books, please visit 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



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*

# 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.

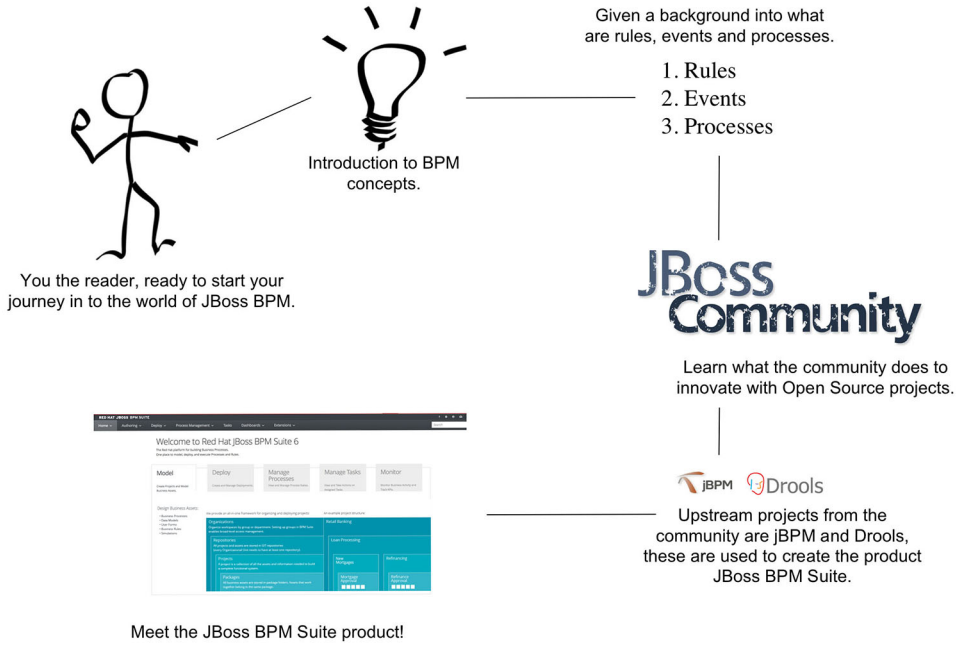


**Figure 1.1 The path you take through this chapter, which introduces you to BPM concepts, the path from community project to products, and getting started with JBoss BPM.**

## 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[1].

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-

[1] 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.

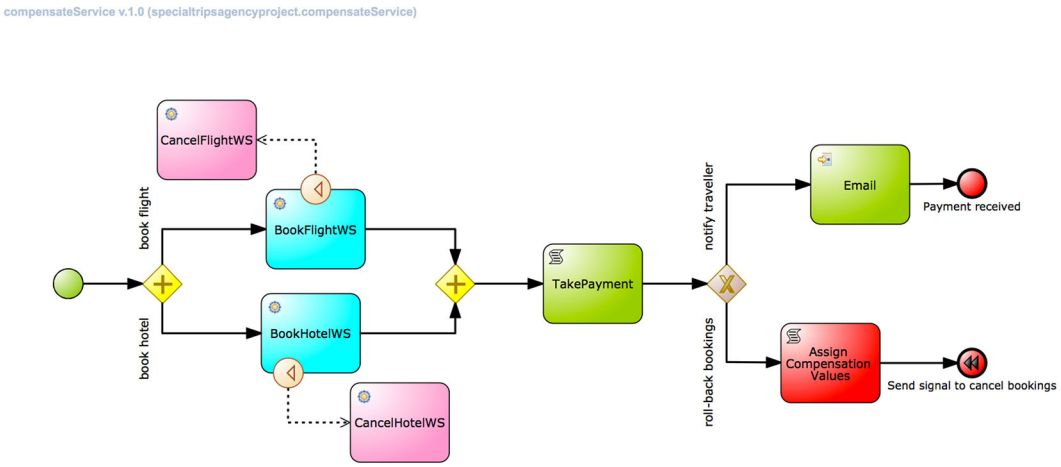


**Figure 1.2** **A process that captures a piece of business value; registering a booking for a hotel and a flight by taking payment from the provided credit card before notifying the buyer or determining that the payment was fraudulent and triggering a roll back of the bookings.**

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

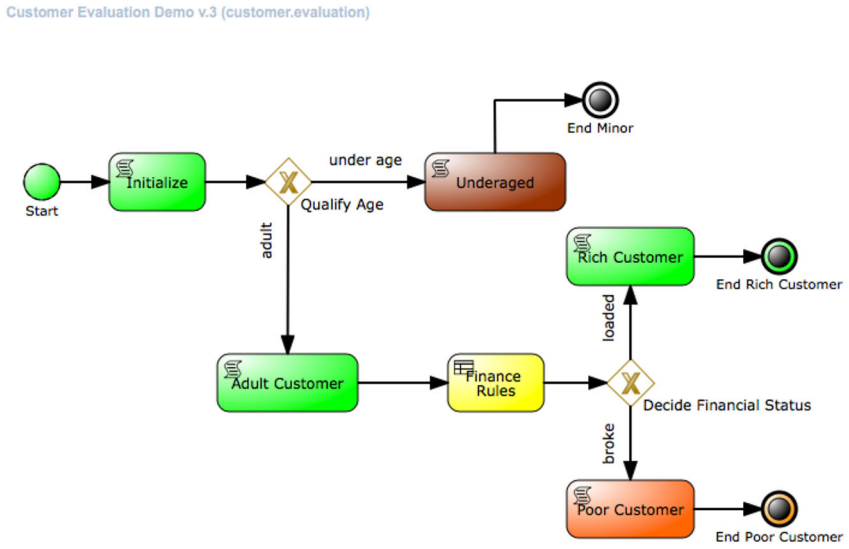


**Figure 1.3** **An example of a Straight Through Process (STP), this Customer Evaluation process has only tasks and decisions from start to finish.**

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.

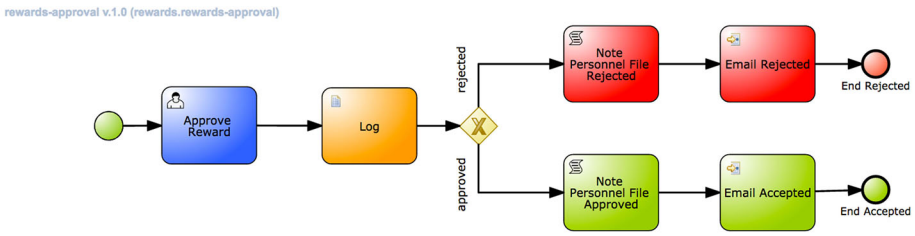


**Figure 1.4 The Rewards process has a wait state in the form of a user task which is used by a manager to approve or deny a submitted employee reward. The managers decision determines the completion path to be taken, approved or rejected.**

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[2]

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.

[2] 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, andyou'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.

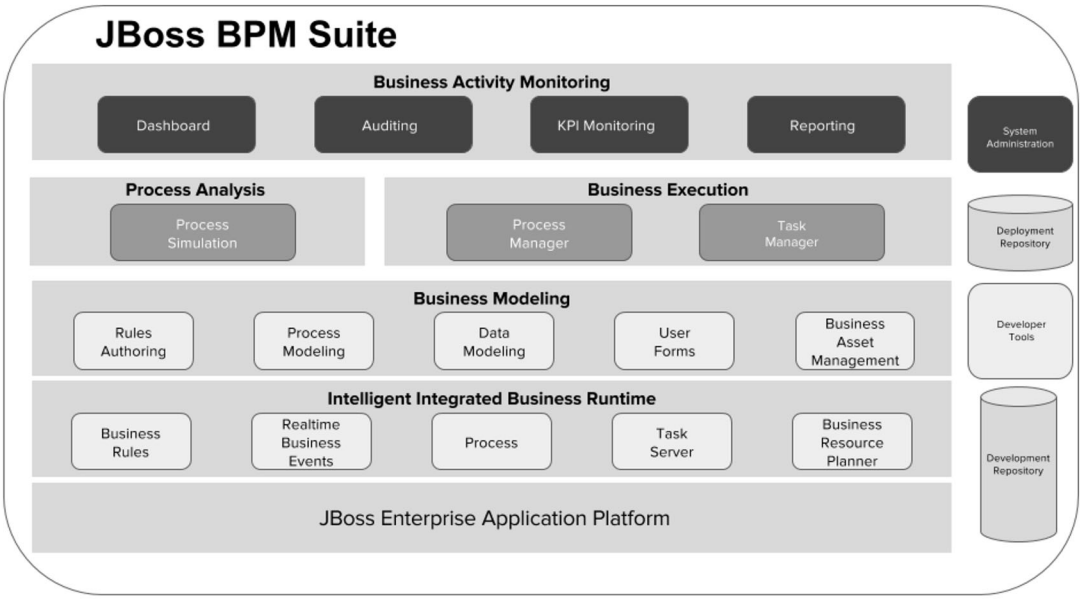


**Figure 1.5 JBoss BPM Suite component architecture shows the layers that make up a BPM suite. From engines, modeling tools, execution monitoring, analysis tools and reporting, the JBoss BPM Suite has everything you need to design, develop, run and manage your business processes.**

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 persons' 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.

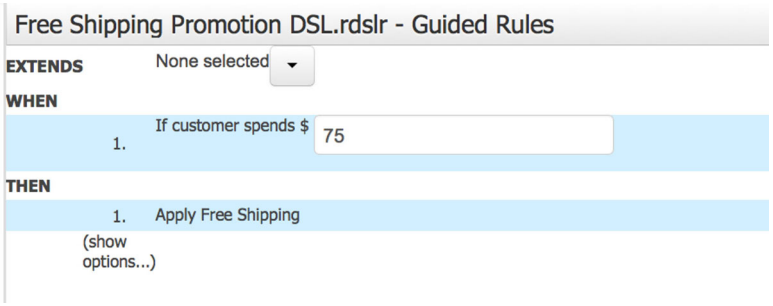


**Figure 1.6 An example of the Guided Rule Modeler where a developer's guided through designing a rule to apply free shipping (shipping value set to zero dollars) if the total value of the customer purchases exceeds seventy-five dollars.**

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

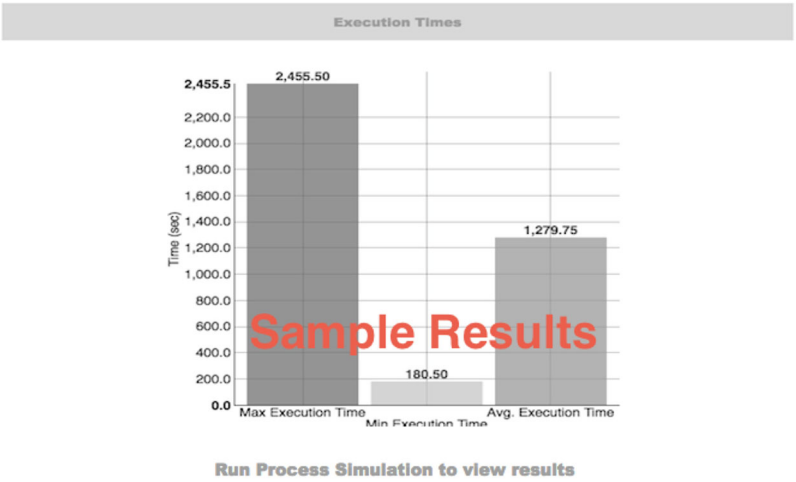


**Figure 1.7 The simulation output report can be a bar chart, like shown here, or changed into one of several other types to give a visual representation of the simulation data.**

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.

# 2 Tour a real project

**This chapter covers**

- Introducing the JBoss BPM Travel Agency
- Booking a trip
- Reviewing a trip booking

When starting off with a new technology it's nice to have a working example project to explore, preferably one that touches upon most of the major features and components that help us understand how to get started. This gives you something to lean on as you take your first steps into the world of JBoss BPM. As with most new technologies you want to investigate, there's nothing like working with a live functioning example, as not everyone's a fan of reading the provided technical documentation as a starting point, right? This chapter introduces the JBoss BPM Travel Agency project for you to explore as I walk you through how it works and how to execute a process instance.

After reviewing the project, you're going to submit your own booking for a fictional trip in Europe which showcases the elements of the process as covered in the first part of this chapter. To close out this chapter you review your executed travel booking process and examine the process instances, task data, and historical information through the provided process dashboards.

## 2.1 Introducing the JBoss BPM Travel Agency

Before you can run, you must learn to walk, as the saying goes. With this in mind I want to take a look at a completed working project, breaking down the various parts, explaining how the business process is progressing, and completing the tasks

necessary to put together a travel booking for the customer. Although there are many moving parts, I'm not explaining how to write rules, how to design the data model, how to design the process nor any of the other artifacts that make up the project. That sort of depth into specific BPM topics takes place in later chapters, ones that devote their entire focus on a specific topic. Here I'm explaining how these artifacts work, either individually or as a whole related to the travel booking process, providing the overall feeling of how a travel booking process is started and how it works its way through towards one of the possible endings.

The example project's called the JBoss BPM Travel Agency and, as you might guess, it's a travel agency booking process. It provides us with a chance to tour working rules, user tasks, services, processes, forms, dashboards and experience the integrated runtime tooling when executing an instance of the process. This project's completely self-contained and can be found online (https://github.com/effectivebpmwithjbossbpm/chapter-2-travel-agency-demo). You're installing, exploring and executing this BPM project to get a more complete impression of what a JBoss BPM project can do. This project works with Java 7 or Java 8 versions and it's assumed you've one of these installed on your machine. Checks are made in the demo installations for Maven to be installed and if you're missing it, it points you to the download location for you to install it.

If you're looking for an example installation project for JBoss BPM Suite that targets getting you setup without an example project installed, then see chapter 3. It's a simple installation project that gets you from nothing to up and running with a fresh install.

### 2.1.1 Install project

To get started, follow the project's provided readme document which contains all of the installation instructions. Two options exist for the installation, one for installing on Unix-based systems, and a second option for installing on Windows-based systems. The installation steps are simple and short, provided by the project's main readme file:

1. Download and unzip (https://github.com/effectivebpmwithjbossbpm/chapter-2-travel-agency-demo/archive/master.zip)
2. Add products to installs directory
3. Run init.sh or init.bat files. For init.bat, you need Administrative privileges on Windows.
4. Login to JBoss BPM Suite on http://localhost:8080/business-central (user: erics / password: bpmsuite1!)

Step one includes a link; click on it to start downloading the project. You end up with a compressed ZIP file on your local machine that you can unzip to ready the travel agency project for use. Step two's where you're downloading the products as listed in the README file found in the directory installs. These products are available on the

Red Hat Customer Portal[1] or on the Red Hat Developers website.[2] You need the following products for this project:

- JBoss BPM Suite 6.4.0.GA deployable for JBoss EAP 7
- JBoss EAP 7.0.0 installer

Once downloaded they're placed into the installs directory before you run the installation or warnings appear that they're missing.

Finally, you need to start the installation by using the init script that works for your operating system. For Unix based systems, run this in a console of your choice:

```
$ ./init.sh
```

For Windows based systems, run this in an Administrator console:

```
> init.bat
```

This starts the installation and you should be able to follow the output on the console screen as shown in figure 2.1. It ensures you only watch as it installs first JBoss Enterprise Application Platform (EAP), then JBoss BPM Suite, adds adjusted configuration files and finally the actual Travel Agency project's artifacts.

A second option to generate a containerized installation's found in the readme included with this project. It isn't possible to provide a container image as the products JBoss BPM Suite and JBoss EAP aren't delivered at this time in container form. You need to download the products as you did for a local installation, placing them in the installs directory, and then build your containerized project using Docker tools[3]. It's assumed you're familiar with the use of Docker tools, building containerized applications and can follow the instructions provided in the readme to create this containerized project.

### 2.1.2 Configuration details in JBoss BPM Travel Agency project

Before digging into the project deeper, let's walk through the files used to configure and set up the JBoss BPM Travel Agency when you run the install script. The adjusted configuration files are used to provide a pre-configured user and set some sane defaults with the optional system properties. You're invited to explore these files and you can see which ones are being installed where on the BPM server by examining the init file. It's written in the Bash language and is easy to follow along; the installation's

---

[1] The Red Hat Customer Portal hosts all the products but requires a customer subscription to the JBoss BPM Suite. It gives you access to more versions of the products which are still currently supported by Red Hat and can be found at https://access.redhat.com.

[2] The Red Hat Developers website gives access to certain versions of Red Hat products for developers to explore and test in their projects, but don't allow for supported production deployments without a valid subscription from Red Hat. You can find the downloads you need for this project there, JBoss EAP (http://developers.redhat.com/products/eap/download) and JBoss BPM Suite (http://developers.redhat.com/products/bpm-suite/download).

[3] 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/docker-toolbox.

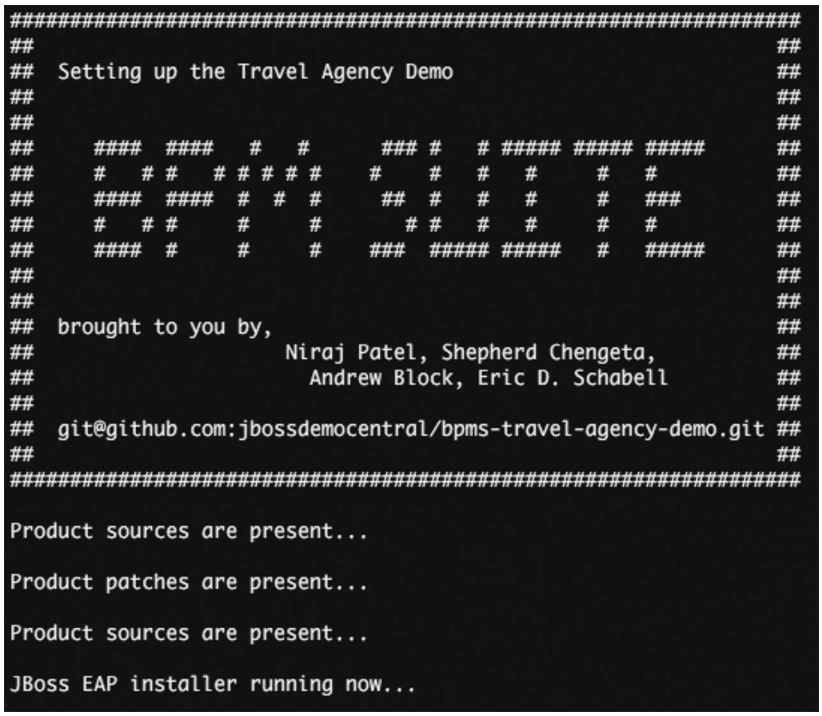


**Figure 2.1** **The installation of the JBoss BPM Travel Agency project.**

setup in a new directory called target/jboss-eap-7.0. All supporting configuration files which are referenced in the init script are found in the directory called support. Let's take a look at a few of them and what they do exactly.

Two installation files are required:

- installation-eap
- installation-eap.variables

These files are used by the installer for JBoss EAP for an automated installation. The file ending in variables is used by the installer to setup the user passwords.

The file unzip.vbs is a helper file for when you install this demo on windows and is of no further interest to us here.

To ensure that process email tasks work correctly and that notifications can be sent for user tasks, a bit of configuration's setup by placing a modified userinfo.properties file in the installed server. The format of the entries in this file are:

```
[username | groupname]=[email]:[country-locale]:[userlist]
```

An entry starting with a username needs to be valid in the system where JBoss BPM Suite's installed and in this case, it ensures that these names are created for this example project. It's also possible to setup group entries, and these lines would start with

the group name. The email address can be any valid email address for this user. The country locale's used to keep the character encoding in the language the user can understand. The entry's completed with at least one user but can be a list of users separated by a comma and placed between square brackets.

For example, the following's a line from the example projects file that shows the setup for a user erics and a line that puts erics into the manager group, which causes email notifications sent to the manager group would be sent to erics. If you wanted to add another valid user to the manager group, you would add a comma separated list after the current erics user:

```
erics=erics@domain.com:en-UK:erics
manager=manager@domain.com:en-UK:manager:[erics]
```

The user erics is created using the add-user tool found in the JBoss EAP installation and the roles needed for this project are assigned. The user has the following roles assigned and is located in the file application-roles.properties file in the JBoss EAP server configuration directory:

```
erics=analyst,admin,user,manager,taskuser,reviewerrole,employeebookingrole,ki
     e-server,rest-all
```

These roles are either default JBoss BPM Suite roles with pre-defined access setup within the online web console called Business Central, or they're project roles defined for user task access. Let's take a closer look at each role below in table 2.1.

**Table 2.1 This is an overview of the default roles that determine what a user sees and can access when logged into Business Central. Seven roles are shown with the corresponding descriptions of their level of access to Business Central work areas.**

| Role | Description of role |
|---|---|
| admin | Access to all of Business Central web console |
| developer | Access to everything except the administration area |
| analyst | Access to everything except administration area, asset repository and deployment console |
| user | Only access to process management, tasks and dashboards while hiding project authoring |
| manager | Only access to the reporting dashboards |
| kie-server | Provides ability to start the kie-server which supports rule execution |
| rest-all | Provides access to execute RestAPI calls to JBoss BPM Suite where available |

Don't worry about what each of these areas in Business Central mean, soon they become clear when you start up the example project and being to take a look at what it has to offer.

**Table 2.2 This is an overview of the user task roles used in the JBoss BPM Travel Agency. Each user task's designed to assign itself to a group role. Here the roles are listed for the two user tasks found in the process and a group description's provided for the tasks that users need to do to review and complete travel bookings.**

| Project task roles | |
|---|---|
| **Task role name** | **Task group description** |
| reviewerrole | Group in example project that needed to claim, work on and complete user tasks related to reviewing higher value travel bookings |
| employeebookingrole | Group in example project that needed to claim, work on and complete user tasks related to finalizing a travel booking. |

The task role names listed in table 2.2 pass the review when digging into the user tasks that are part of the example project.

Finally, the last item in the support directory covered's the directory bpm-suite-demo-niogit. This directory contains a JBoss BPM Suite code repository that contains the example project JBoss BPM Travel Agency assets. It's copied into the JBoss BPM Suite and automatically picked up by the server when it starts the first time. It's copied into the location:

```
./target/jboss-eap-7.0/bin/.niogit
```

Once the installation script's finished, you see the output on your console like figure 2.2, where you're given the command to cut-and-paste for starting the JBoss BPM Suite server and the URI, user name and password for logging into Business Central to get started with this example project.

```
=========================================================================
=                                                                       =
=  You can now start the JBoss BPM Suite with:                          =
=                                                                       =
=   ./target/jboss-eap-7.0/bin/standalone.sh                            =
=                                                                       =
=  Log in into business central at:                                     =
=                                                                       =
=    http://localhost:8080/business-central  (u:erics / p:bpmsuite1!)   =
=                                                                       =
=  See README.md for general details to run the various demo cases.      =
=                                                                       =
=  JBoss BPM Suite 6.4 Chapter 2 Travel Agency Demo Setup Complete.     =
=                                                                       =
=========================================================================
```

**Figure 2.2 After the init script's finished.**

You can start the JBoss BPM Suite server by either copy and paste or entering the line in your console at the prompt:

```
$ ./target/jboss-eap-7.0/bin/standalone.sh
```

This starts the server and you see some log messages roll across the screen until it's finished. Once finished starting you can open a browser, I suggest using the Firefox (https://www.mozilla.org) browser, as this is tested and certified for using JBoss BPM Suite Business Central.

### 2.1.3 Business central web console login and overview

After starting the server and opening the browser you can login to JBoss BPM Suite Business Central (http://localhost:8080/business-central) with the given user and password from the installation output. You find yourself in the home screen as shown in figure 2.3, where you've the chance to browse through some of the documentation by clicking on the tabs.

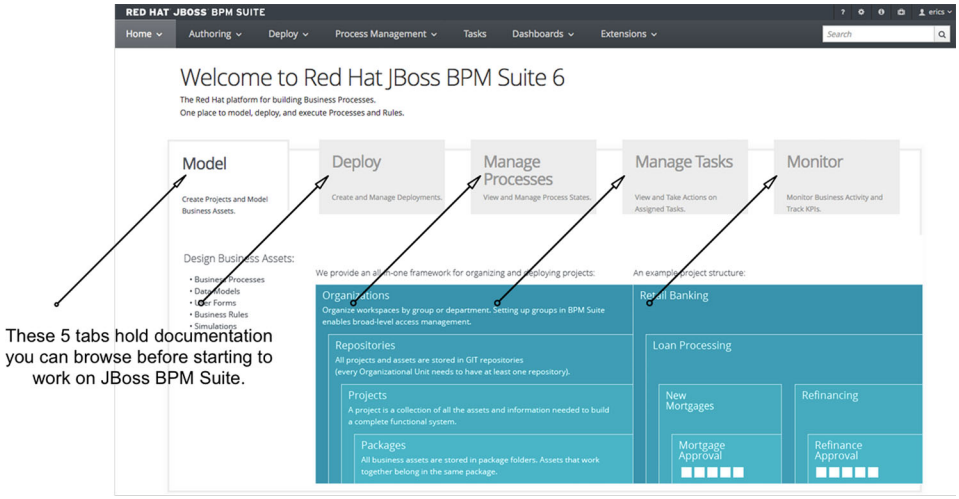


**Figure 2.3 The JBoss BPM Suite Business Central web console home login screen, as seen after initial login.**

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. I limit exploration of the example project at this time to the areas of Business Central that pertain to examining project artifacts, running the project, completing tasks and viewing dashboards.

### 2.1.4 Touring the JBoss BPM Travel Agency project

In this section, you start a detailed tour of the JBoss BPM Travel Agency project by looking at each artifact that impacts completion of the travel booking process. The idea's to explore how you get from the start, to the end and what's happening each step of the way. You're left with an understanding of how this booking process works, what artifacts are used and why. I'm not diving into low level details on how to design any of the artifacts that are reviewed. I'm also not teaching you how to use any of the

editing or designing tools, only showing you enough to investigate how that particular artifact in that specific tool works.

You're exposed to a finished project and every artifact's working to produce a successful travel booking process once deployed. In later chapters I dig deeper into the individual editors and designers. For example, you're shown a guided rule in this section but must wait for a later chapter to learn how to design your own guided rules.

To start the tour of this example project, open it in the Project Authoring perspective, which you can find under the Authoring menu as shown in figure 2.4. This opens your project authoring perspective where you see the project explorer on the left side of the screen along with a listing of the project artifact groupings, the messages panel on the bottom of the screen and the main process called specialtripsagencyprocess opened in the process designer tool on the right.

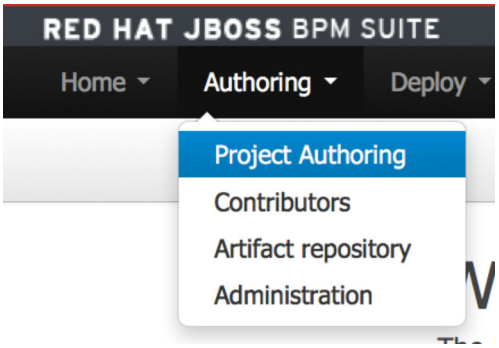


**Figure 2.4** **Open the project authoring view through the Authoring menu.**

### TAKING A THOUSAND FOOT VIEW OF THE PROJECT

At this point, before diving into how it works, it's helpful to look at this process project from the high level main process to gain some understanding of what it's doing without diving into the technical details yet. You now have the main travel booking process on your screen, looking something like figure 2.5.

The booking process starts after a user submits data; who they are, where they want to go and when. This is taken by the booking process and enriched by making two service calls to collect flights and hotels that match the submitted request. This data's put into the projects data model to be validated. If it fails validation, the process ends here in an error state; it's important to be able to collect failed processes for dashboard reports later. If data validation passes, the process proceeds to call a sub-process to calculate the pricing for the requested booking based on flights and hotels that were found. When the sub-process completes and returns, the process continues with a check if the total booking price's over a threshold set at 2500. If it's over this threshold, then there's a review by a manager to decide if any changes need to be made to the bookings total pricing. After this review, the booking's reviewed by an employee to either cancel the booking, send it back around for another price review, or add payment details to finalize the booking. When you add payment details it proceeds to a sub-process which is used

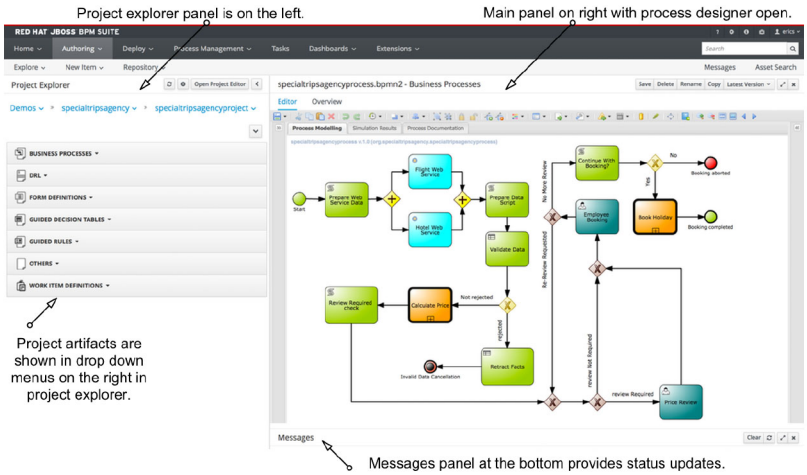


**Figure 2.5 This is the initial project authoring view with the Project Explorer with the specialtripsagencyprocess open in the process designer.**

to book the flights and hotels before checking for a fraudulent payment method. It's done in this order to showcase the possible rolling-back of the flight and hotel bookings after they've been booked, which is known in BPM circles as compensation. If the payment method's fraudulent, then the booking's rolled back and the process ends in an error state. If not, then the payment's charged and an email's sent to the customer as confirmation before the process ends in a successful state.

Now let's examine the process in detail. Your screen should look like figure 2.5, but first a few suggestions when working on BPM projects. You need to realize that a large viewing area provided by a good-sized monitor's essential for BPM process projects. If you need to, find the plus-magnifying-glass and minus-magnifying-glass buttons in the row of icons at the top of the process designer to resize your view of the process to fit your screen resolution.

Because the process designer's open on the main specialtripsagencyprocess, let's start by exploring the global layout of the designer. Available processes in a project can be found by first opening the group Business Processes found on the left in the Project Explorer by clicking on it. In this project there are three entries; calculatepriceprocess, compensateService and specialtripsagencyprocess. By clicking on any one of these entries you open that process diagram on the right in the process designer for viewing and editing. I'm going to stay with the initial specialtripsagencyprocess and start by opening the Object library on the right by clicking on the double arrow button on the left of the process designer. This causes a pane to slide out on the left which is filled with process node types that can be used to design your process, see figure 2.6.

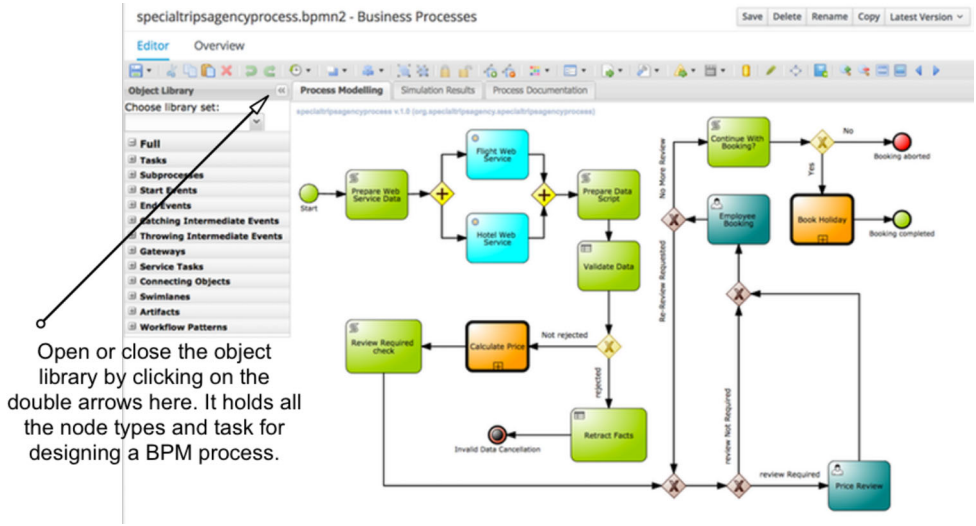


Figure 2.6 **The process designer with the Object library open in the left pane.**

At the top of the object library pane you see a pull-down menu that lets you select from one of three options for the object library set of node types you want to see:

- Full – default and contains all available out of the box node types
- Simple – contains a sub-set of the full set of node types
- RuleFlow – contains only the node types supported by rules in a workflow which is supported by JBoss Business Rules Management System (BRMS)

As noted in chapter 1, these are all standard node types and tasks as defined by the OMG specification called BPMN. Currently JBoss BPM Suite's based on the BPMN 2.0 specification and aligns it's work with the executable node types which is a specific section of the specification. See the OMG site for more details at http://www.omg.org/spec/BPMN/2.0.

In this book I refer to the default full set of node types, unless stated otherwise. The process node types are grouped into the categories you see and each category can be explored by clicking to open them. The available node types are covered in the chapter on the process designer, but as an example, the Service Tasks menu expands to show the following service tasks that are available for drag-and-drop into your process design:

- Email – a task to send an email
- Log – a task that logs a given message in the JBoss BPM Suite server logs.
- Rest – a task to call a REST-based web service
- WS – a task to call a standard web service

Note that these service tasks are all for integrating with backend systems. After this you can close the left pane of the object library by clicking on the double arrows.

The next pane to explore's the right pane, which opens by clicking on the double arrow button found there. This slides out the Properties panel, which is filled with information about any element in your process, but with no element in the process currently selected it shows the high-level process properties. Try selecting elements in the process, like a node or a gateway, and see what the properties are. For example, in figure 2.7 I've selected the Flight Web Service and you can see that the properties are filled with information needed to connect to an external web service in a service task.

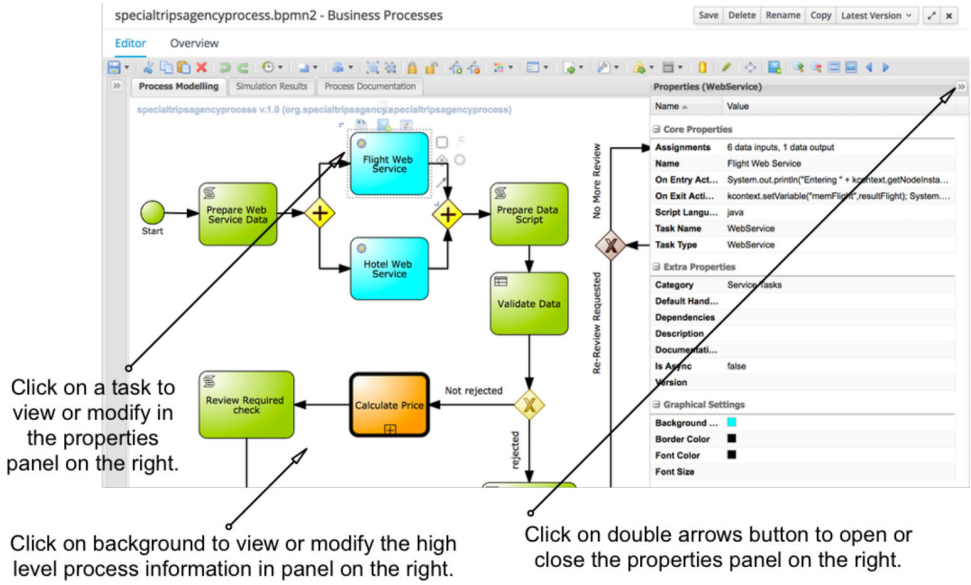


**Figure 2.7** **Viewing process information or specific task properties accessed in pane on the right.**

### STARTING THE TRAVEL BOOKING PROCESS

At this point it's time to start examining what this process does. To do this I take you through each process step while examining the rules, services, task form or sub-process which are being used to complete the step.

As luck would have it, the specialtripsagencyprocess opens in the process designer and is the starting point when any travel agency booking's started. The first thing of interest is what data's submitted when this process starts. JBoss BPM Suite generates a task form to submit the necessary data to start an instance of a process. This can be found under the drop-down menu Task Forms listed on the right in the Project Explorer panel. Click on this to view the available task forms, opening the form named org.specialtripsagency.specialtripsagencyprocess-taskform as shown in figure 2.8.

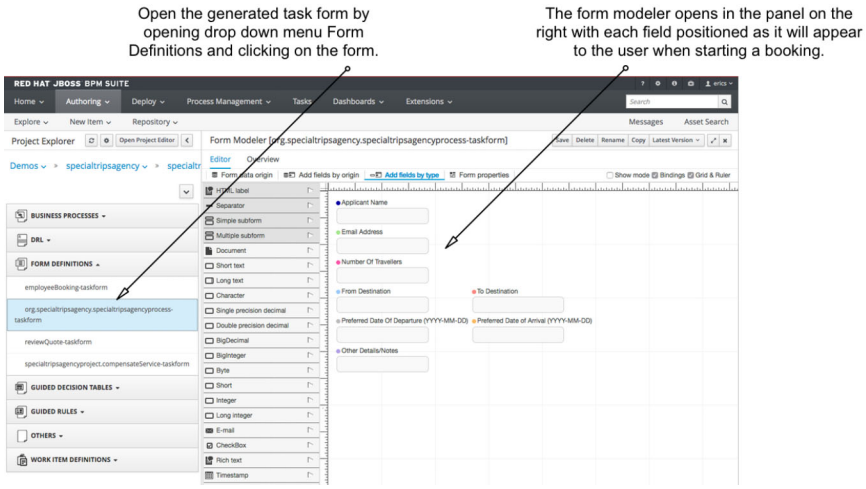


**Figure 2.8 The form modeler used to create the initial form users see when starting a booking.**

This task form lists the data to be collected from a user who wants to start a bookings process for their trip. It asks for traveler details, departure and arrival destinations, dates and any comments. This form appears every time a new process starts from within the JBoss BPM Suite and is provided for convenience. In most cases this sort of information would be collected from a custom web application and an example's provided, as shown in figure 2.9. It's deployed during the installation and ready for you to use in the web browser (http://localhost:8080/external-client-ui-form-1.0), but only works after you've built and deployed the travel agency process. If you try to use the form without a project deployed, it will fail to start a process as it's unavailable.

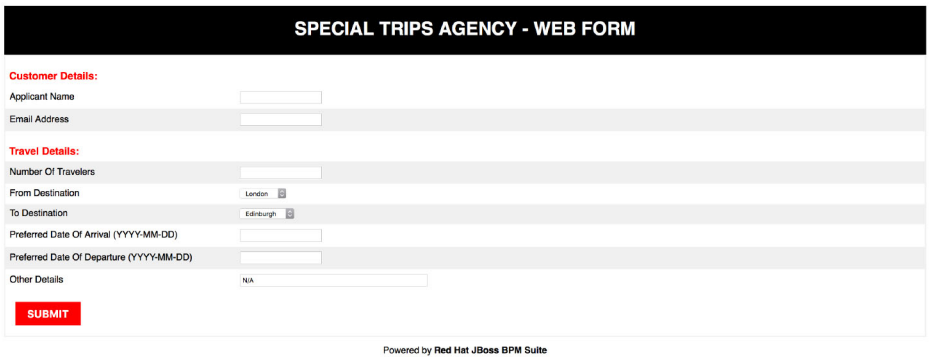


**Figure 2.9 This example web application collects and submits booking data through the JBoss BPM Suite RestAPI to start a booking process.**

The source code for this web application can be found in projects/external-client-ui-form-1.0 if you're curious as to investigating how it works.

### DETAILED WALK THROUGH THE MAIN BOOKING PROCESS

Now that you understand what data's being submitted, let's go back to the main process specialtripsagencyprocess and take a look at what's done with this data to reach the final goal of a travel booking that the customer's willing to pay for. Make sure the main process is open and make use of the properties panel on the right by selecting each task in the process described to follow along.

The first task after starting the process is named Prepare Web Service Data. It's important that everything done in a process is modelled visibly, nothing hidden down in the implementation code base which is backing the process tasks. The reason for this is that you want to be able to both understand and maintain this process moving forward without having to dig through every layer of the project to find out what's happening. The business process you're viewing's the communication vehicle for everyone involved in this project; make sure that everything being done's visible.

The task here's a script task which is used to allow for anything technical to be implemented here. This type of task uses either Java, JavaScript or MEVL languages to compose what needs to be done. In this task, it's being done in the Java language and you can examine what's been composed by clicking on the property Script in the Properties panel on the right after selecting the task node in the diagram. It pops up the Expression Editor as shown in figure 2.10 displaying the code used to map incoming data from the initial task form to the available data model or variables in the process.

Once this data's been mapped it becomes available as variables to be passed into and out of any of the following tasks in a process. Think of these as global process variables. For this example, it isn't important to understand the specific mapping details of the Java code found in this task, but to understand the visibility you're giving to this step happening in a process.

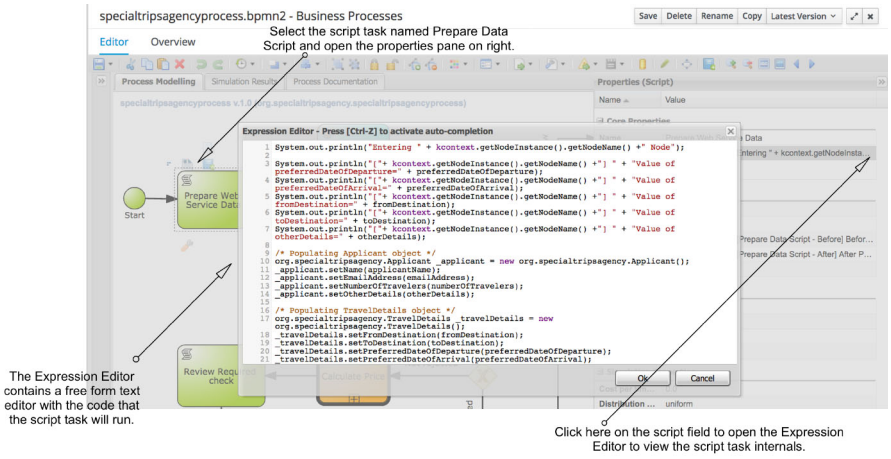


**Figure 2.10 Viewing the implementation details of a script task.**

The next node in the process is a parallel gateway. Gateways are used to provide divergence or convergence in the process flow. The three most common gateways are:

- **Parallel gateway** – when used to split a sequence flow, all outgoing branches are followed simultaneously. When used to merge branches, it waits for all incoming branches before continuing to the outgoing flow.
- **Inclusive gateway** – when used to split a sequence flow, one or more outgoing branches are followed based on branching conditions. When used to merge branches, it waits for all incoming branches before continuing to the outgoing flow.
- **Exclusive (XOR) gateway** – when used to split a sequence flow, it routes to only one outgoing flow based on branching conditions. When used to merge branches, it waits for one incoming branch before continuing to the outgoing flow.

As this is a parallel gateway, the process branches on to both of the paths provided to arrive in parallel at the service tasks named Flight Web Service and Hotel Web Service. The source code for these two web services can be found in the example projects here:

- projects/acme-demo-flight-service
- projects/acme-demo-hotel-service

Both of these projects are built automatically for you and installed in the JBoss BPM Suite server for use by these two service tasks. You can explore the properties of these service tasks for now as an exercise. The only thing I'm going to mention here's that these service tasks call the backing web services to retrieve flight and hotel data based on the given bookings request.

After leaving the service tasks and having gathered flight with hotel data, the process flow continues into a parallel gateway where it waits to merge the two flows before continuing on to the script task Prepare Data Script. As in the first script task examined, open the properties panel on the right for this task and explore the Java code used to place the collected web service flight with hotel data into the process variables before continuing onwards. After they're placed into process variables, the data needed for evaluating any rules later in the process are placed into working memory. The work done in this task concludes with some statements that insert parts of the data into this working memory. More on the details around rules, facts and working memory's covered in the chapter on business rules. For now, be aware that this task inserts data into working memory at this point, and later this data can be used to determine which rules are triggered.

Validate Data's a rule task that uses rules to determine if submitted form data's valid and, if not, the process ends here. If you examine this rule task by viewing the properties, you find an entry under Ruleflow Group which is naming the rules that are to be applied in this rule task. Here you see rejection's the ruleflow group name used, and if you browse the project rules you discover rules that belong to this group. I've collected the rules here which are part of the rejection group; they're nothing

more than simple data validation rules for the user submitted data and can be found in the Project Explorer panel on the left:

- DRL menu – ObjectValidation that validates the submitted email address.
- Guided Rules menu –
  - InvalidFromDestination – validates departure city's in given list.
  - InvalidToDestination – validates arrival city's in given list.
  - InvalidRatePerPersonForFlight – validates that rate per person's within given limits.

If any of the validation rules fail, they set a rejection flag by creating a Rejection object that signals a problem at the next gateway. From there a decision's made to continue processing or stop due to any data validation errors detected.

The following's a simple rule, found in the Guided Rules and called InValidFromDestination, which is used to validate the departure destination that was submitted. It's a guided rule that I walk you through which has a condition to be met in the first part, which, if met, triggers the second part of the rule. It reads as if written in standard English as it's a guided rule, meant to keep you up and out of the weeds during rule design.

Let's look at it here before going into details:

```
WHEN

  There is a TravelDetails with:
    fromDestination is not contained in the (comma separated list)
    "London", "Gatwick", "Farnborough"

THEN

    Insert Rejection [fact0]
    reason "Sorry!!! We do not provide services from this destination."
    System.out.println("Rejected due to Invalid From Destination");
```

The first section in the rule starts with WHEN, which is where all the conditions that need to be met to cause this rule to be activated are found. To apply the resulting actions, you look for the section that starts with THEN. The conditions of this rule are using process variables and data found in them. It states that within the TravelDetails object from the data model, if the fromDestination field isn't set to one of the three given cities, then this rules actions found in the second part of the rule are applied.

The second section of this rule starts with THEN, which is where the actions to be applied are found. These actions are to create one of the data objects called Rejection and to insert this into the process instance. This makes it available to the rest of the process tasks when moving onwards after this current task. This data object's labeled with a name, fact0, and if necessary you can identify this particular Rejection data object from others that might be created. Within this object there's an attribute called reason which can be used to show what the detected data validation error was by assigning an explanation. Finally, there's a technical line of Java used to insert a line

into the JBoss BPM Suite server log, which is only here for the demo function of this project. It's making everything visible for the user who's running this demo and might not be found in your real life projects beyond initial development.

### SUB-PROCESS TO CALCULATE TOTAL PRICE OF THE TRAVEL BOOKING

The next task in the process is a reusable sub-process, which is a task which calls a separate process, and when that process completes it returns to this point to continue onwards. If you open the properties for this task you see a field Called Element which is where it names, in the form projectname.processname, the process to be called. Here shown in figure 2.11 is the sub-process to be called specialtripsagencyproject.calculatepriceprocess.

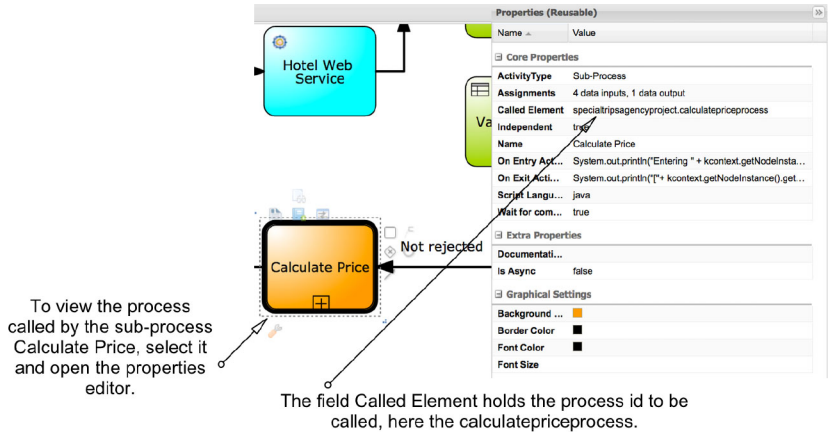


**Figure 2.11 Examining the sub-process task Calculate Price reusable to determine the process being called.**

You can find and open the calculatepriceprocess found in the Business Processes listed in the Project Explorer panel on the left. The process shown in figure 2.12 is your first ruleflow, a process type that contains only rule interactions and orchestrates the order in which these tasks are processed. Note that each task in this process is a rule task, specifically designed to take a given group of rules and evaluate them against data found in the process variables within this process instance.

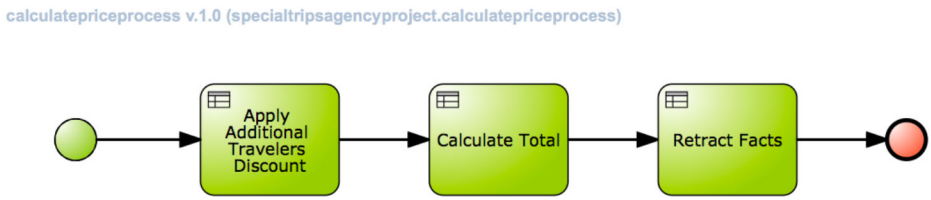


**Figure 2.12 This is a ruleflow, a process type that contains only rule tasks that interacts with only rules.**

The first rule task, Apply Additional Travelers Discount, has the ruleflow group discount, which is located under the Guided Decision Tables found in the Project Explorer in the left panel. A single guided decision table rule's called AdditionalTravelersDiscount that you can open by clicking on it. In figure 2.13, you see a spreadsheet type of rule with a single entry in the table with the ruleflow-group column denoting that this rule indeed is the discount rule attached to this rule task.

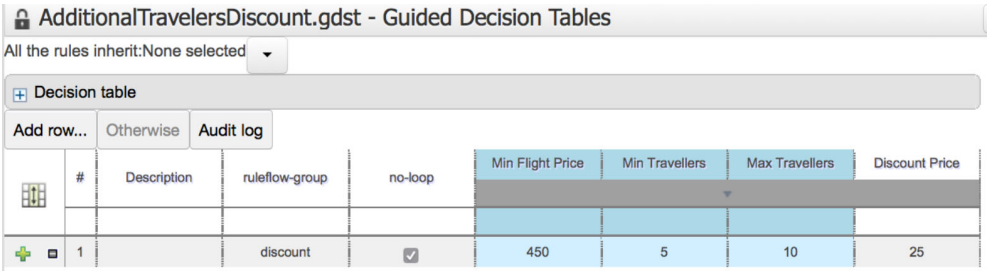


**Figure 2.13 This is a decision table used to determine if the booking rate a discount based on the minimum price of the flight.**

To get a clear idea of which data objects are looked at when evaluating this rule, you can click on the tab Data Objects at the top of the decision table editor and see that this decision table uses:

- Applicant
- Flight

By using the entire data object, all their attributes are available for evaluation. You can go back to the actual decision table editor by clicking on the tab Editor located at the bottom.

The rest of the columns give you some idea as to the evaluation that needs to be made in the one row in this table. The conditions that need to be met are that a flight must have a minimum price of 450 and a minimum of five travelers, but no more than ten. If these conditions are met by the data in working memory, then the rule triggers to set the flight discount price to the value of 25. The flight data object's set with the new value and the rule task, when it completes, ensures that this is put back into the process variable that holds the flight object. You can assume that mapping data that change in a rule task, like the discount prices attribute, happens automatically back to the main process variables. This ensures that they're available to any tasks that might need them later in the process.

The next rule task's called Calculate Total and it uses a technical rule which is part of the ruleflow group called total. Technical rules are known as DRLs, which is short for Drools Rule Language. This rule's located under the DRL menu entry in the Process Explorer in the left panel. Open by clicking on the rule found there with the name CalculateTotalPriceBasedOnDiscountProvided. This is a descriptive name and

it should be obvious what it does, but let's walk through it to see that it's in a form which is normally be created by a rule developer using their code development tools.

This rule's written in the Drools rule language known as MVEL. It goes too deep right now to discuss this language in depth here, but you might be able to see that it first checks that in this booking:

- There's at least one traveler
- That a flight price has been set for each traveler
- That the hotel price has been set

Once these conditions have been met, the rule writes the value of various data objects to the server log before totaling up the booking cost. It also reduces the total booking cost by the discount calculated in the previous rule task.

The final rule task's called Retract Facts, which signals that the process cleans up the working memory by removing the facts that were used in this ruleflow before returning to the main process flow. The reason for this is to ensure that a later rule task has a clean working memory, one that isn't populated with data objects from the calculations made in this ruleflow. Always try to remember to manage your working memory as you use a ruleflow, cleaning up the facts before you leave. This rule task calls the rules assigned to the ruleflow group called retract, which are found under the Guided Rules menu in the Project Explorer on the left side panel and listed here:

- RetractApplicant
- RetractBookingObject
- RetractFlight
- RetractHotel
- RetractRejection
- RetractTravelDetails

Pick any of these to view by clicking on the rule of your choice. Each one checks for the existence of the given object in working memory, if it's detected, it's then removed from working memory. This concludes the calculatepriceprocess, which takes you back to the main process flow specialtripsagencyprocess at the Calculate Price sub-process task.

#### CONTINUING TO REVIEW TASKS IN THE MAIN TRAVEL BOOKING PROCESS

The next task's another script task called Review Required check, which checks the total price of the booking request to see if it's greater than 2500. If this is the case, a process variable called reviewRequired is set to true, as it's a Boolean variable. This process variable's checked later to decide if a review of the booking's required before proceeding to collect payment details.

The next item in the workflow's a gateway which is for the purpose of joining two incoming process flows, the one currently on and one that brings the process back into the main process flow at this gateway.

After the two flows come together it proceeds to a gateway with two outgoing possibilities. This is a decision point where it might seem odd that gateways aren't the decision point, but the outgoing sequence flows are the point of decision. These are the outgoing arrows and if you select the sequence flow labeled review Required, you can view the properties panel on the right (slide it out if you need to). The Condition Expression Language field shows the language to be used's Java, and the Expression field contains the actual expression to be evaluated. It needs to be a Boolean result, and it must evaluate to true or to false. You see the return of the process variable reviewRequired here being checked for returning the value of true to take this path:

```
return  KieFunctions.isTrue(reviewRequired);
```

It makes use of a helper function that tests if the process variable's true. Now if you look at the other path, labeled review Not Required, you find that the expression's looking for a false evaluation:

```
return KieFunctions.isFalse(reviewRequired);
```

Remember the current settings in the main process flow causees the review to be triggered if there's a booking with a total price of more than 2500. If this review's required a user task's reached, called Price Review, which asks someone to review the pricing that belongs to the group called reviewerrole. Figure 2.14 shows the task form shown to the reviewer once they claim a task to work on. Only the last two fields are for the reviewer to modify; the rest are displaying the booking information for her to use to decide on to adjust the price of the total booking.

The following node in the process is a gateway used only to bring the two incoming flows together before proceeding to the user task called Employee Review. This'd be the point where you've submitted your travel booking request and the travel agency employee isn't calling you back to ask if you like the flight and hotel options along with the final price. If you do, they take your credit card payment information and submit this to continue on with processing. Another option's that you might find the flights, hotel or price aren't to your liking, and you decide to cancel the booking. The travel agency employee can then submit your request to cancel before continuing onwards with the process. Another option's that, after reviewing the flight, hotel and price, that there might be a reason to request that another review take place due to some error in pricing. The travel agency employee can then select this option and the booking's sent back to the previously discussed Price Review task for another appraisal by the reviewer.

All of this can be found in the Form Definitions menu in the Project Explorer panel on the left, under the form employeeBooking-taskform. The form displays the entire booking for review and then presents the three above described options for the travel agency employee to choose from.

Following the Employee Booking review task, it's possible that the entire booking was selected to be reviewed again to reach the next gateway node and evaluate to the

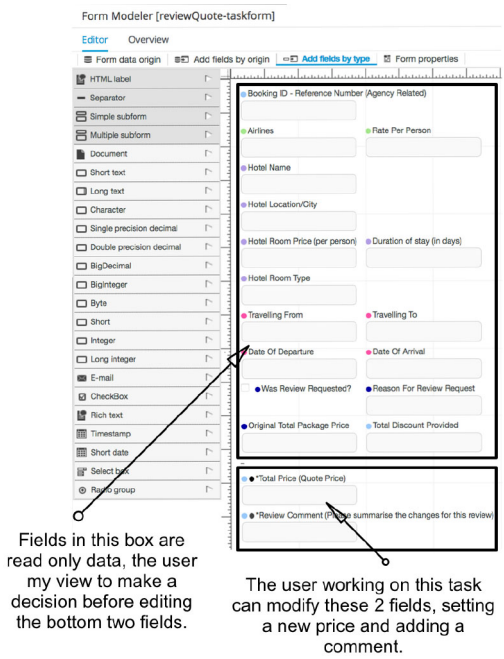


**Figure 2.14 The task form called reveiwQuote taskform's to be used when a review's required due to the total price of a booking exceeding 2500.**

sequence flow labeled Re-Review Required. Should that be the case, it's sent back to the Price Review user task before coming back to the Employee Booking user task.

The other result than can come out of the Employee Booking task's the booking's cancelled or payments made by giving credit card information. Both of these options would resolve in the following gateway to continue on the path labeled No More Review into the script task Continue With Booking.

This script task's used from some housekeeping around the data collected in the last user task. It determines if the booking's to be cancelled and sets the process variable bookingCancelled. It also extracts the payment, flight and hotel details, which are used to update their respective process variables. Finally, this script task writes some of the details into the server log for posterity.

The next gateway's used to determine if the booking's to be cancelled. If it is, then it ends in the Booking aborted node. It's a good practice to label all end states for a process flow to allow reports to filter out successful instances and provide lists of the ones that result in lost bookings.

### EXAMINING THE SUB-PROCESS TO FINALIZE TRAVEL BOOKINGS

If the booking isn't cancelled, then it enters another sub-process task called Book Holiday. This process is called compensateService and can be found in the Business Processes menu on the right in the Project Explorer panel. Figure 2.15 shows you the process responsible for making the booking by reserving the flight and hotel with

their respective services before taking payment if the credit card isn't fraudulent. If the credit card's fraudulent it causes the flight and hotel reservations to be rolled back, otherwise it notifies the customer of their final booking.

The first node in this process flow's a parallel gateway which splits the flow into two parallel paths. They both encounter a service task, the BookFlightWS and the BookHotelWS. These are special service tasks which can be found under that label in the Object Library when that panel's slid out on the left. These are specific tasks that are linked through their properties to call web services that a developer's deployed to interface between the BPM process and specific back end systems in an organization. In this project, the web services are provided and provide simulated interactions with simulated back-end systems.

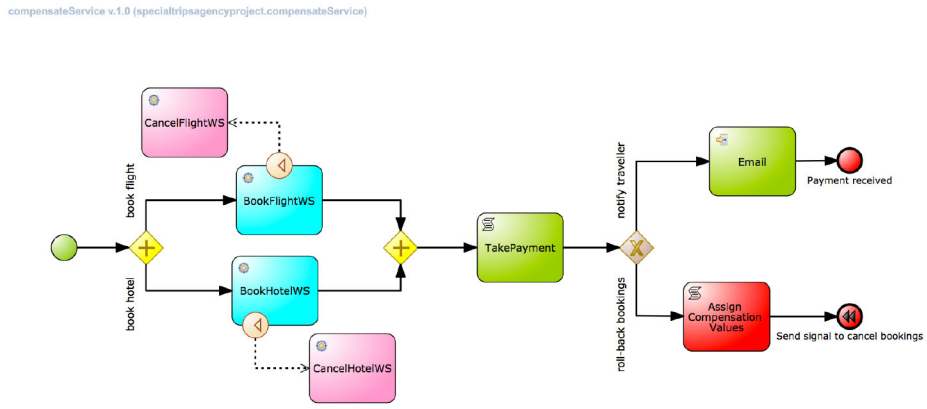


**Figure 2.15** **This process finalizes the customers booking.**

You discover attached Compensation events, which can be found in the Object Library under Catching Intermediate Events, and are used to catch a signal which is sent if a fraudulent credit card's detected. A signal's a way for a process to contact another point in the process; here it's a compensate signal which is sent if a fraudulent credit card's detected. The signal's caught in the compensation event node and follows the dotted sequence flow to the corresponding service task to cancel a flight or hotel reservation. You can only attach a single task to this compensation event and it's often a service call that you want to roll back.

After the flight and hotel have been reserved, the parallel flows are joined in a gateway before proceeding to a script task called TakePayment. This could've been a rule task, but it was chosen to be done with a script task to give you clarity in both how the process variables are setup for emailing the customer and how fraud detection works. It's a simple check for any credit card number starting with "1234…", noted by setting a process variable called moneyTaken to the value false. Within this script task you also find log messages that note various data values as they're processed.

The following gateway's used to check if the credit card was fraudulent, taking the path to the service task Email if it wasn't. In this case an email's filled with booking details before ending in the node labeled Payment received. This'd be the desired end state and the one the travel agency wants to reach with every process instance!

If the credit card's determined to be fraudulent, a script task labeled Assign Compensation Values is used to set up the necessary process variable that needs to be given to the compensation services. The next node's called Triggering compensation and is found in the Object Library under the Start Events menu. It triggers the compensation catch events discussed earlier which lead to the service tasks that call the compensation web services to roll back reservations.

You might be asking yourself why you'd book the flight and hotel before checking if the payment method was valid? It was a conscious decision because it'd be possible to showcase BPM compensation, which is the rolling back of the web service calls to reserve the flight and hotel in a booking. Sometimes the story must be manipulated to control the results you want to obtain, remember, it's an example project.

This completes the walk-through of the JBoss BPM Travel Agency booking process, the calculations ruleflow, and the compensation process that handles the final booking activities. Next, you look at running an instance of this process from submitting the initial data, examining the progress, completing user tasks, and viewing the final completed booking process results.

### BUILDING AND DEPLOYING TRAVEL BOOKING PROCESS PROJECT

Our final task's to ensure the project's built and deployed in the JBoss BPM Suite server, ready for you to start booking your travel requests. This is done by clicking on the Open Project Editor button found on the left side in the Project Explorer panel. I provide more details later on the elements found in the panel that opens before you, but for now you can click on the menu item in the top right labeled Build, followed by Build & Deploy to start a deployment of the project.

If all goes well, you see a green bar pop-up on the top of the screen that says Build successful. To verify it's deployed, click on the top menu item named Process Management and then Process Definitions. Here you make sure that the three process definitions listed as show in figure 2.16.


**Figure 2.16 When you deploy the Travel Agency, you've three process definitions ready to use.**

Now you're ready to start booking a trip to see how the process, rules, and tasks work.

## 2.2 Booking a trip

The astute reader notices that when you examine the deployed process definitions listing as shown in figure 2.16, there are individual Start buttons. If there are forms defined to start a process, you can click on the button to get this pop-up form to fill with data which is submitted to its respective processes. In this project, you find only the specialtripsagencyprocess has a start task form as was shown in figure 2.8, and you can start booking a trip by using the start button found here.

The more interesting way to start the process is to use the web client application that was deployed when you installed this project. As you saw in figure 2.9, open the booking form at the following link in a new browser tab:

```
http://localhost:8080/external-client-ui-form-1.0
```

This form can be filled in with the following data for the fields shown:

- Applicant Name: [YOUR_NAME]
- Email Address: [YOUR_EMAIL]
- Number of Travelers: 8
- From Destination: London (your only choice)
- To Destination: Edinburgh (your only choice)
- Preferred Date Of Arrival: 2016-08-20
- Preferred Date of Departure: 2016-08-30
- Other Details: N/A

Now you can submit the form by clicking on the red SUBMIT button, wait a bit and you should see the reply from the JBoss BPM Suite after accepting the data. It displays the following response and gives you the process id, which is only done here to demonstrate use of the RestAPI:

```
Thank you for filling out the form. Your request has been processed
     successfully!
Your Unique Process ID is: [2]
Our team will get in touch with you shortly with a Quote..
```

Now you look at where this process instance is, what it's done, and what tasks are waiting, if any. Back in the JBoss BPM Suite Business Central console, select the Process Management and then Process Instances menu entries at the top. You're presented with three tabs labeled Active, Completed and Aborted, with the Active tab already selected in front of you. You see the specialtripsagencyprocess instance with id 2 (remember, your id might be; remember the one returned to you when you submitted the travel booking form).

If you select the completed tab, you'll see that the calculatepriceprocess has already completed, telling you that the process is past that point. Going back to the active tab, click on the specialtripsagencyprocess entry in the list and a small panel appears on the right with tabs for Instance Details, Process Variables, Documents and Logs. For now, you can explore these tabs but I won't detail all of them as I'm most

interested in discovering where the process instance's  at right now. A hint can be found at the bottom of the Instance Details tab, under Current Activities where you see the instance's waiting for a user task to be completed called Price Review.

I can't remember exactly what step in my process that was, but I'm going to open a visual aid that shows me the process model and greys out the path that was taken, while putting a red outline around the task currently awaiting action. This is found in that right panel at the top under the menu Options; click on it to select a Process Model.

This opens a viewer that allows you to use the small buttons on the top left to resize the process until you can see the overview of where this instance's at exactly, as shown in figure 2.17. You might remember that the setup of the process was to ask for the price review if the total price of the travel booking was larger than 2500. With eight people traveling on this booking request, it isn't hard to imagine that a price review's required.

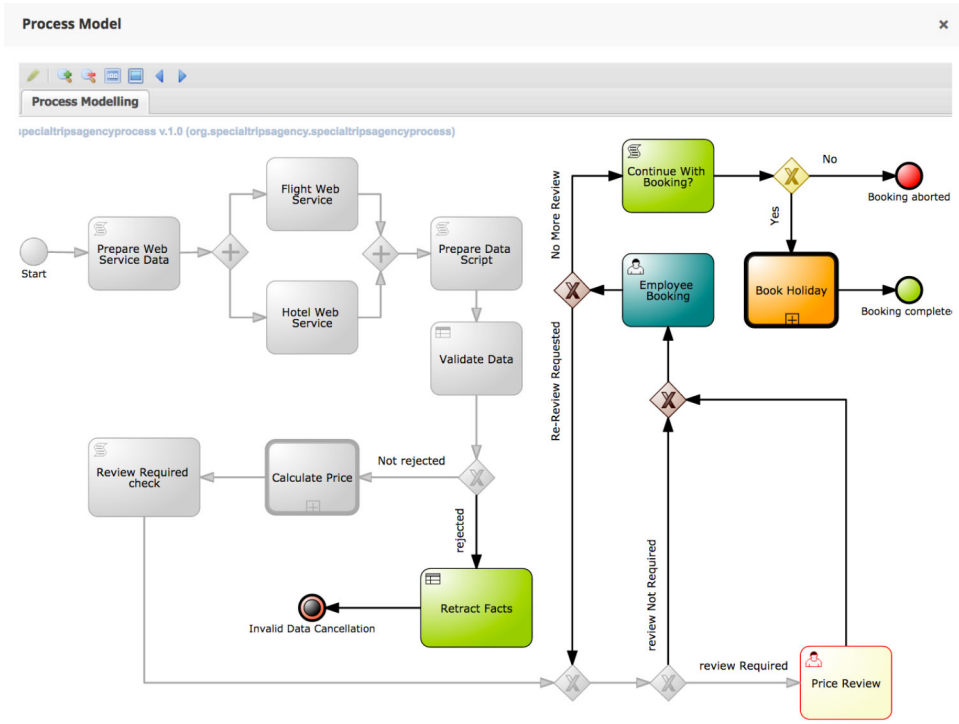


**Figure 2.17 The process model viewer shows where this process instance's greyed out the process steps and a red circle's around the current task awaiting processing.**

You can check the user task waiting to be completed by first closing the process model viewer, then looking under the top menu Tasks, selecting the Tasks List. The view presented contains tabs and each one's a different filtering view of the available tasks currently in the system. First you've the Active tasks, in which you see the task which is waiting. If there are other uncompleted tasks, they'd also appear here.

Under the Personal tab are all tasks that are claimed by the logged-in user. A task can be found under multiple tabs if you look under the Group tab where the same task's waiting to be assigned to a group role that the current user's a member of. The final two tabs are All and Admin. The first contains all tasks, no matter their state, and the Admin contains special tasks for the admin user.

You can claim the waiting Price Review task from the Groups tab by clicking on the Claim button. This opens the task form in a panel on the right and contains all the information mapped into the form fields for the price reviewer to examine before filling out the two fields at the bottom marked with an asterisk. Not all other fields are read-only; some can be modified. Adjust this booking price to give a discount for such a large group and provide the following reason:

- Total Price: (Quote Price) – 5000
- Review Comment (Please summarize the changes for this review) – Group discount

Now click on the blue Complete button at the bottom of the screen to submit your modification and complete this user task. You can guess now that upon submission the process continues onto the second user task, Employee Booking, which appears in the Group task tab. You can verify all of this by re-examining the process instance process model to make sure that the red circle's around the Employee Booking task, an exercise left to you.

Claim the final task left for you by clicking on the Claim button. This opens the task form designed for this task in the right panel. Scroll down and review the read-only data and chose one of the three options; send back for review, confirm the booking by providing credit card payment data, or cancel the booking.

Fill in the following to confirm the booking:

- check the box at the top of the option 2 to select this option
- Credit Card Number: 1111567890 (remember, any card number starting with 1234 is detected as fraudulent, but it's not desired this time around)
- Expiry Date (MM/YY): 08/18
- Name of Card Holder (as it appears on the card): Eric Schabell

Click on the blue Complete button at the bottom of the form to submit the data and complete this user task. You can now go back to the Process Instances and view the Process Model for the main specialtripsagencyprocess to verify that the path to a successful travel booking's been taken, as shown in figure 2.18. It's an exercise for the reader to validate that the compensateService used in the sub-process to finalize the booking details has completed correctly and resulted in a travel booking.

Now let's take a look at the reporting available to review what's transpired during the process instance executions. Before continuing, I'm going to start more process instances with active data and in various wait states alongside the completed process instance. This makes for more interesting graphics when reviewing the dashboards and reports that JBoss BPM Suite offers.

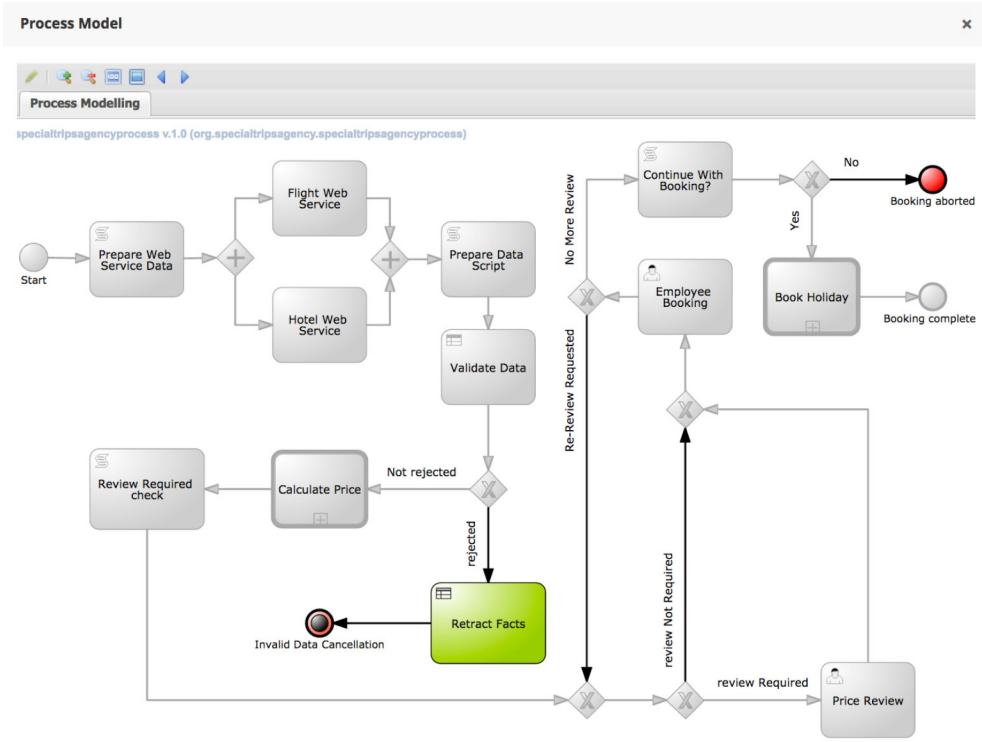


**Figure 2.18** **By viewing the Process Model you can verify that this process instance has completed the path to a successful travel booking.**

## 2.3 Reviewing the booking

This section covers the basics of reviewing the current status of running process instances and provides an overview of the history of completed or aborted process instances. It helps you to keep track of the running JBoss BPM Suite that you're working on, and showcases a few of the standard graphics, overviews, and reports that make up the reporting dashboards.

### 2.3.1 Process & Task Dashboard guided tour

The dashboard that shows the status of your running JBoss BPM Suite's found under the top menu Dashboards, where you can select Process & Task Dashboard. This opens the top-level view of a report laid out into several sections or panels. This is a standard setup which is provide out of the box and you can't modify this.

On the top left you see the tab labeled Processes. Under that tab's a row of process counters, for the various states that a process instance can be found in. Below that you see a set of six reports spread across the page. These are various views of the process instances sorted by one of the following criteria:

- By Type
- By Start Date
- Started by User
- By Running Time
- By End Date
- By Version

These can be explored by hovering and clicking on parts of the reports to gain more insights. For example, hover over part of the pie chart in Processes by Type and you see the process name that ran it and the number of instances created for that process definition.

If you select the next tab, labeled Tasks, you're presented with an identical layout of information about tasks from the process instances. Feel free to explore more of the offered dashboards and reports, and if you dig too deep you can reset any of the reports with the link found under the individual reports title that appears when you drill down the first time. It's labeled reset.

Finally, both on the Processes and Tasks tabs, click on the link in the top right corner. For processes, the link Show Instances takes you to a list of all process instances and their details. For tasks, this link Show Tasks takes you to a list of all existing tasks. To return from either, click yet again on the link in the top right labeled Show Dashboard.

This completes the examination of the Process and Task Dashboard. Although it isn't all encompassing, there are enough guidelines for you to delve deeper into your own process instances. Feel free to explore more of the data and history provided by these reports on your own.

## 2.4 Summary

- A process contains the tasks or steps defined to complete a piece of business value, such as completing a travel booking from the initial inquiry to the final paid ticketed travel booking.
- Process designer's the component used to create, edit and review any of the three processes found in the example project; *specialtripsagencyprocess, calculatepriceprocess* and *compensateService*.
- Script tasks provide for coding tasks and are often used for small maintenance tasks in a process flow; for example, to manage process data and update process variables.
- User task in the process designer can generate task forms automatically and the form designer's the tool to customize the user task forms which collect user input data.
- Service tasks are used for backend service integration where external systems are called to provide the process with information or to contact systems external to the process.

- Rule tasks tie in business logic by passing data to the rule engine and applying the resulting changes to data back into the process variables. This is done with technical rules, guided rules, and decisions tables in the travel booking process.
- XOR gateways are moments in a process for deciding which path in a process to take and are often used following a rule task where decisions have been made. The outgoing sequence flows are used to check process variables that determine the path to be taken.
- Parallel gateways are used to split the process into multiple paths that execute at the same time.
- Sub-processes are process flows that allow one process to start a separate process and wait for that process to complete before continuing from the point at where the sub-process was called.
- Ruleflows are processes that contain only rule tasks and the travel bookings example project has one for determining the pricing of the booking request, which is called *calculatepriceprocess*.
- Data validation's done with DRL rules, pricing discounts are done with a decision tables, and various guided rules are used to maintain business logic around the travel booking project.
- Process projects can be built and deployed easily with a single click of a button, done before you start your first instance of a travel booking process.
- Starting a process can be done within the process management tooling in business central, or through external web applications, by submitting data provided by the user of the application.
- Running through a process from submitting initial data, to viewing the status of a running instance, to processing user tasks and viewing the final results, which are achieved with the provided business central process management runtime tooling.
- Process and task dashboards are provided by JBoss BPM Suite to view, examine and drill down into process instance and task data from the travel agency project.

# 3 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[1] 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.

[1] 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.

## 3.1 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.

### 3.1.1 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*[2]. 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).

[2] 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.

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

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

4 Start demo container:

```
docker run -it -p 8080:8080 -p 9990:9990 effectivebpmwithjbossbpm/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 3.1.

```
[ 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 effectivebpmwithjbossbpm/chapter-3-easy-install-demo .
Sending build context to Docker daemon 703.6 MB
```

**Figure 3.1** **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 3.2. 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 effectivebpmwithjbossbpm/chapter-3-easy-install-demo
```

**Figure 3.2** **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 3.3.

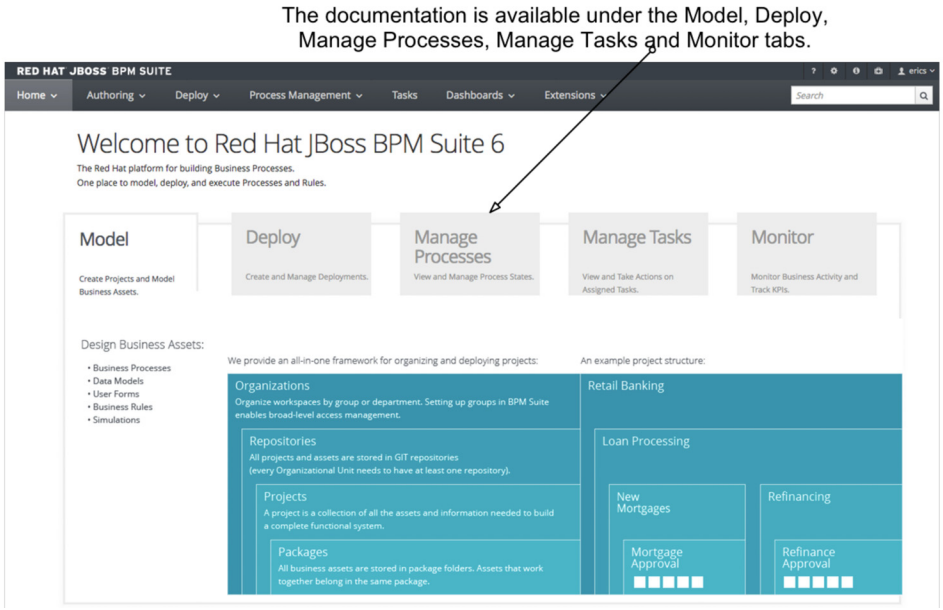


**Figure 3.3 After logging in to the Business Central you'll be at the home screen where you can browse the documentation. This home screen's where you start your work on BPM projects.**

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.

## 3.2 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.

### 3.2.1 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 3.4.

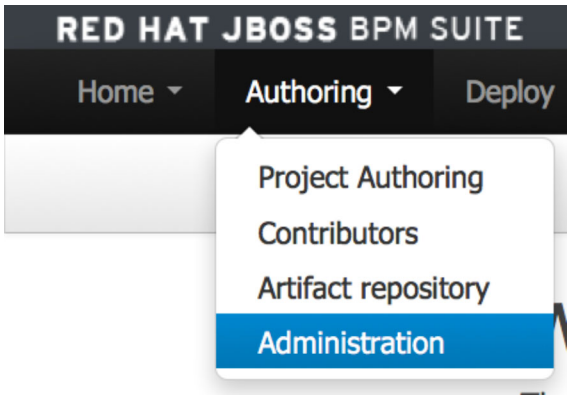


**Figure 3.4 The Authoring menu contains the Administration item that leads to the Administration perspective where you start your project setup.**

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 3.3:

- *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.

### 3.2.2 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*.

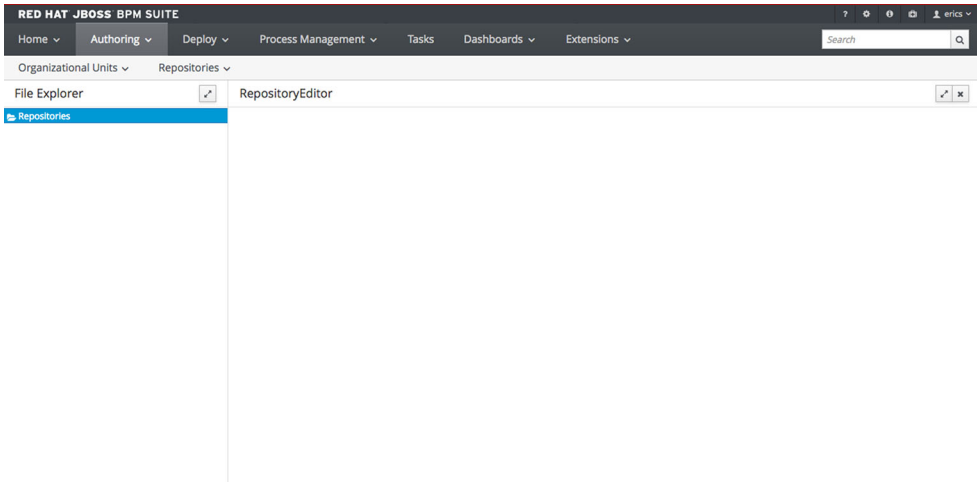


**Figure 3.5 The initial Administration perspective view's empty, nothing has been setup yet. You start by setting up your organization using the menu in the top right labeled Organizational Units.**

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 3.5. 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 3.6.

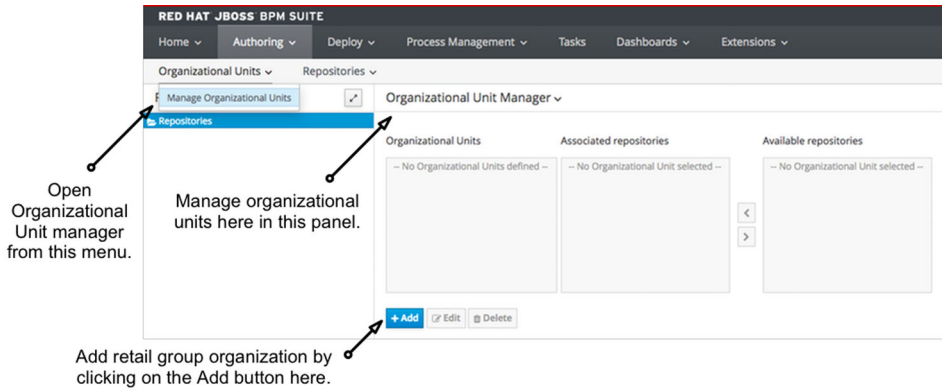


**Figure 3.6** **In the administration perspective, you start by setting up your organizational structure for projects. Open the Organizational Unit Manager by accessing it through the menu. To add your retail group organization, you click on the Add button as shown.**

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 3.6 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.

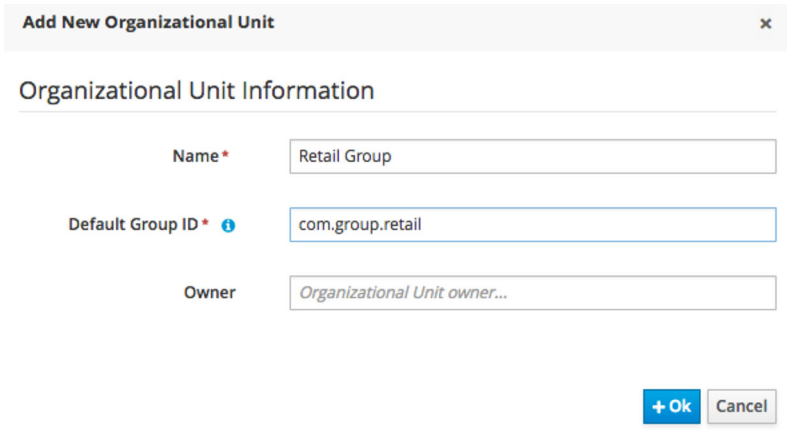


**Figure 3.7** **The organizational unit information pop-up provides you with the fields to add *Retail Group* and a default group ID of *com.group.retail* while leaving the owner field blank.**

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.

### 3.2.3 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 3.8.

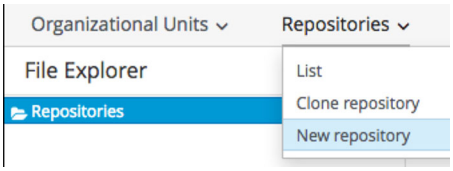


**Figure 3.8 Several options are available to you when looking at the Repositories menu. You're creating a brand-new repository.**

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 3.9, 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.

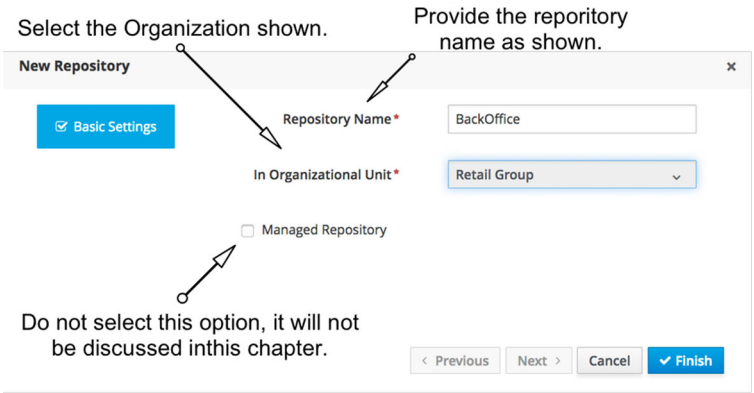


**Figure 3.9 When adding a new repository, you're shown this pop-up in which you enter a repository name and then select the organizational unit.**

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 3.10. 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
```

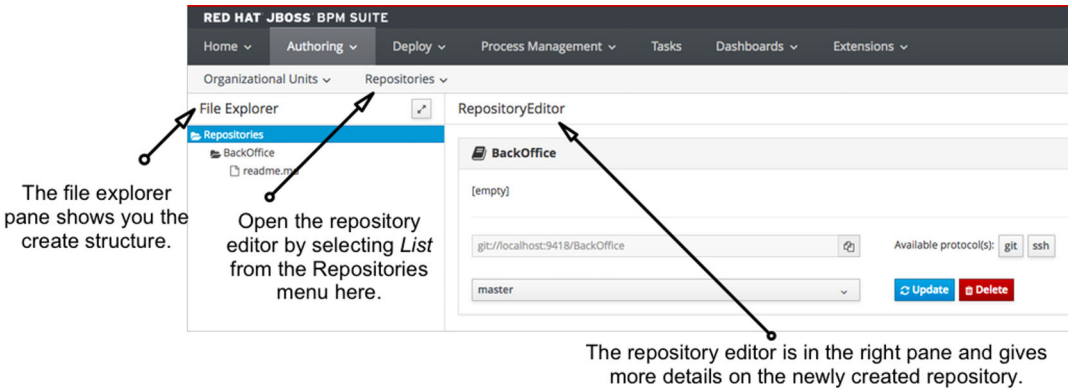


**Figure 3.10 Once the *BackOffice* repository has been created it's visible in the left *File Explorer* pane. You can open it in the *Repository Editor* by clicking on the folder *Repositories* or by selecting the *Repositories* menu entry *List*.**

This can be used by developers to check out a local copy of this repository[3], 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.
```

[3] 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 -1 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.

### 3.2.4 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 3.11 shows you the *Project Explorer*, the left panel on your screen where you see the menu structure in the form of:

```
ORGANIZATIONS - REPOSITORIES - PROJECTS
```

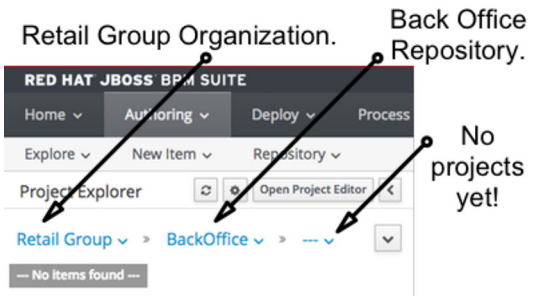


**Figure 3.11 After using the *Authoring* menu and selecting the *Project Authoring* perspective, you're shown the *Project Explorer* with your structure.**

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 3.12, all other available assets are greyed out and you can't select them. The only option available's the *Project* item.

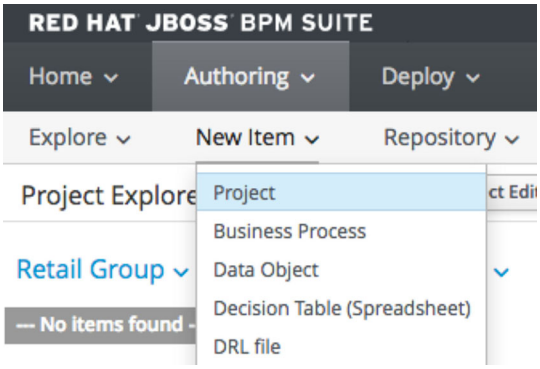


**Figure 3.12 The *New Item* menu's used to add any of the assets you find listed. As you currently don't have a project defined, all the assets except *Project* are greyed out and unavailable. Select the *Project* to reach a pop-up labeled *Create new Project* where you can enter a project name.**

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 3.13. It contains the project name, a description and some details which are directly related to the fact that the

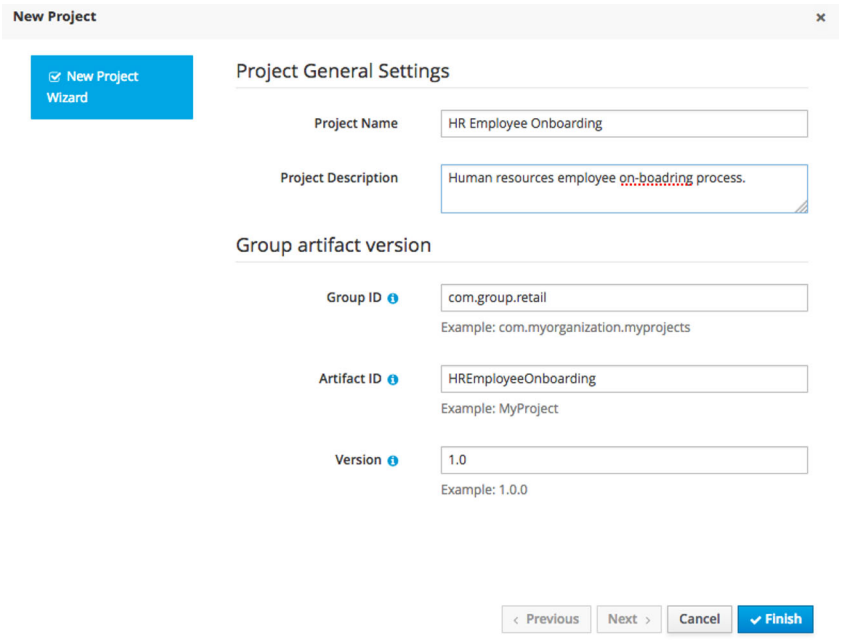


**Figure 3.13 This is the New Project pop-up where you can fill in all the information and details for your new project. Some fields are pre-filled based on previously entered information, and others can be filled or modified as desired. The last three are related to JBoss BPM Suite using Maven as a 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 3.14.

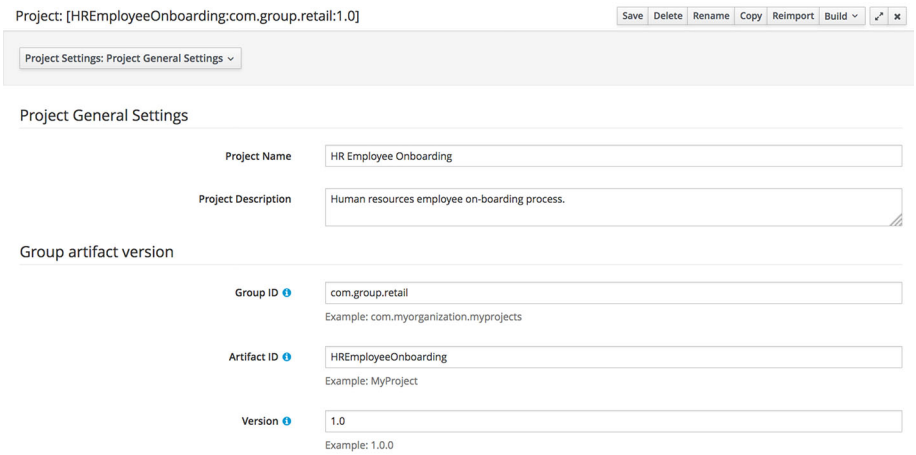


**Figure 3.14 After creating a project, the Project Editor's displayed with project details.**

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 3.15. 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 3.16 where you can determine how you want to view to look in the Project Explorer.

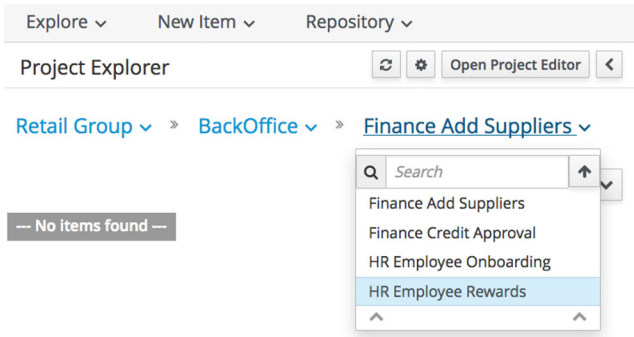


**Figure 3.15** From within the Project Explorer you can view all available projects from the drop-down menu. Here you see all four of the projects you've created. If you select one it opens, showing the Project editor in the right pane and the project assets listed in the Project Explorer pane on the left.

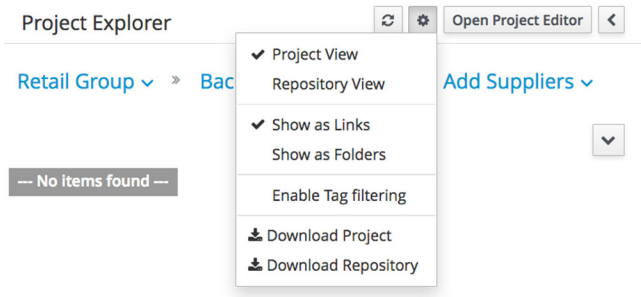


**Figure 3.16** The Customize View button produces a menu where you can adjust the view and how a project's shown. The Project View's the default, and the Repository View exposes project files as if you're viewing the file system.

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 3.17.

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 3.16 are *Show as Links* (the default) and *Show as Folders*. Selecting them appears to change nothing, as

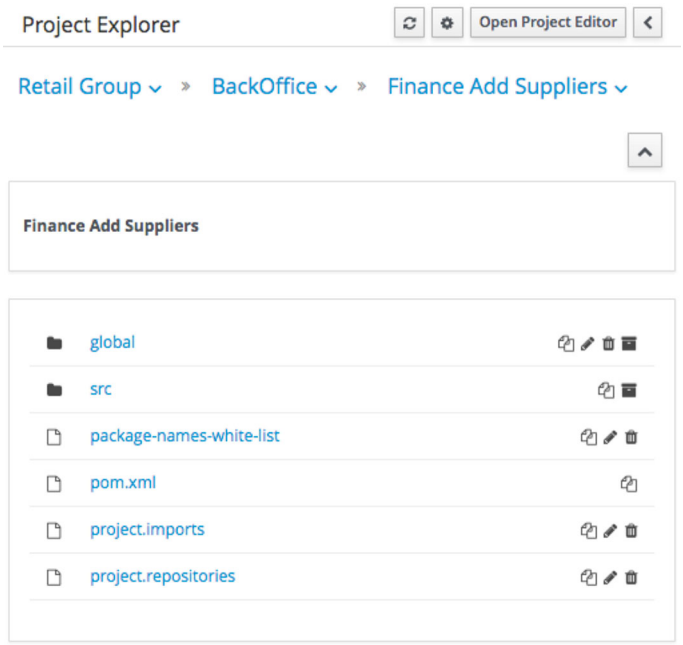


**Figure 3.17** **This view in the Project Explorer's called *Repository view* and exposes all files in a project as if you're viewing the file system.**

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 3.18 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.

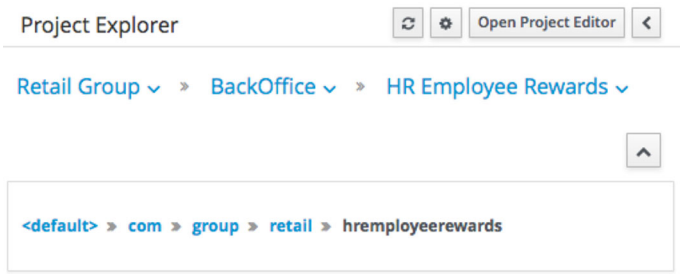


**Figure 3.18** **This is your view of a project in the Project Explorer after you configure your view to show the HR Employee Rewards project as links. You can click on the project structure from com – group – retail – hremployeerewards to expand the linked view down into the project structure. Any 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 3.19.

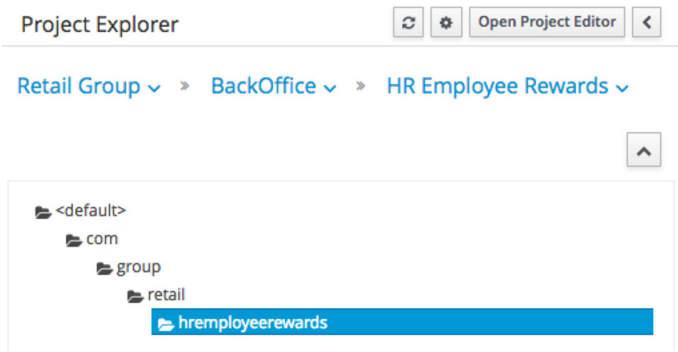


**Figure 3.19 This is your view of a project in the Project Explorer after you configure your view to show the HR Employee Rewards project as folders. You can click on the project structure from com – group – retail – hremployeerewards to expand the folder view down into the project structure. Any artifacts 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 dropdown menu in your Project Explorer, by selecting the menu items *Download Project* or *Download Repository* respectively. Depending on your browser settings you might 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's 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.

### 3.2.5 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) 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.

## 3.3 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.

### 3.3.1 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.

### 3.3.2 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.[4]

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.

### 3.3.3 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.

---

[4] 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.

## 3.4 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.

# 4 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.

## 4.1 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[1], 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 4.1, from the example project described above and I'm going to walk you through implementing it in JBoss BPM Suite.

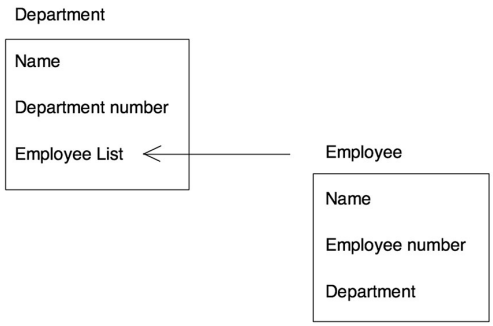


**Figure 4.1 This example data model has been given to you for implementation in JBoss BPM Suite. It consists of two data objects; Department and Employee.**

### 4.1.1 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 4.2.

---

[1] A formal definition can be found at 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."

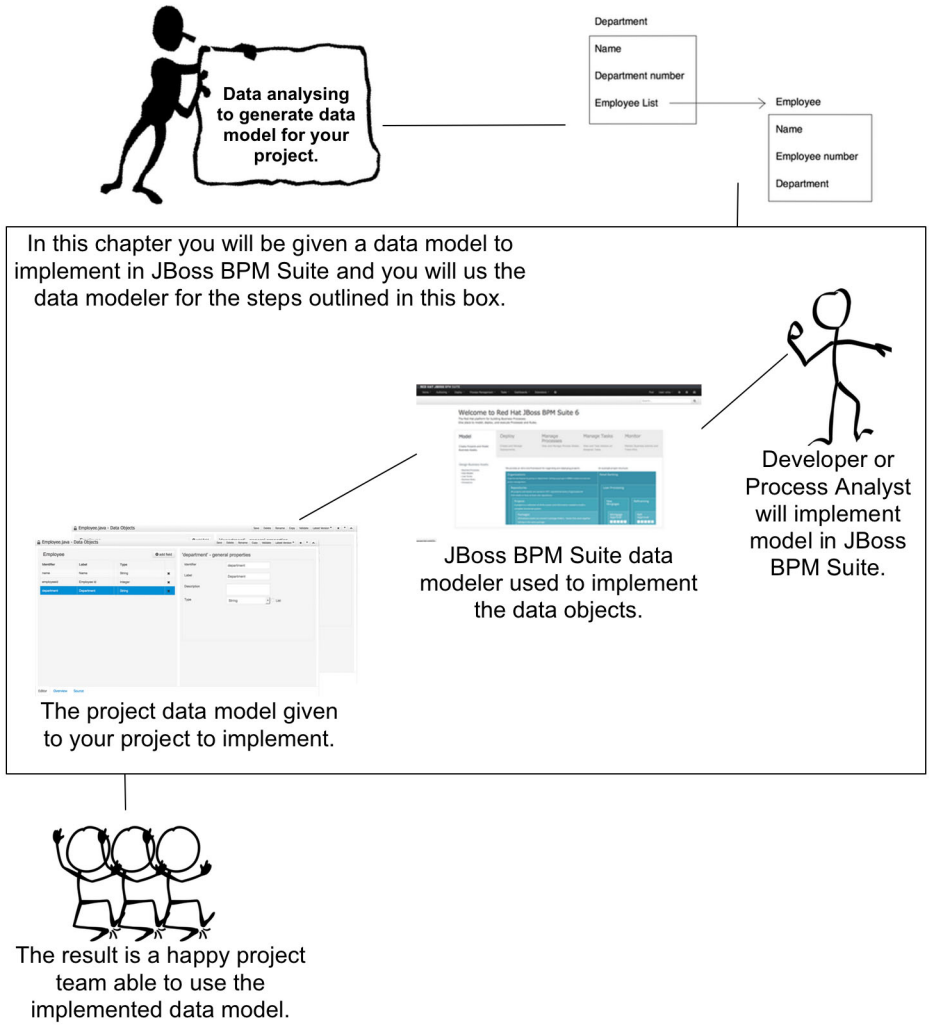


**Figure 4.2 An overview of the path you'll take in this chapter's outlined. Inside the box are the steps to be covered, from the moment you receive a data model design, through implementation in JBoss BPM Suite, to the final data objects in your project for use by the rest of your team.**

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 4.3. 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 4.4; select *Project Authoring* to open the perspective where you see the data modeler.

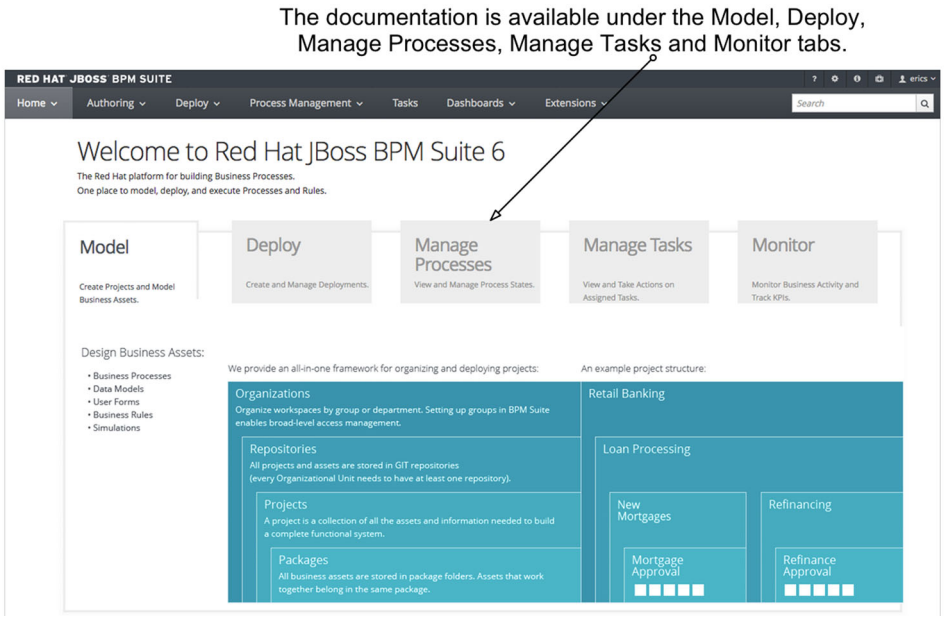


**Figure 4.3** **After logging into the Business Central you'll be at the home screen where you can browse the documentation. This home screen's where you start your work on data modeling.**

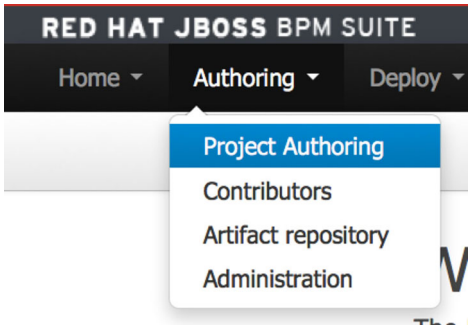


**Figure 4.4** **The Project Authoring perspective can be found in the Authoring menu and opened by clicking on the Project Authoring entry. This'll get you to the various authoring tools for any BPM projects you're working on.**

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 4.5, 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,

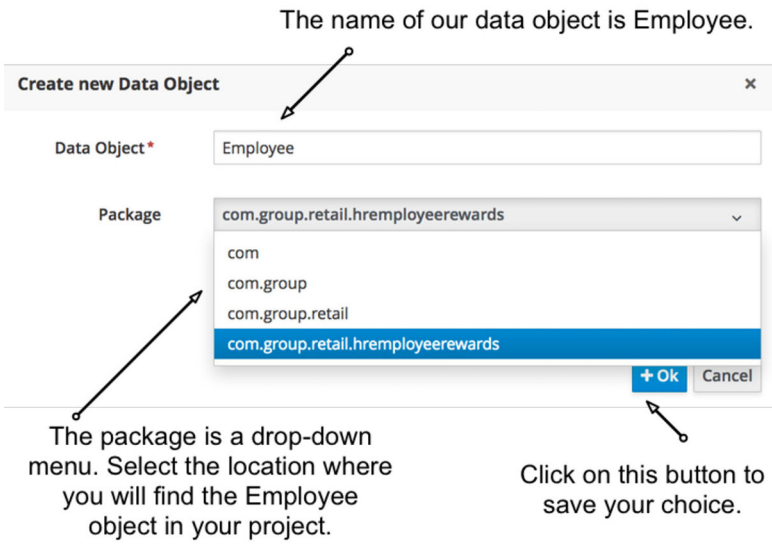


**Figure 4.5** **Use this pop-up to create a new data object.**

*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/hremployerewards/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 4.6, 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.

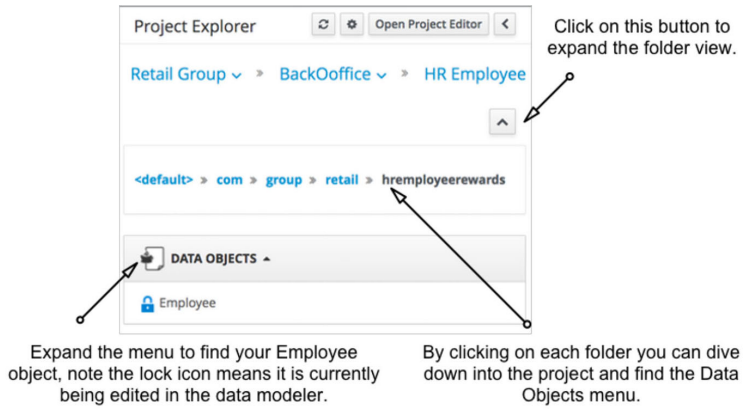


**Figure 4.6 Data objects are put into packages in which they appear hidden from the main Project Explorer default view. To find them you need to expand your folder structure.**

### 4.1.2 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 4.7 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.

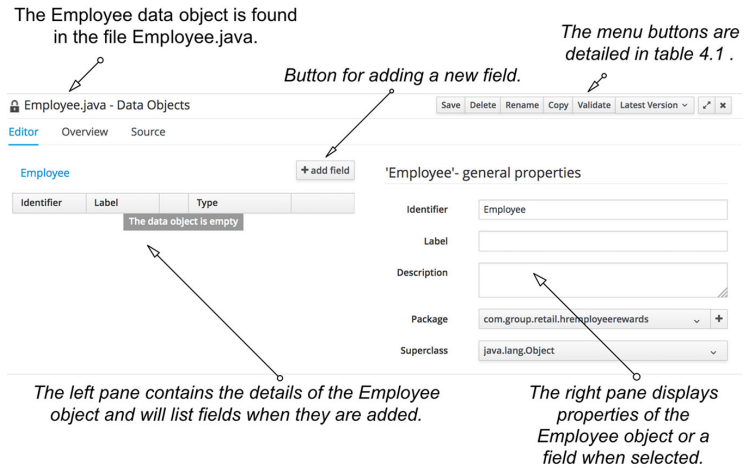


**Figure 4.7 The JBoss BPM Suite data model editor where you create, modify and view data objects.**

You'll use a few menu bar buttons while creating and updating your data objects. Let's look at their actions in table 4.1.

**Table 4.1 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.

### 4.1.3 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 4.8.

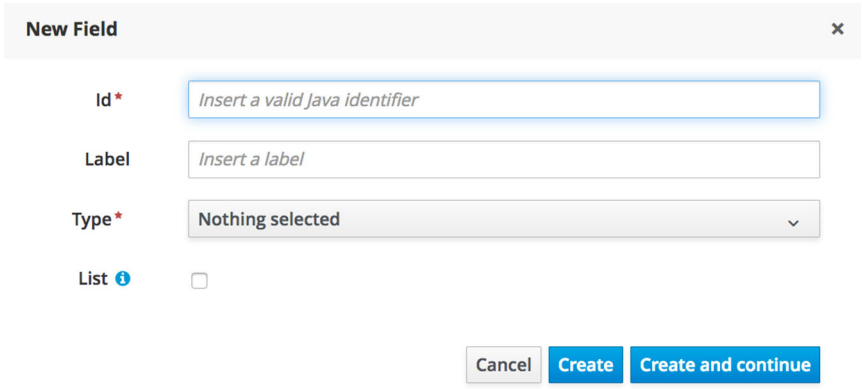


**Figure 4.8 The *New field* form pops-up when you click on the *+add field* button in the data model editor.**

This pop-up's asking for the values that you can enter with the following values shown in figure 4.9:

New field

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

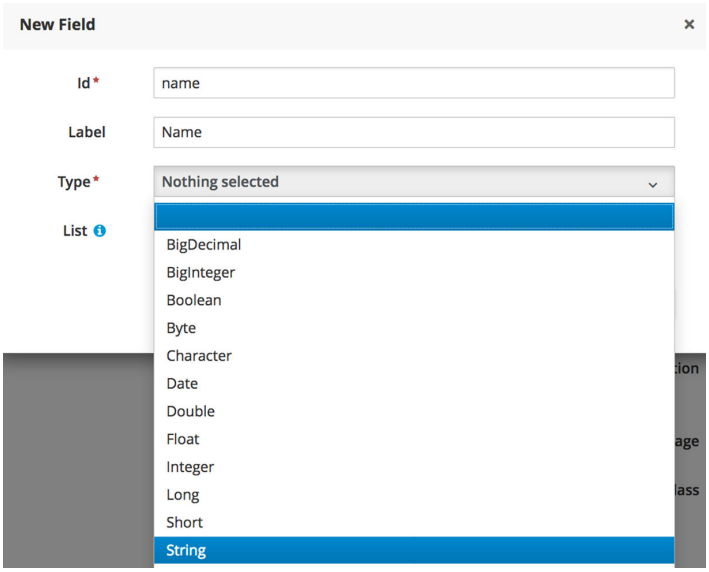


**Figure 4.9 This is the entry form called *New field*, used to add relevant information when adding a field to an existing data object. Here's a field with an *id*, *label* and *type* with the values *name*, *Name* and *String* are being created.**

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 4.9 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.hremployeerewards.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 4.10.

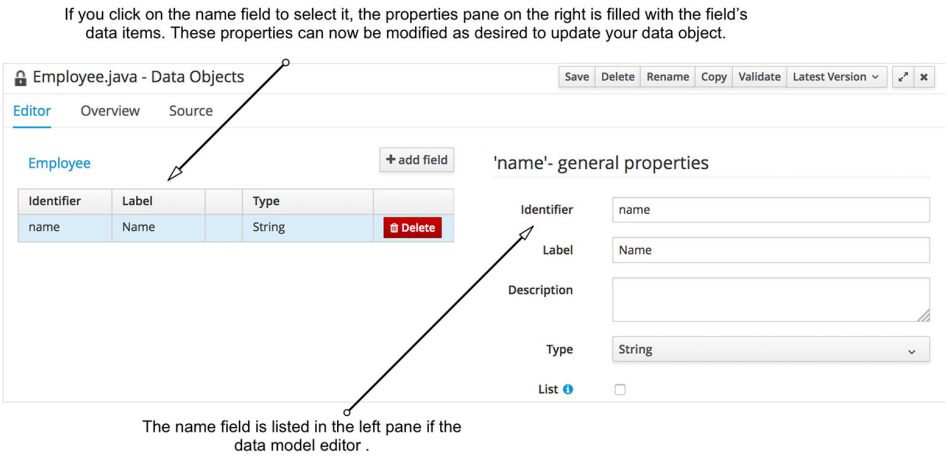


**Figure 4.10 After adding the first field *name*, it appears in the data modeler.**

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 4.11 and 4.12. 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 4.10 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 4.12:

Employee id field

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

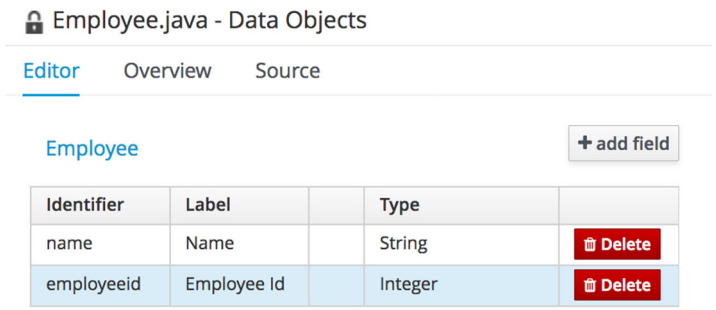


**Figure 4.11 The Employee object in the data model editor should look like this when you finish adding in the employee id field.**

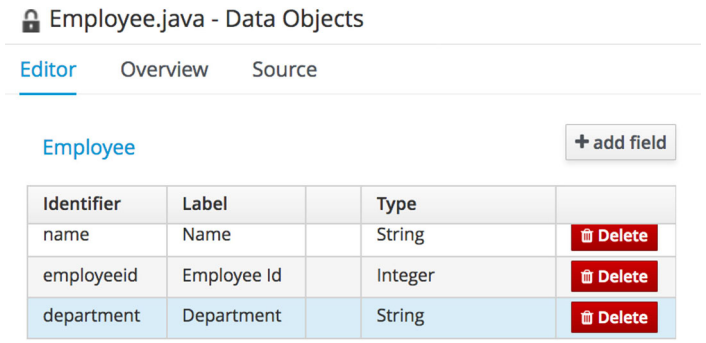


**Figure 4.12 The Employee object in the data model editor should look like this when you've added the last department field.**

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.

### 4.1.4 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 4.13. Note that this view contains the same main header and menu buttons on the top left or right as was covered in the editor.

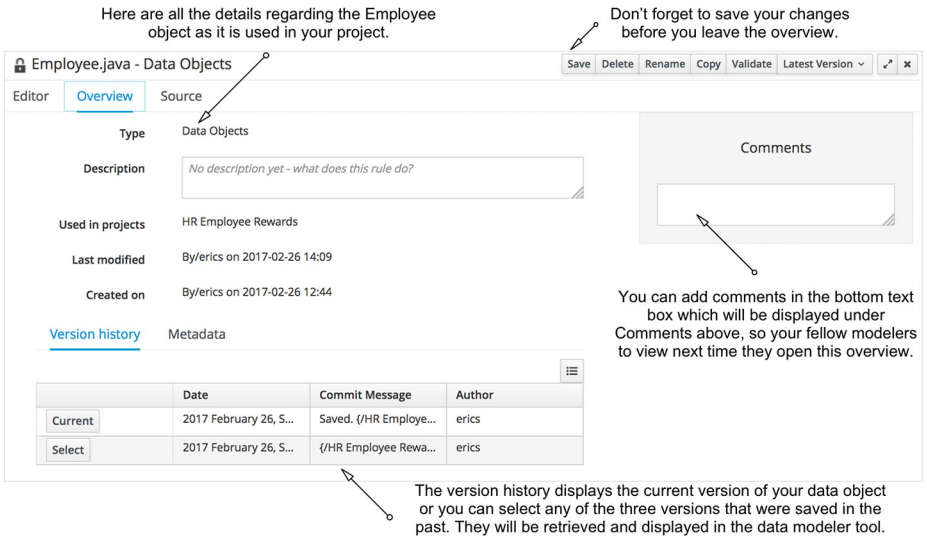


**Figure 4.13 The data modeler Overview tab provides all the details around the currently selected object. In our case, it's the Employee object.**

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 4.14. 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

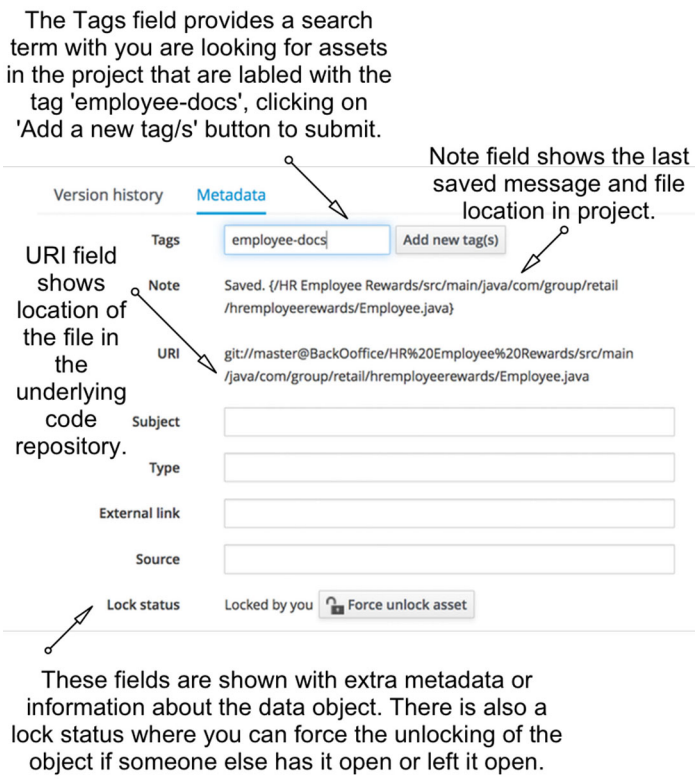


**Figure 4.14 The tab labeled Metadata provides you with the chance to add extra information about the current data object.**

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.

### 4.1.5 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 4.15 you can see an example of a portion of the Employee object source code.

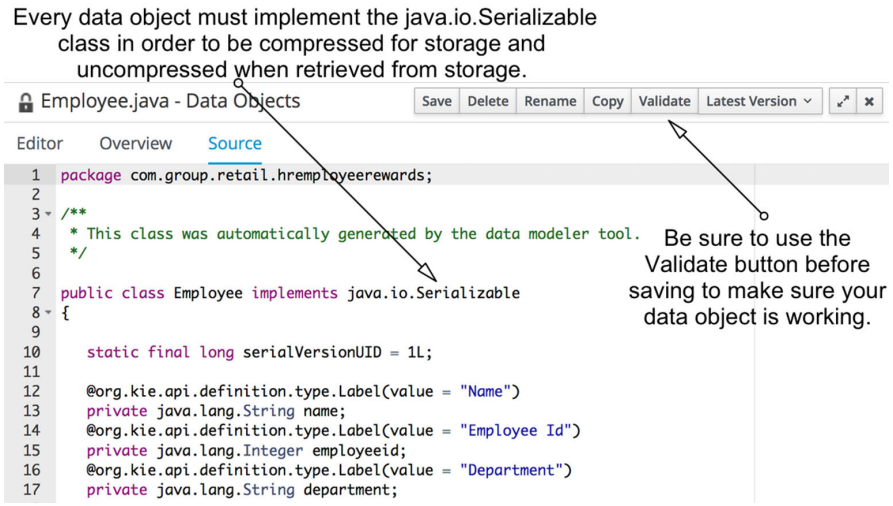


**Figure 4.15 The *Source* editor's the tab that provides a developer view of the actual Java source code that makes up your Employee data object.**

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 4.15, 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 4.16.

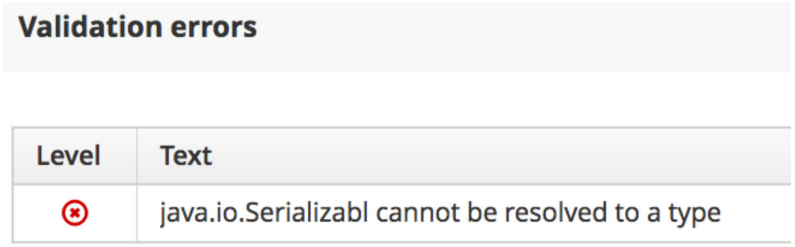


**Figure 4.16 If you're going to be editing a data object's *Source*, you'll need to use the *Validate* button to ensure you haven't broken your data object. This pop-up shows that you've broken your data object by removing the 'e' from line seven in figure 4.15. The only way to fix this is to click 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.

## 4.2 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.hremployeerewards
- Persistable : (leave check box empty)

The new Department object should look like figure 4.17.

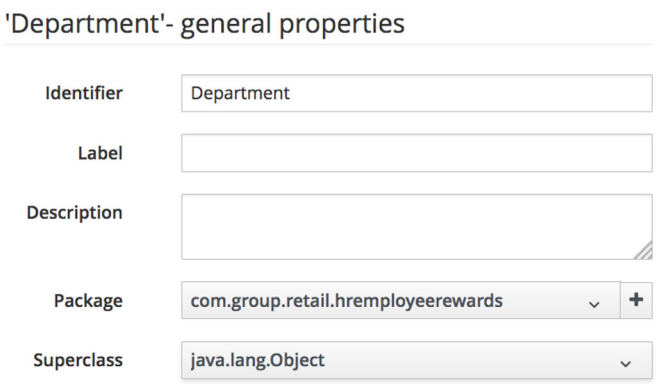


**Figure 4.17 The Department data object as shown in the right pane properties view.**

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.hremployeerewards.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 4.18.

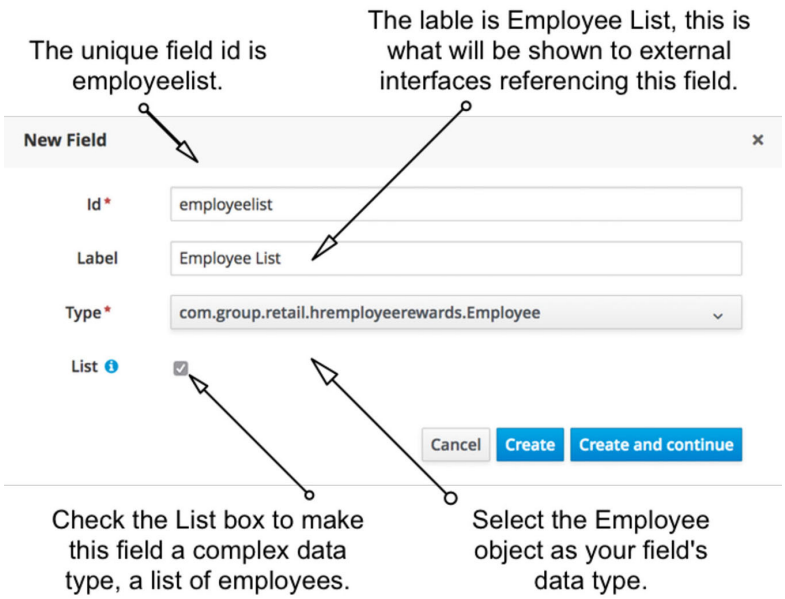


**Figure 4.18** **The final field for your Department object's an advanced type that uses the Employee object previously created, and it's part of a list. You'll need to check the box next to *List*.**

This completes the Department data object with the fields name, department id and employee list. Save your Department object, it should look like figure 4.19. 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!

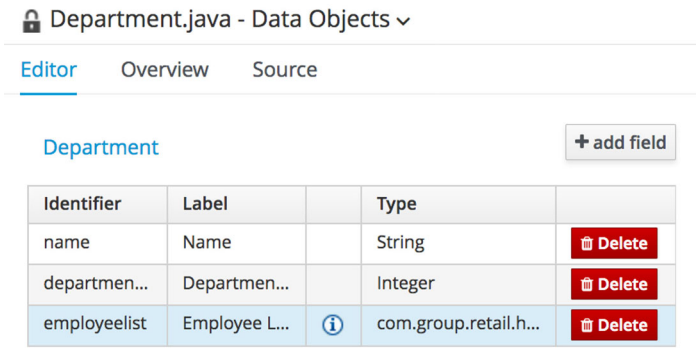


**Figure 4.19** **This is what your Department data object should look like (the order may differ for you) after completing all the fields you needed to create, including the advanced list of employee objects for the Employee List field.**

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.

### 4.2.1 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.

### 4.2.2 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 4.19. 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 Serialized class as covered in section 4.1.5, and be built into a JAR file.

### 4.2.3 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 4.20.

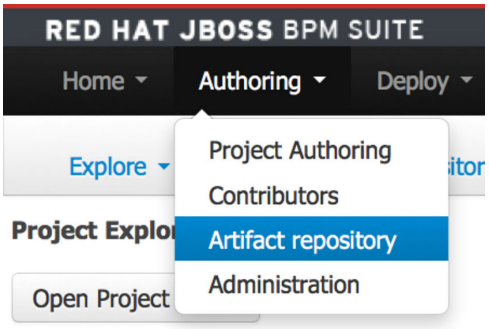


**Figure 4.20 The Artifact repository view can be found in the Authoring menu and it opens a list view of everything currently available to your BPM projects. Click on Authoring and then Artifact repository to open.**

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.[2] 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 4.21.

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.

---

[2] The *ACME Travel Data Model Project* (https://github.com/effectivebpmwithjbossbpm/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 4.21** **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.**

### 4.2.4 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 4.4, then click on the *Open Project Editor* button and open the *Project Settings* menu to select the *Dependencies* view as shown in figure 4.22.

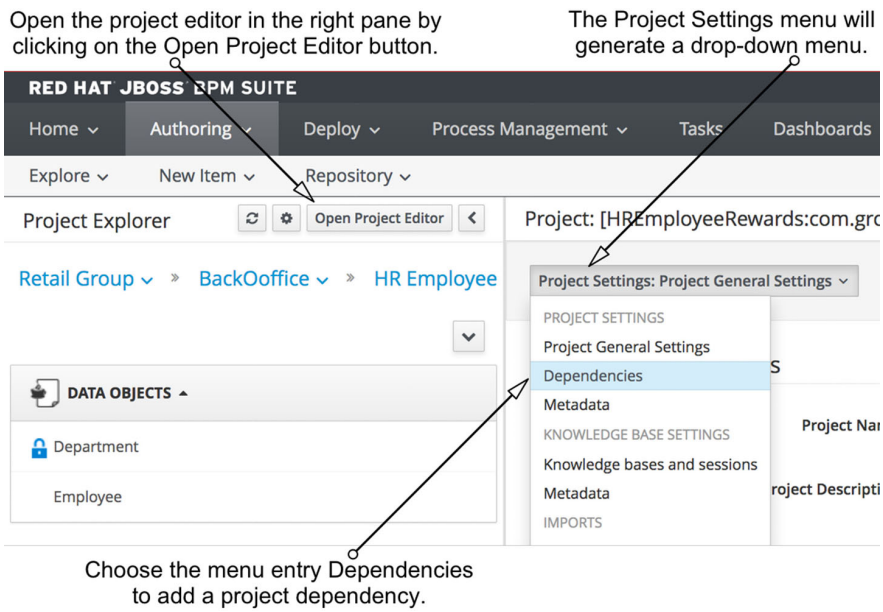


**Figure 4.22** **From the Project Authoring perspective, you'll be able to add the dependency on our imported external data model.**

This opens the project editor's *dependency view*, which is currently empty but has options to add dependencies to this project. In figure 4.23 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.

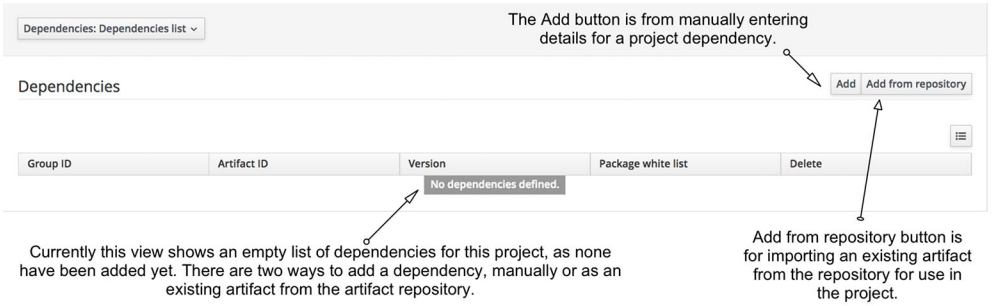


**Figure 4.23 The project editor's dependency view, used to view and add dependencies to your project.**

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 4.24. 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.

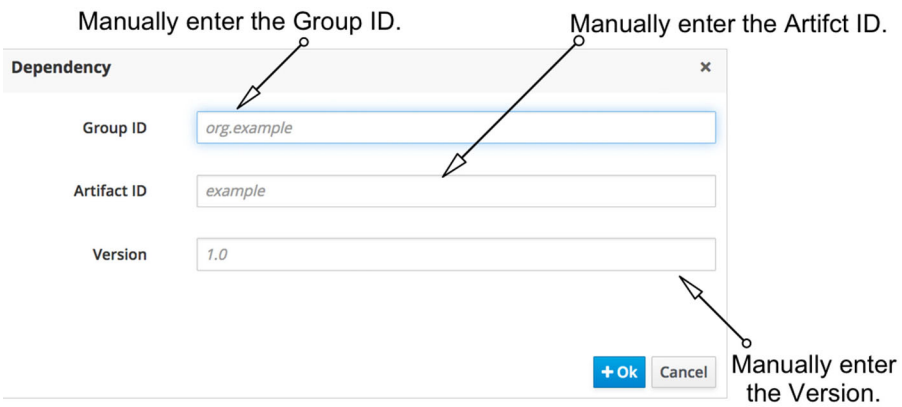


**Figure 4.24 The *Add* button generates this pop-up where you'll need to manually enter each item asked for to complete the entry.**

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 4.25 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.

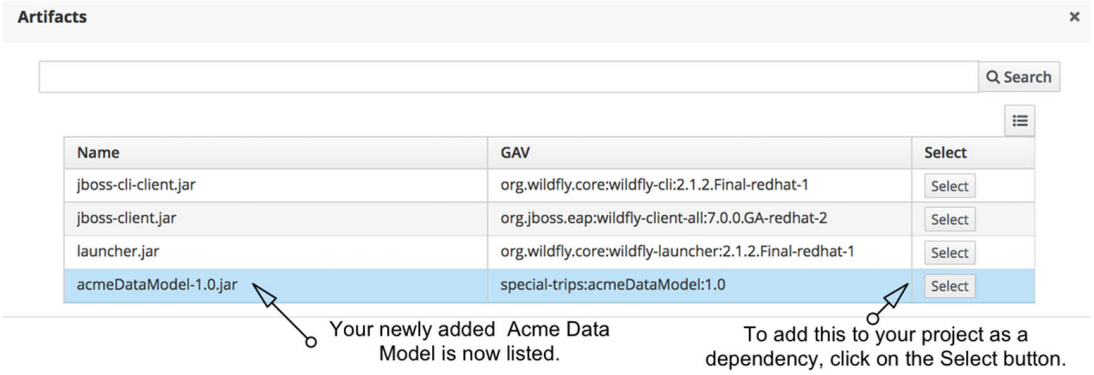


**Figure 4.25 The list of artifacts available in your repository are shown here with your newly imported acmeDataModel-1.0.jar waiting to be selected.**

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 4.26.

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.

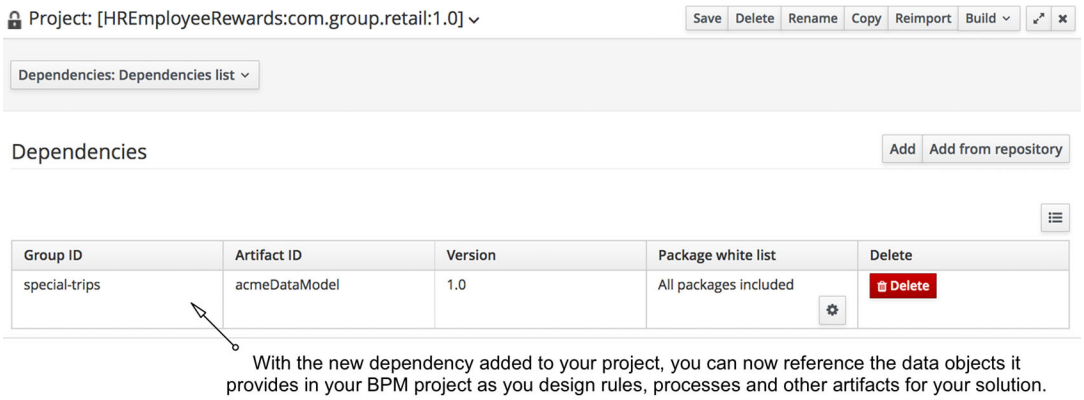


**Figure 4.26 The Acme Data Model dependency has been added into the list of dependencies for your project with all the details automatically filled in for you.**

## 4.3 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.

# 5 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[1] 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[2]. 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 5.1 shows the path taken to learn how to use the various guided editors and tooling to implement externalized business logic for your organization.

## 5.1 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-

[1] 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.

[2] 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

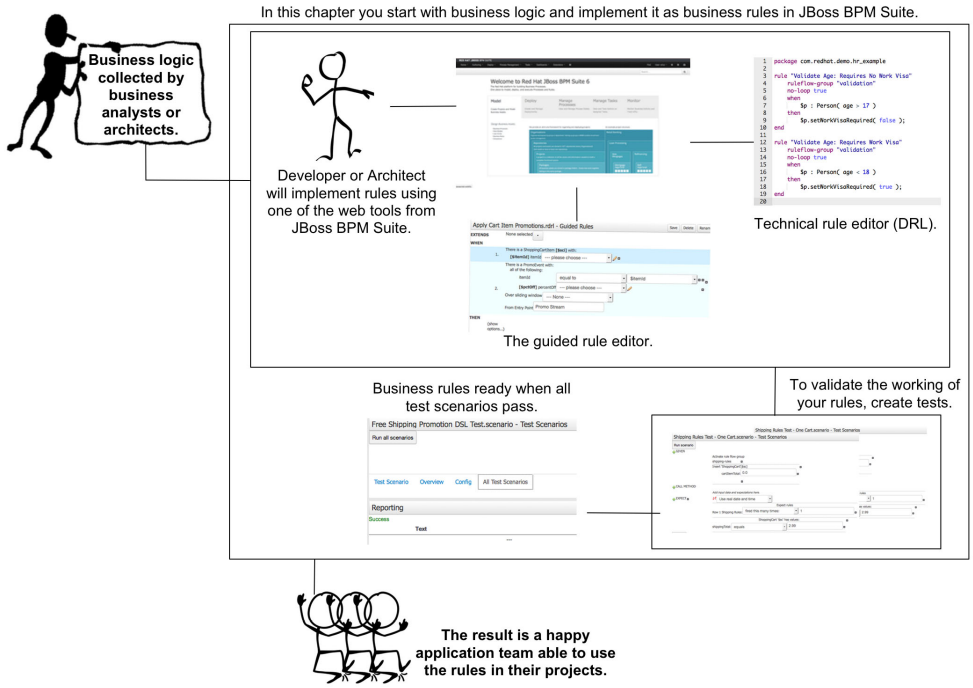


**Figure 5.1 The overview of the path you take in this chapter's outlined here. With the provided business logic, you implement business rules leveraging JBoss BPM Suite.**

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.

### 5.1.1 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 higer
    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 5.2.

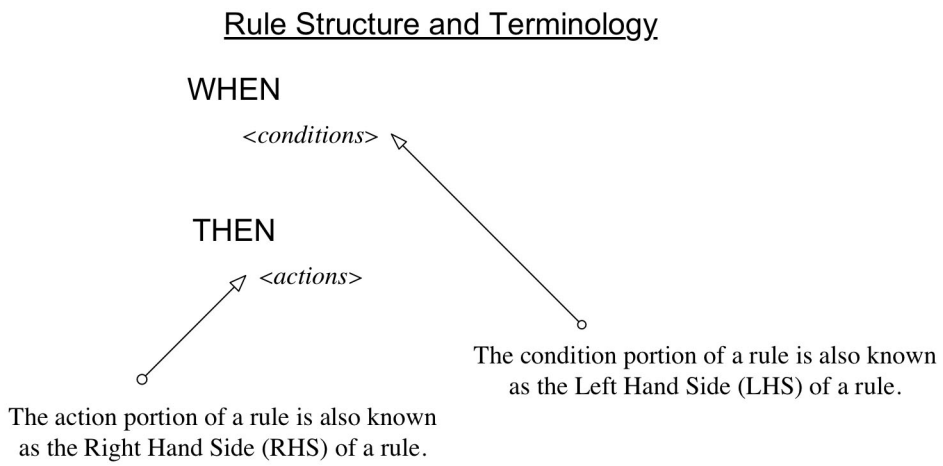


**Figure 5.2** **The basic structure to a rule includes new terminology such as LHS and RHS.**

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.

### 5.1.2 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 5.3 shows you an architectural overview of the inference engine as it is used in JBoss BPM Suite.

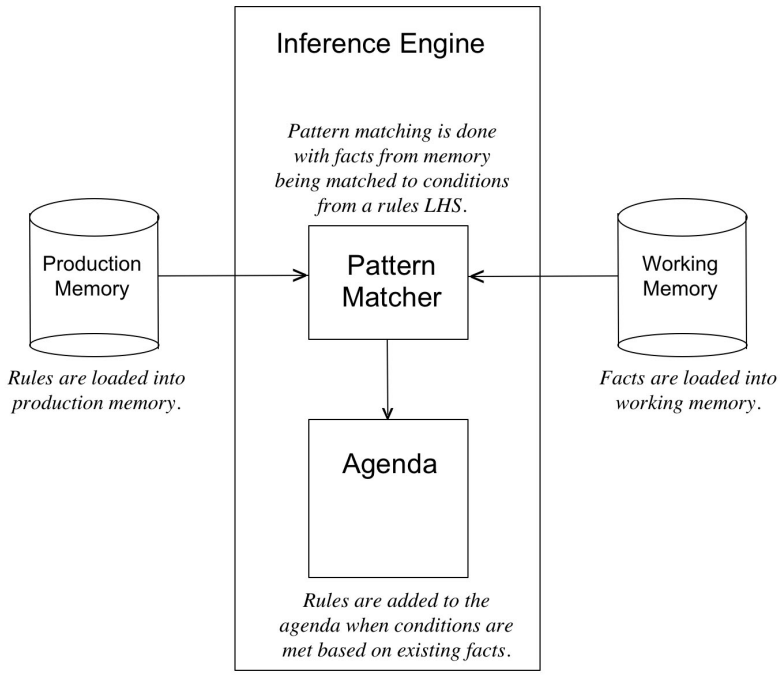


**Figure 5.3 The inference engine in architecture used by JBoss BPM Suite.**

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 5.4 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.

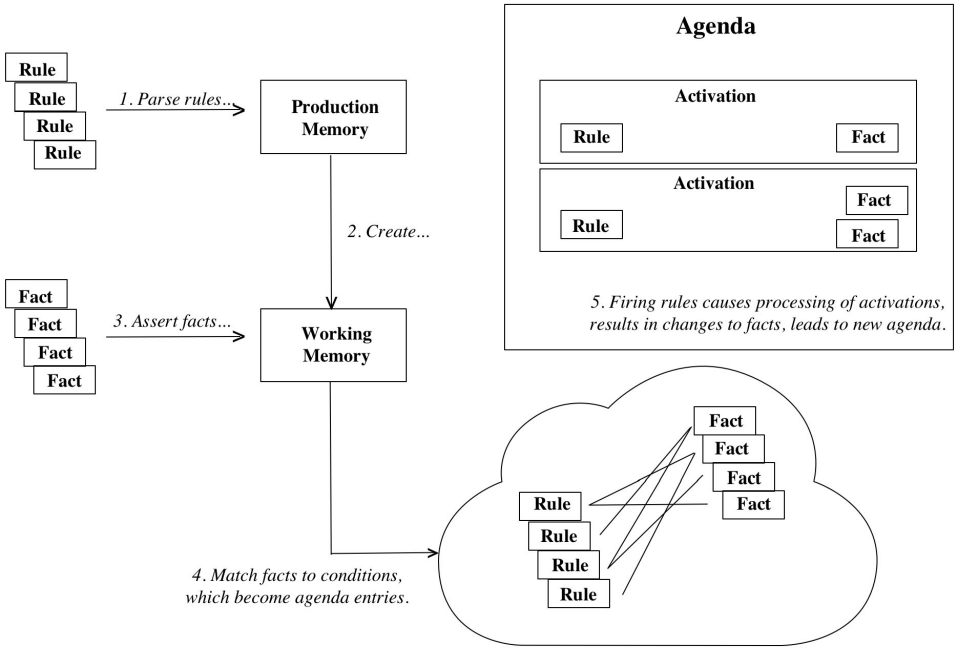


**Figure 5.4 The overview of the JBoss BPM Suite processing of rules, facts and agenda to determine which rules conditions are met and how they're activated.**

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.

## 5.2 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.

### 5.2.1 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[3] 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 5.1.1 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/effectivebpmwithjbossbpm/chapter-5-rules-demo) for this chapter and note, as shown in figure 5.5, the data model already set up.

If you need to review how to open the Project Authoring perspective, the please review chapter three. The first data object you can review's the Document object, which opens when you click it, looking like figure 5.6.

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.

---

[3] 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.

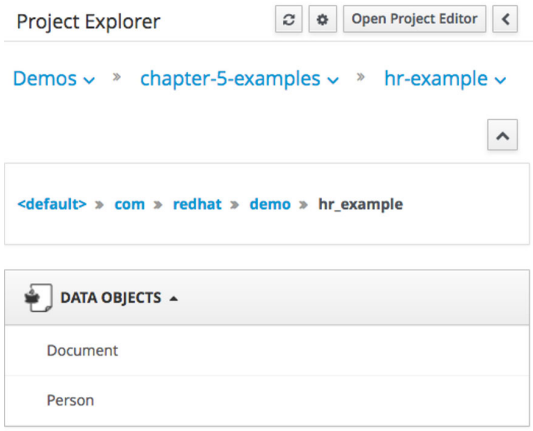


**Figure 5.5 The project explorer here shows you the hr-example project data model found in com.redhat.demo.hr_example with a Document and Person object defined.**

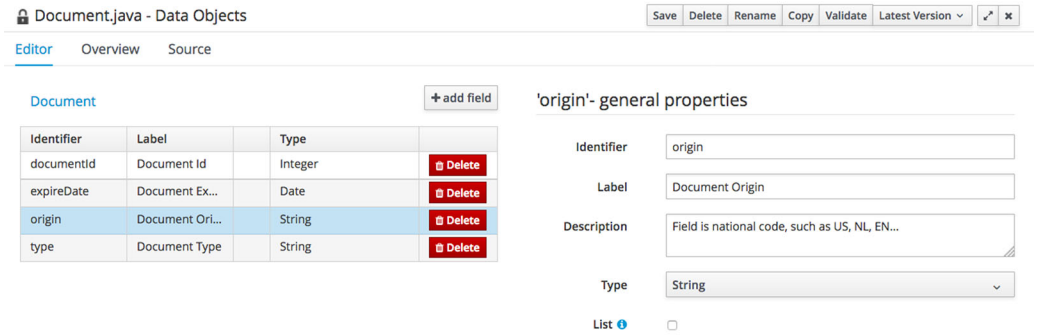


**Figure 5.6 The Document data object in the HR example.**

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 5.7 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 5.6 which allows us to embed the Document object inside the Person object.

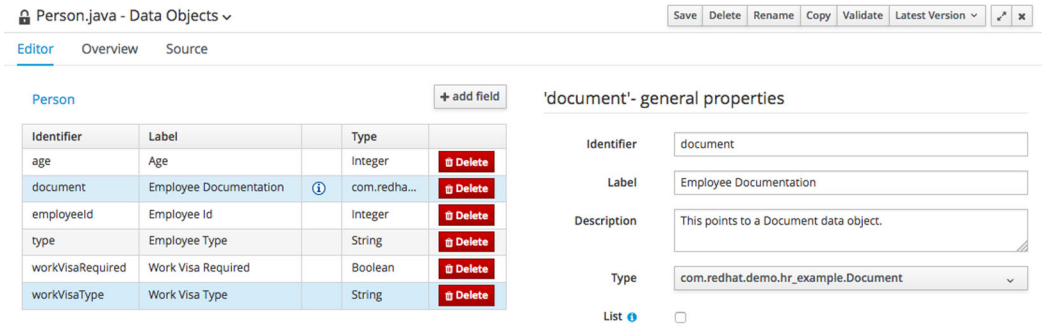


**Figure 5.7 The Person data object in the HR example.**

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 higer
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 5.8.

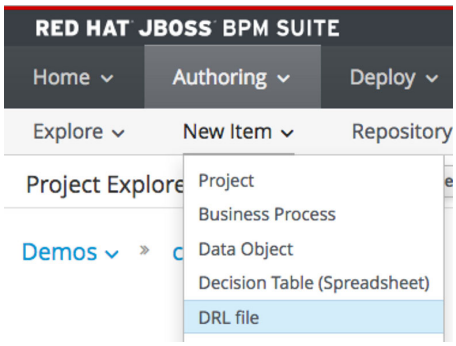


**Figure 5.8** **In your project, select the DRL file from New Item menu to create a new technical rule.**

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 5.9 and click on the OK button to open the rule in the DRL editor.

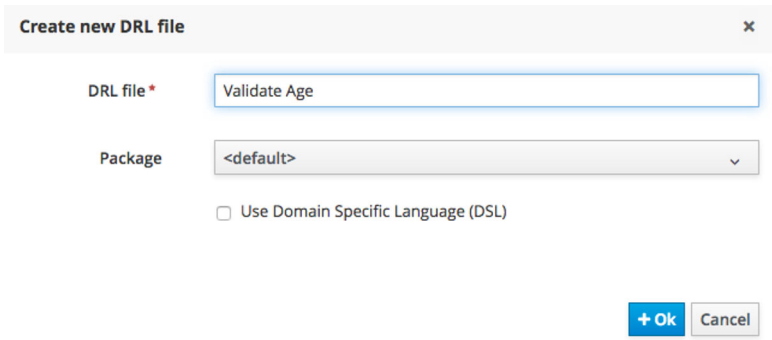


**Figure 5.9** **The pop-up where you create the new DRL rule file, providing the name *Validate Age* and package set to *<default>*.**

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 5.10.

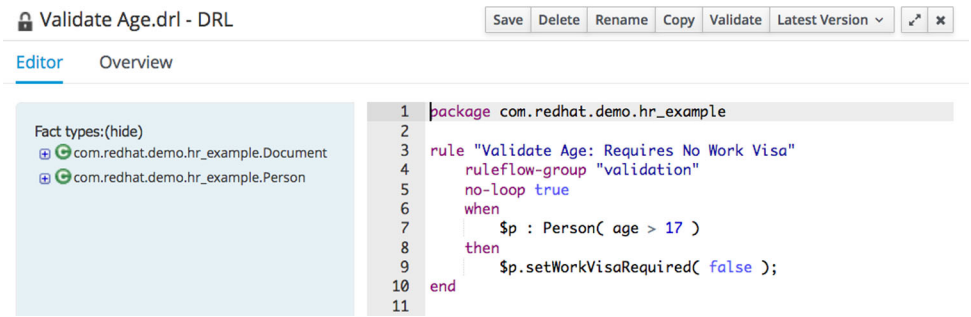


**Figure 5.10 The DRL file editor with the Validate Age rule for when the age is 18 or older, no work visa required.**

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.

### 5.2.2 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 5.11.

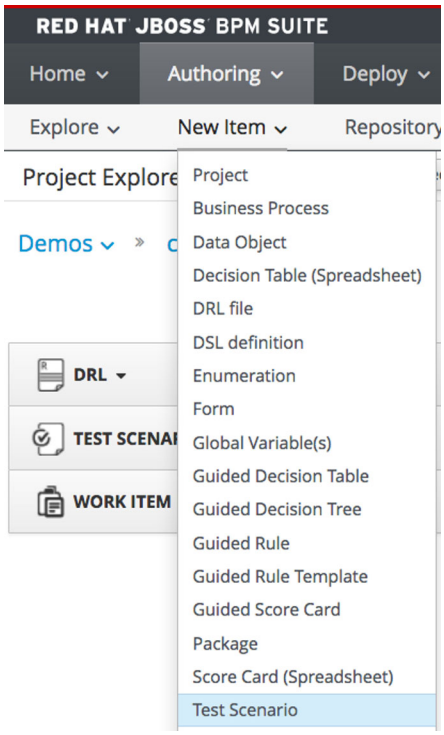


**Figure 5.11** **In your project, select the Test Scenario from New Item menu to create a new guided test.**

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 5.12 and click on the OK button to open the guided test scenario editor.


**Figure 5.12** **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 5.13.

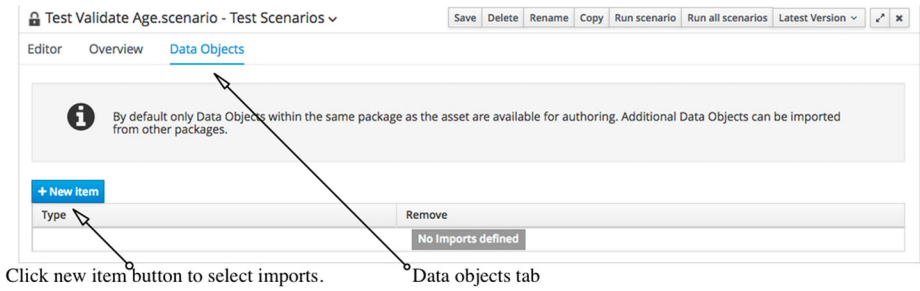


**Figure 5.13** **To import data objects from your model to use them in a test scenario, click on Data Objects tab, adding a new item places it in the list that was initially empty.**

Click on the *+New item* button. Select the person object from the *Import* menu in the *Add Import* pop-up as shown in figure 5.14.

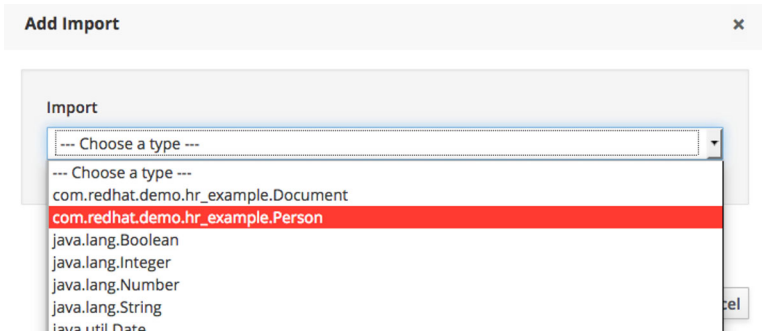


**Figure 5.14** **The add import pop-up contains a list of data objects available to you. Select the Person object for use in your test scenario.**

After clicking on the *OK* button, the Person object listed as an import's shown in figure 5.15. Now start defining your person fact by returning to the *Editor* by clicking on that tab.

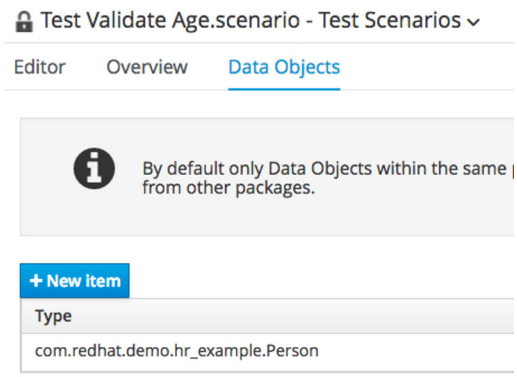


**Figure 5.15** **The Person data object's now listed as imported for use in this test scenario.**

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 5.16.

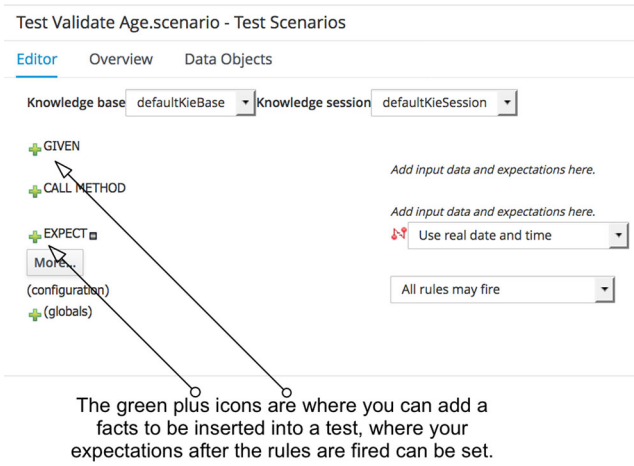


**Figure 5.16 The test scenario waiting for you to define facts to be inserted and the expectations after any specified rules you create have fired.**

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 5.17.

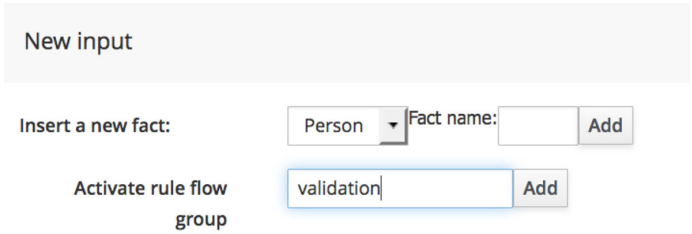


**Figure 5.17 The test needs to be given a rule flow group to provide you with a set of rules to apply your facts to, here *validation*.**

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 5.18.

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 5.19.

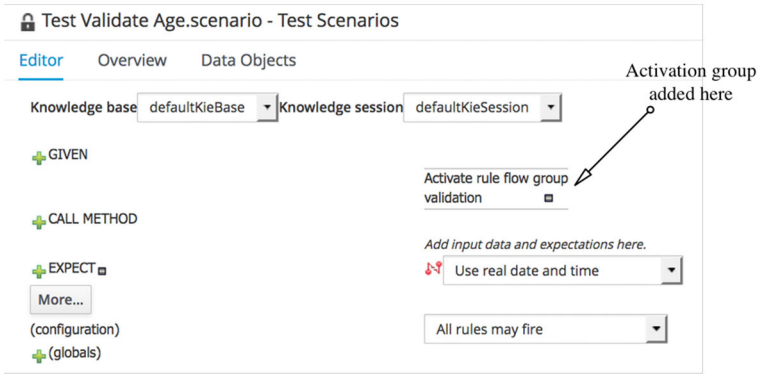


**Figure 5.18 The activation rule flow group's added to the test scenario in the *GIVEN* section.**

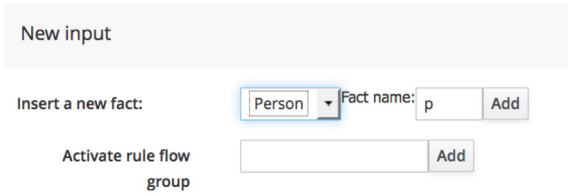


**Figure 5.19 Inserting a new Person fact with the name '*p*' to allow you to reference it later in the test scenario.**

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 5.20.

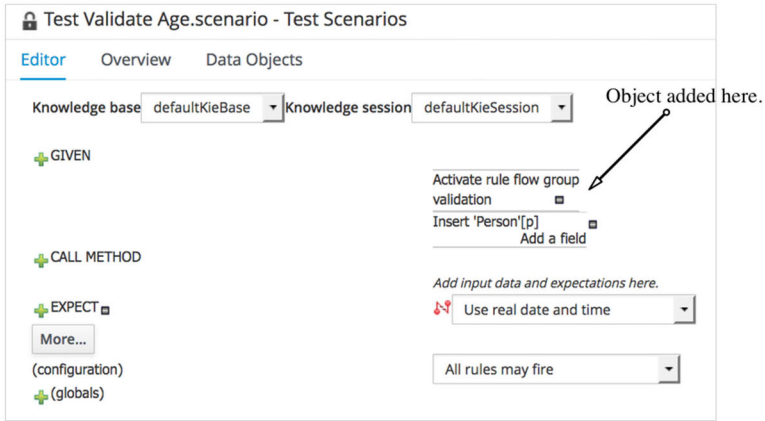


**Figure 5.20 The test scenario has the Person object named 'p' added as the fact to be inserted in this test.**

Now you add a field by clicking on the *Add a field* text, which displays a pop-up as shown in figure 5.21. You need to find the *age* field in the drop-down menu and click on the *OK* button.

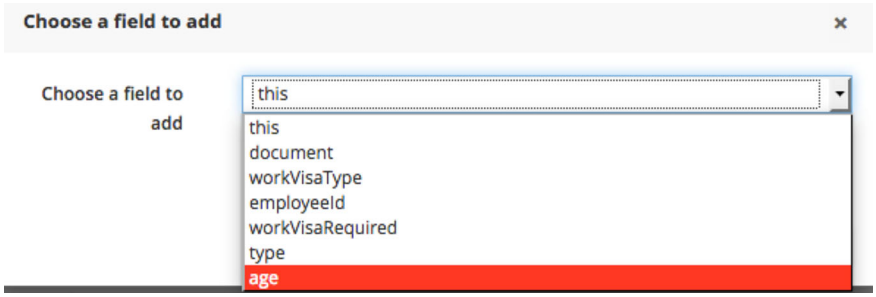


**Figure 5.21 When adding a field from the pop-up, select it from the drop-down menu.**

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 5.22 you see that after clicking on the pencil icon a pop-up appears to define that field.

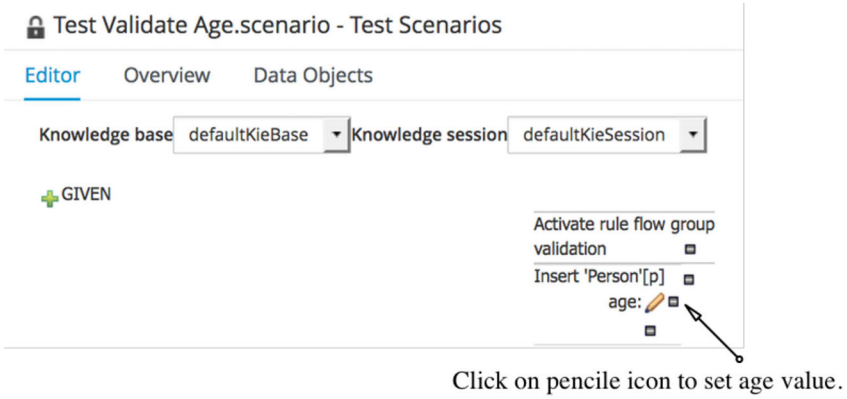


**Figure 5.22 The age field has been added, now you can click on the pencil icon next to it to define what's assigned to the field.**

In figure 5.23 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 5.24 with the facts defined and the rule flow group activation.

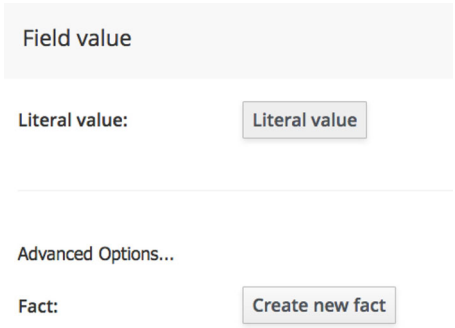


**Figure 5.23** To define the age field value as a number, you need to click on the *Literal value* button in the pop-up.

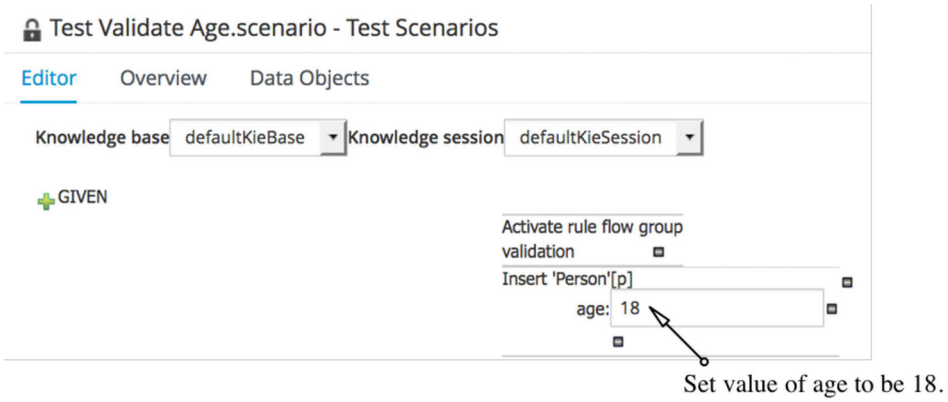


**Figure 5.24** The age field's give a text box for you to add the literal value of 18.

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 5.25 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

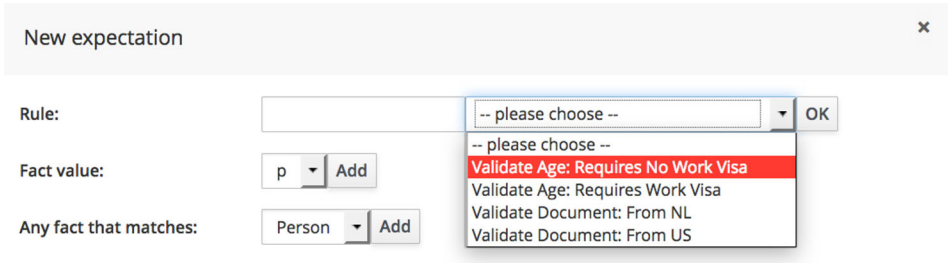


**Figure 5.25** **The new expectation pop-up where you can select the rule you expect to fire with the given facts in this test scenario.**

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 5.26 shows you the status of your test scenario.


**Figure 5.26** **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 5.27, 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.

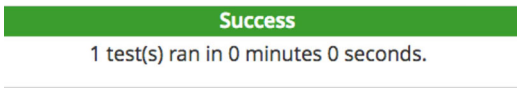


**Figure 5.27** **A successful test run appears in the reporting panel at the bottom of your screen.**

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 5.28 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 5.29.

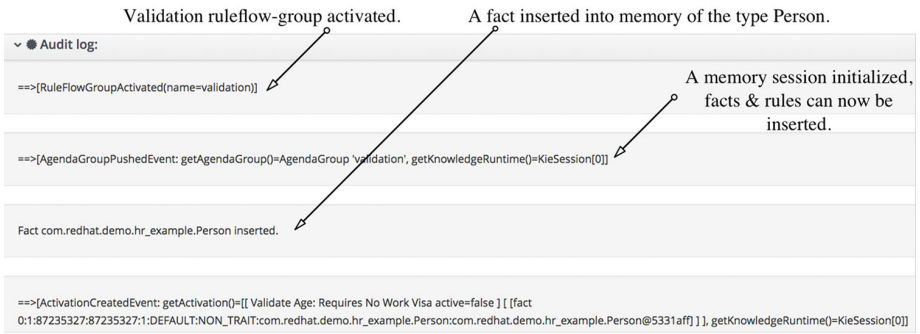


**Figure 5.28 The audit log for a test scenario run shows the rule flow group activation, session being setup, facts inserted, facts deleted, rule firing and more.**

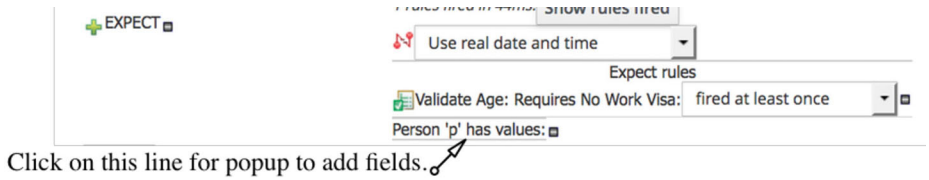


**Figure 5.29 The person 'p' has been added to the expectation, but still need to check if the field value for work visa required has been set properly by the rule.**

Click on the line *Person 'p' has values* for the pop-up to choose a field to add. In figure 5.30 you see how to select the field *workVisaRequired* from the drop-down menu.

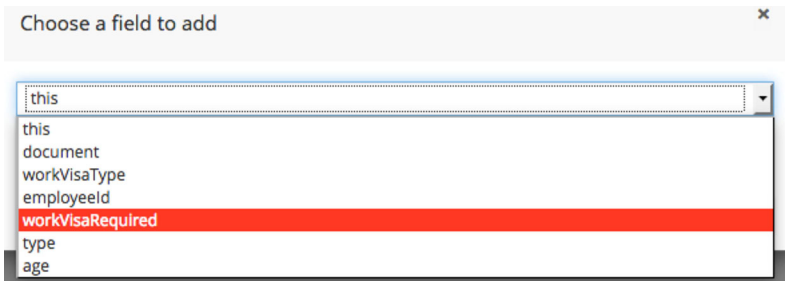


**Figure 5.30 The pop-up to choose a field to add's used now to select the *workVisaRequired* field.**

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 5.31 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 5.31** **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 5.27. 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.

### 5.2.3 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 5.32. 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 5.32** **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 5.33.

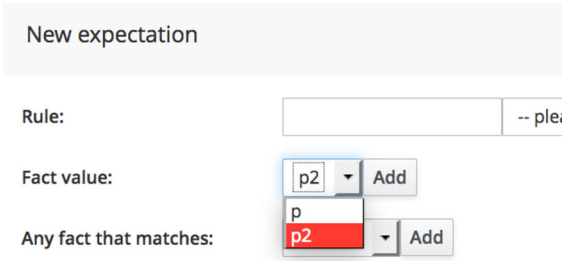


**Figure 5.33 Add new fact value p2 from drop down menu as our new expectation for the second person object.**

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 5.34 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.

### 5.2.4 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 5.1.1:

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

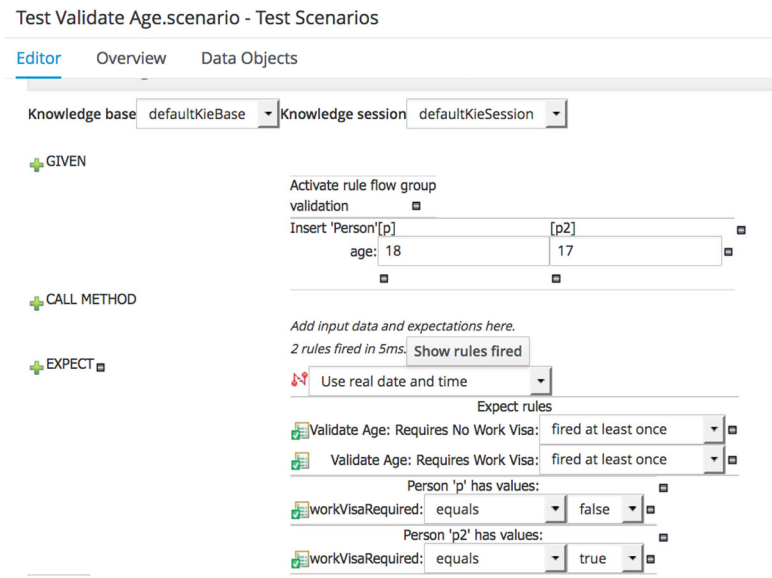


Figure 5.34 **Test validate age scenario completed and running successfully.**

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 5.35.

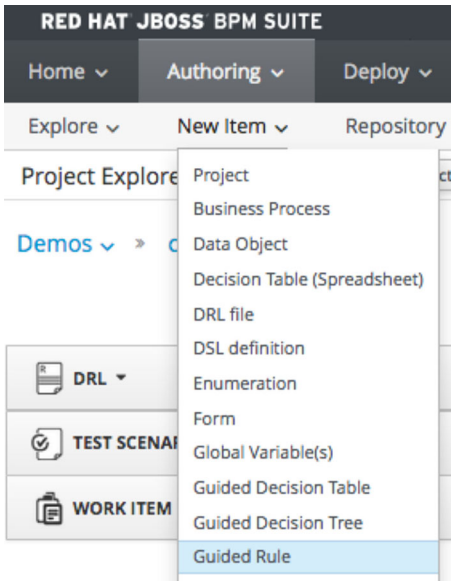


Figure 5.35 **Guided rule creation starts in the new item menu in project authoring perspective.**

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 5.36.

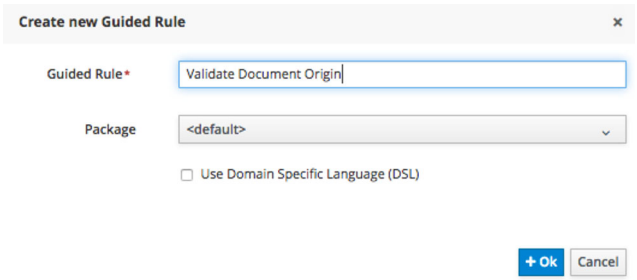


**Figure 5.36 The pop-up for creating a new guided rule.**

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 5.37 and then switch back to the guided rule editor by clicking on the *Editor* tab.

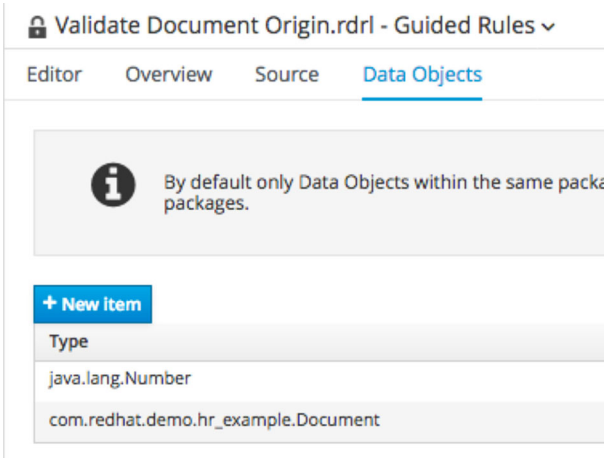


**Figure 5.37 Import a Document object using the new item button like you did in the test scenario.**

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 5.38.

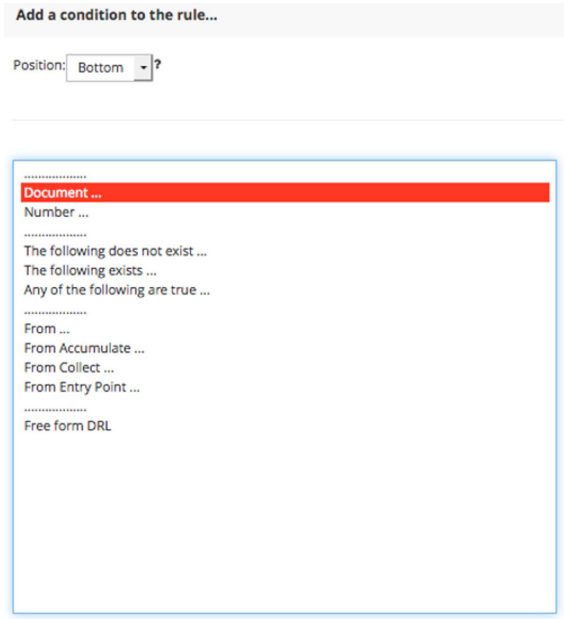


**Figure 5.38** **Add a condition to your rule by selecting the Document object and click on the OK button.**

The guided rule editor has now added a first condition line labeled *1*. 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 5.39 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.

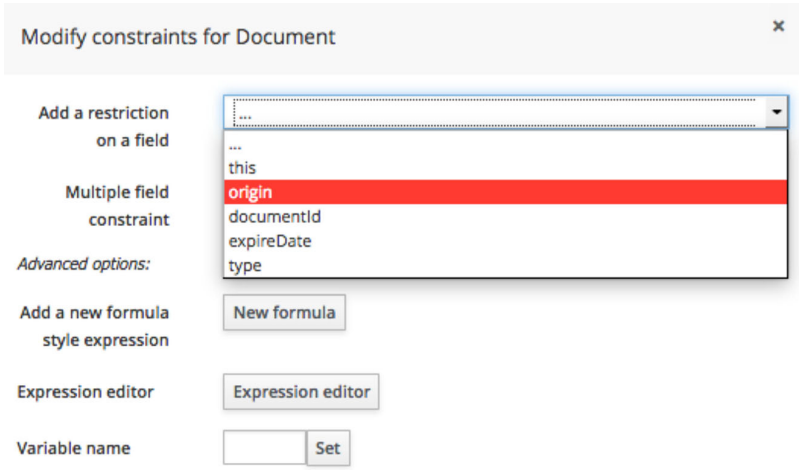


**Figure 5.39** **The pop-up that you can use to select the field you want to constrain. Selecting the origin field for your Document and the pop-up closes, leaving the origin field added to your conditions section of this rule.**

As shown in figure 5.40, 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.

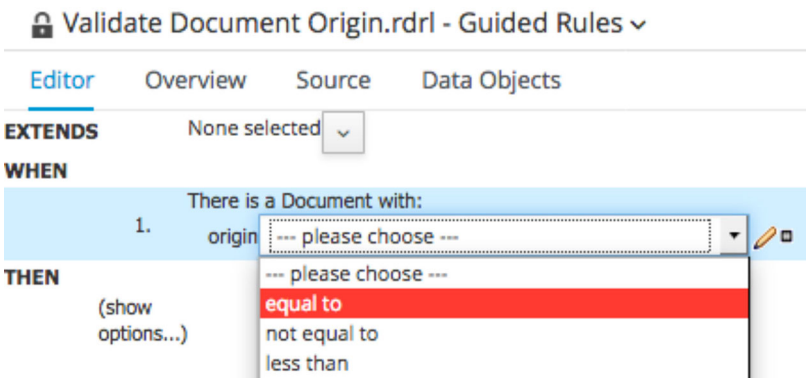


**Figure 5.40** **The document object's origin field now has a pull-down menu. Select as shown the equal to constraint.**

Once the constraint's selected, an empty text box appears for entering the value of *US* as shown in figure 5.41. Now to look for document objects with the origin field set to US.

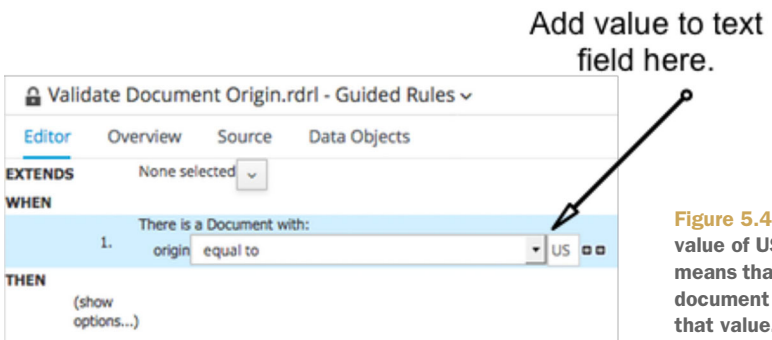


**Figure 5.41** **Adding the final value of US to the text box means that the origin field of any document needs to be equal to that value.**

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 5.42.

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

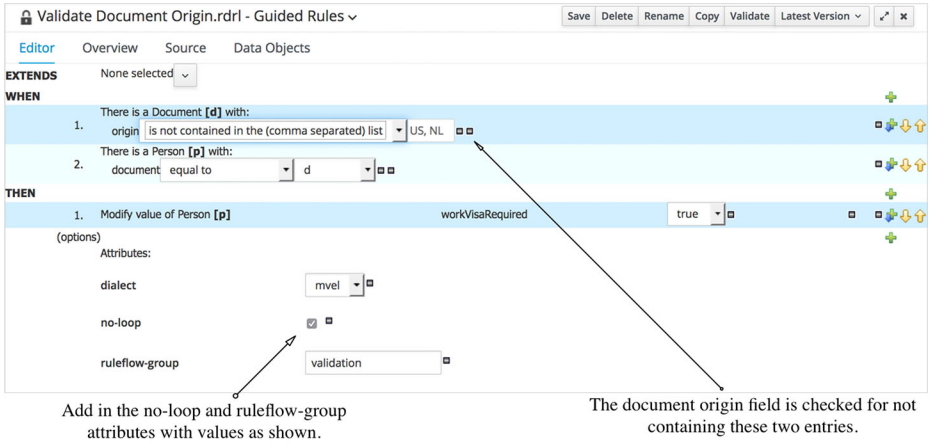


**Figure 5.42 The adjusted rule where you've constrained the document origin field excluding all values except what's in the comma separated list of US and NL. It's also important to set the options as shown.**

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 5.42.

### 5.2.5 Testing a guided rule

Nothing different from creating a test for a guided rule exists than the technical rule as shown in section 5.2.2. 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 5.43 and click on the *Run scenario* button to ensure you get a green test run report.

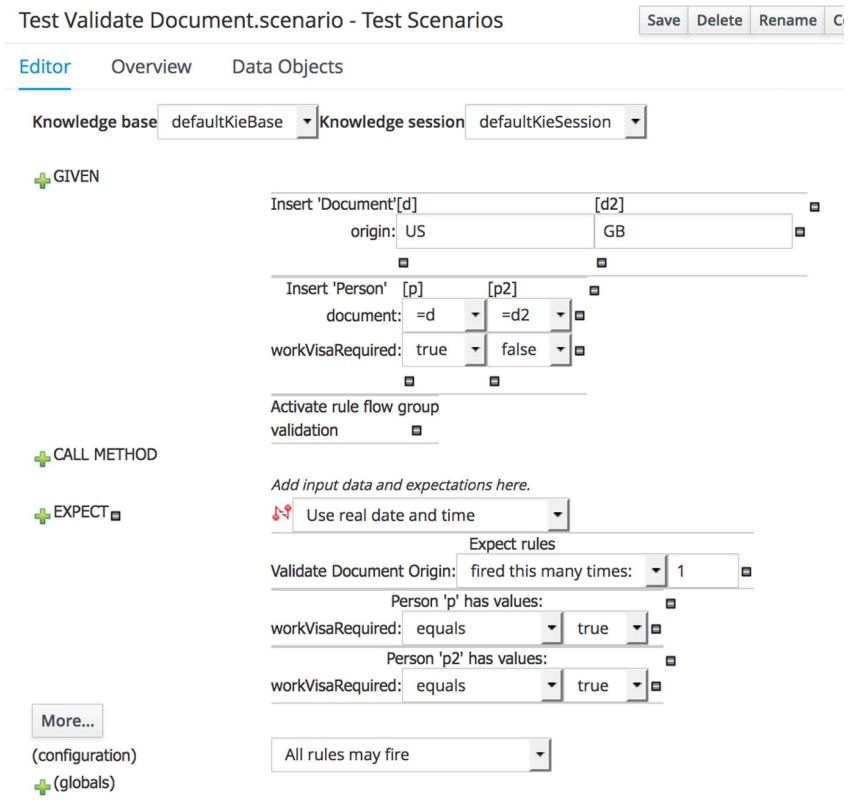


**Figure 5.43 The final test scenario to validate document origins. It includes two sets of facts which are tested for the positive inclusion of the origin in the expected set of data and the negative.**

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.

## 5.3 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.

# 6 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/effectivebpmwithjbossbpm/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 five section 5.1 for details. In Figure 6.1 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.

## 6.1 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."*[1] 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 programing 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.

[1] 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.

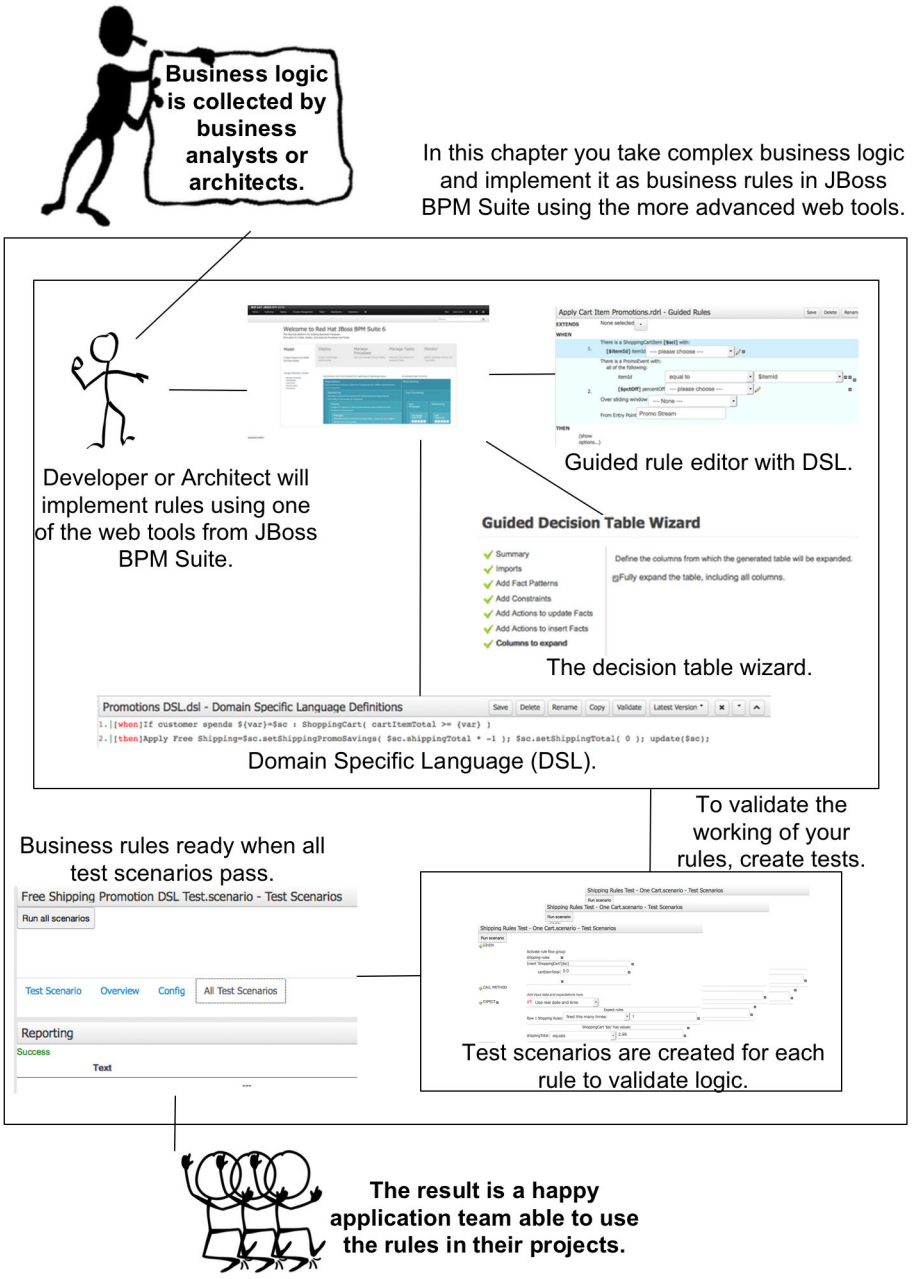


**Figure 6.1** **Business logic's provided to your project for designing rules. In this chapter, you design complex rules using a domain specific language (DSL), create a rule using the DSL and create a decision table.**

### 6.1.1 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[2] 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[3] 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,[4] but for now I dive right in and help you start creating a DSL.

### 6.1.2 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

---

[2] 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 five, 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/6.4/single/development-guide/#all_about_rules

[3] Drools Rule Language (DRL) has been discussed in chapter five section 5.2, refer to that if you need to refresh your knowledge.

[4] Discover all the details behind DSL syntax in the product documentation, https://access.redhat.com/documentation/en/red-hat-jboss-bpm-suite/6.4/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 6.2 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.

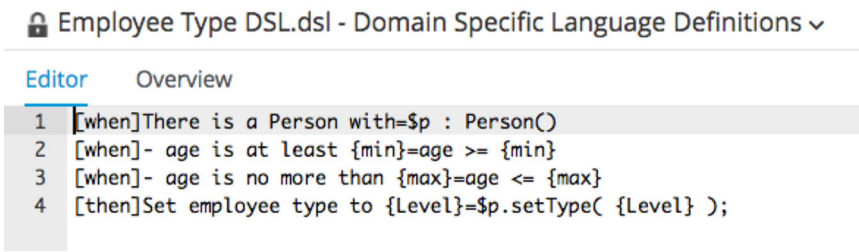


**Figure 6.2 The completed DSL, consisting of four lines of language to provide the natural language for rule developers to design the rules discussed.**

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 6.3.

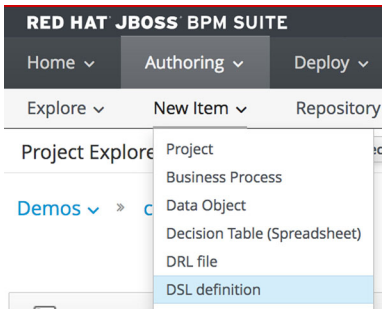


**Figure 6.3 Open a new DSL definition in the *New Item* menu from your project.**

The pop-up's presented and you give the DSL definition a name as shown in figure 6.4. 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.

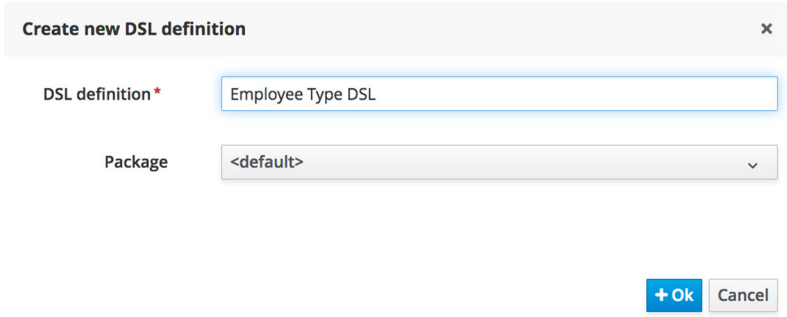


**Figure 6.4 The popup to create a new DSL definition shown here for you to enter the name *Employee Type DSL* and submit by clicking on the *OK* button.**

What you should notice first in figure 6.5 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.

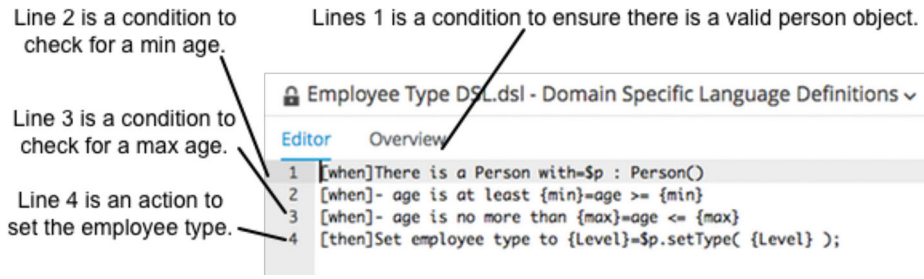


**Figure 6.5 This is the Employee Type DSL with each line annotated for further discussion in this section.**

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 6.5 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 6.5 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 6.5 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 6.5 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.

### 6.1.3 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 five, refer to section 5.2.4 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 6.6.

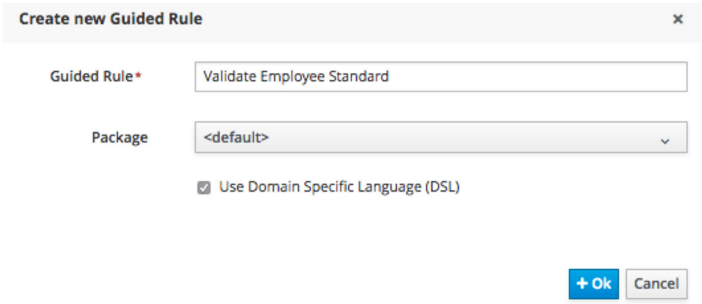


**Figure 6.6 The popup to creating a new guided rule's the same as used before, but now you check the box at the bottom to include a DSL.**

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 6.7 you see by clicking on the box at the bottom

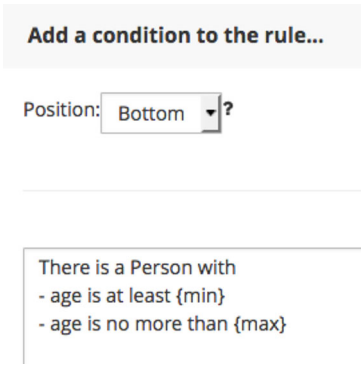


**Figure 6.7 The popup to add a condition now contains the three sentences from the DSL.**

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 6.8 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.

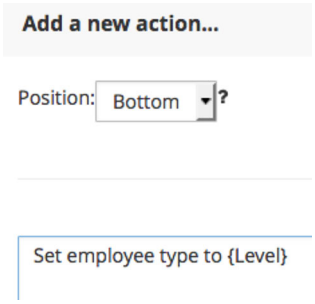


**Figure 6.8 The popup to add an action now contains the sentence from the DSL.**

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*[5] and *ruleflow-group*[6] set to *validation*. When you've completed the options, make sure it matches figure 6.9 before saving and moving on to create the second and final guided rule with DSL.

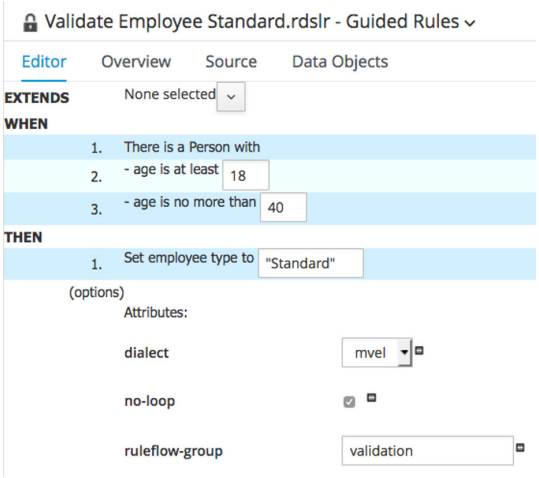


**Figure 6.9 The completed Validate Employee Standard guided rule using the DSL.**

[5] If you need to refresh your knowledge of what *no-loop* attribute's for, see chapter five section 5.2.1 where it was first introduced.

[6] If you need to refresh your knowledge of what *ruleflow-group* attribute's for, see chapter five section 5.2.1 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 6.10 before you validate and save it.

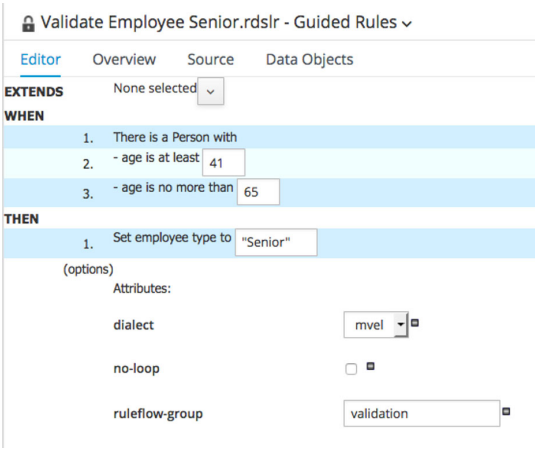


**Figure 6.10 The completed Validate Employee Senior guided rule using the DSL.**

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 6.11 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.

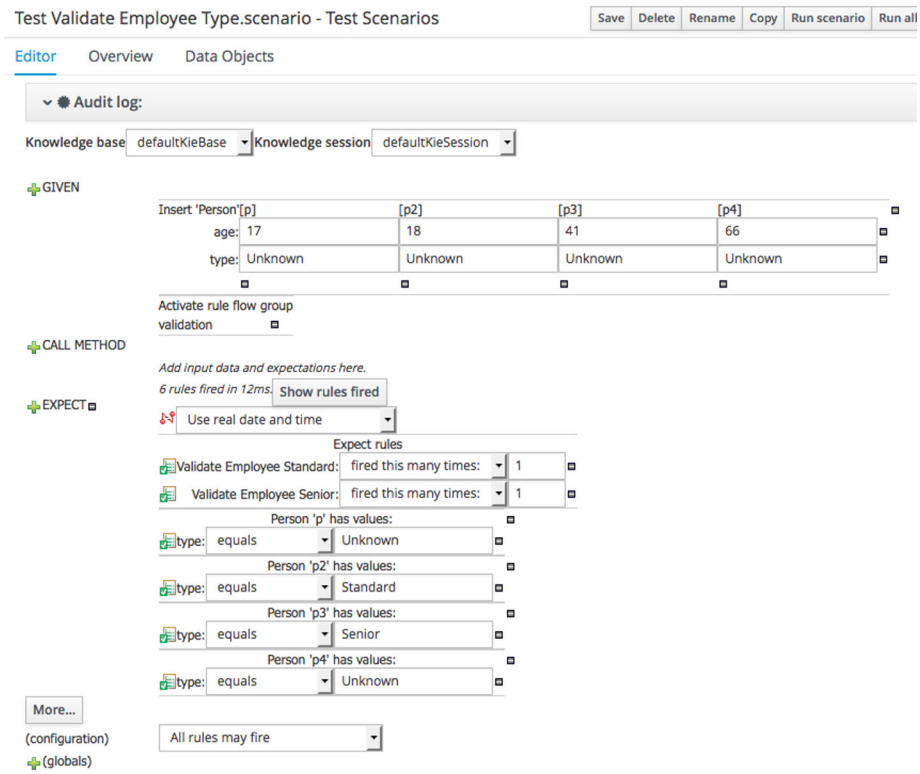


**Figure 6.11** **The completed test scenario which ensures all ages are validated for employee type.**

## 6.2 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.[7] 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.

### 6.2.1 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 6.12.

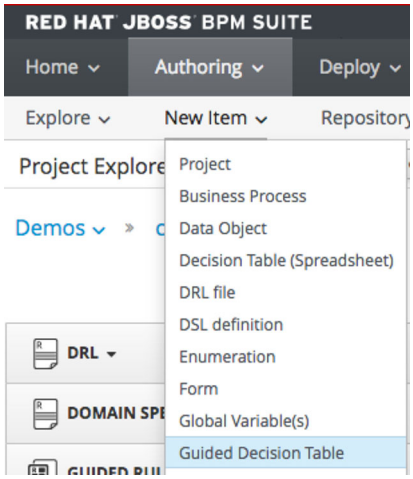


**Figure 6.12 Select the Guided Decision Table's created from the New Item menu.**

[7] 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 6.13.

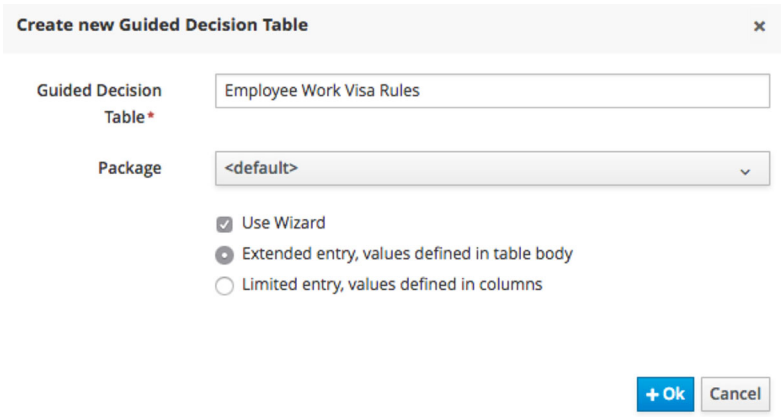


**Figure 6.13 Create a new guided decision table starts with this popup.**

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 6.14 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.

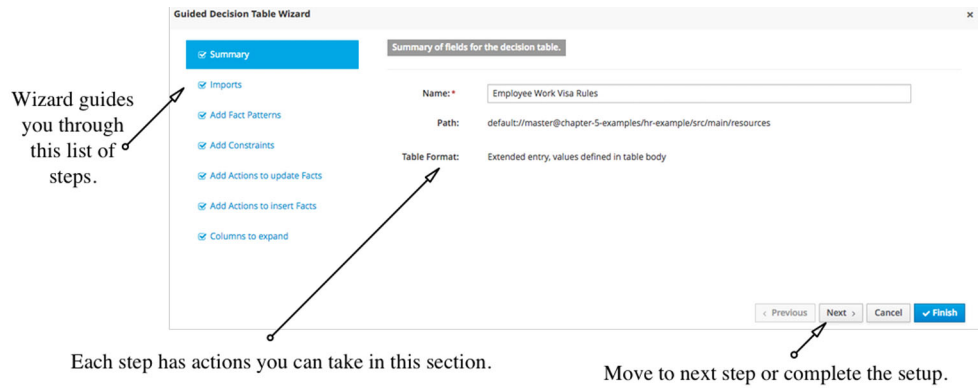


**Figure 6.14 The guided decision table wizard for step by step building of complex rule sets.**

Starting with the imports, shown in figure 6.15, 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.

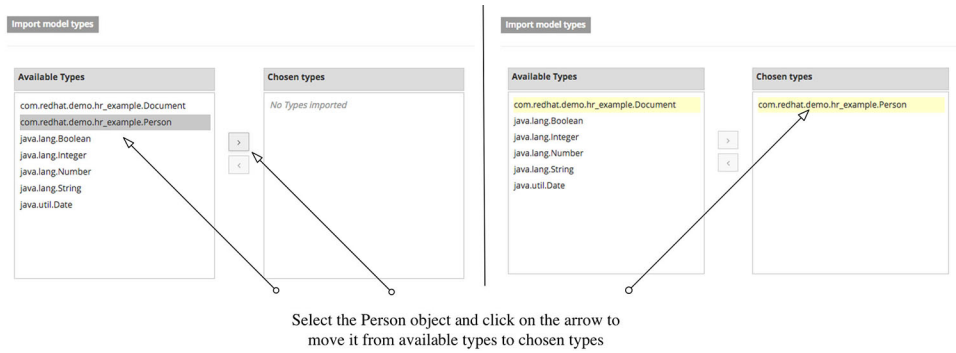


**Figure 6.15 Adding imports step in guided decision table wizard, where you select the data object to be imported and use the arrows to move to the right column to make it available for your decision table rules.**

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 6.16. 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.

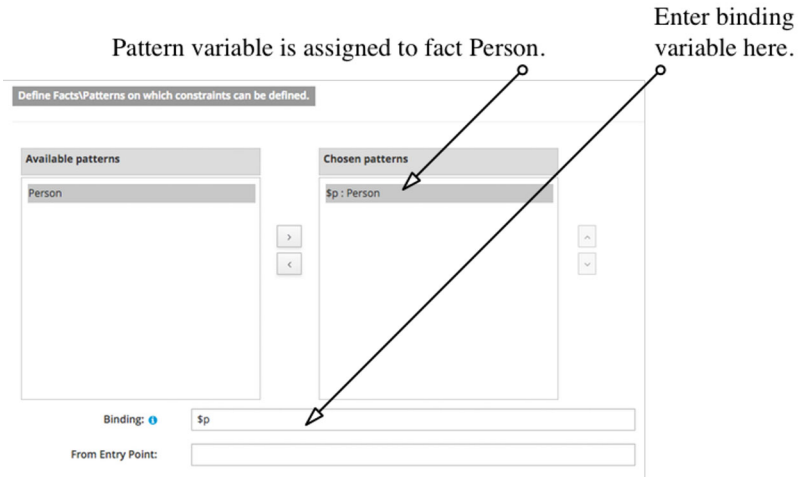


**Figure 6.16 Assigning a binding variable name to reference a fact.**

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 6.17, 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.

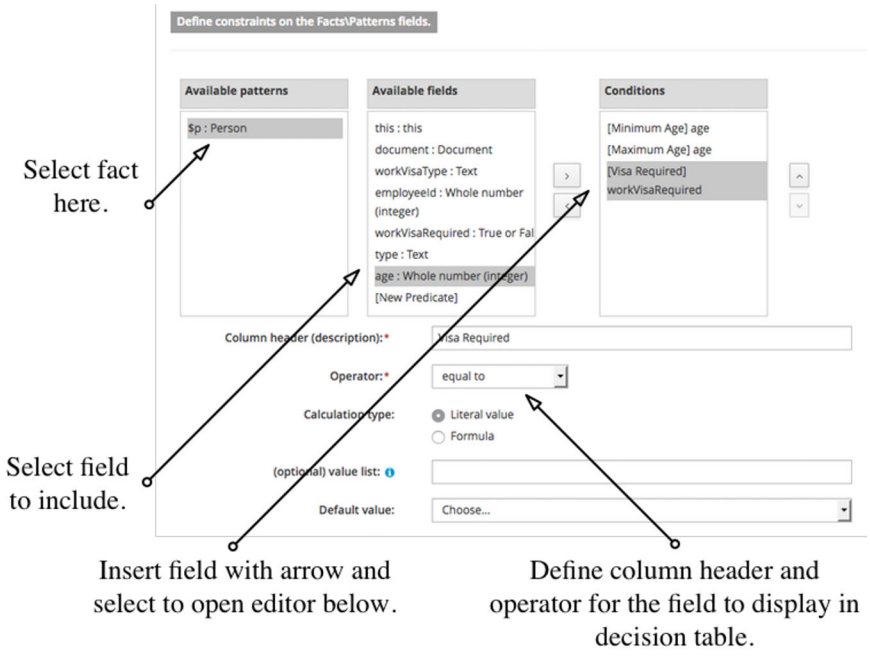


**Figure 6.17 Define constraints on the fact fields, these then appear in the final decision table to create rules.**

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 6.18, it's now red and you must select it to start filling in the column header, but leave the rest of the fields blank.

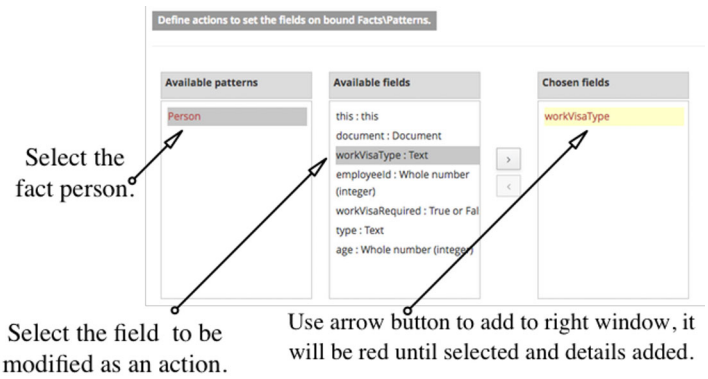


**Figure 6.18 Define actions by selecting the object, then the field you want the action to take place on. The chosen field, workVisaType, remains a red color until selected and the details are filled in.**

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.

### 6.2.2 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 6.2.1. Fill in the rows and your decision table should look like the one shown in figure 6.19. 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 *ruleflow-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 6.20 and click on *OK* button.

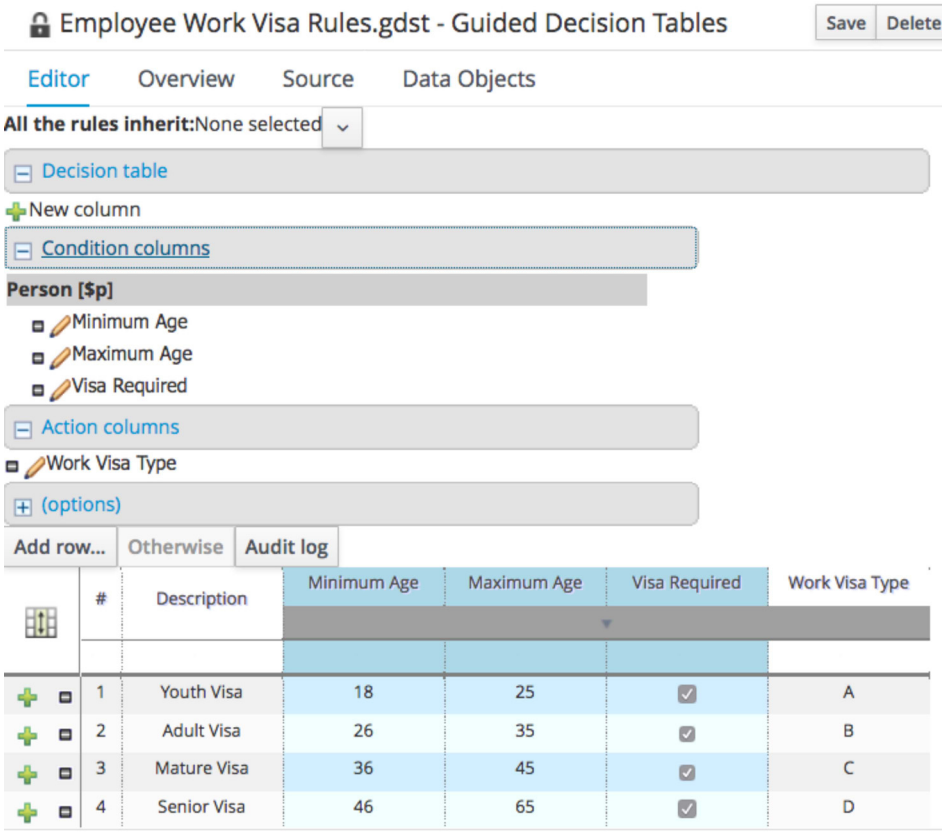


**Figure 6.19** **The decision table with the four complex rules entered as rows, defining what age span requires which visa type.**

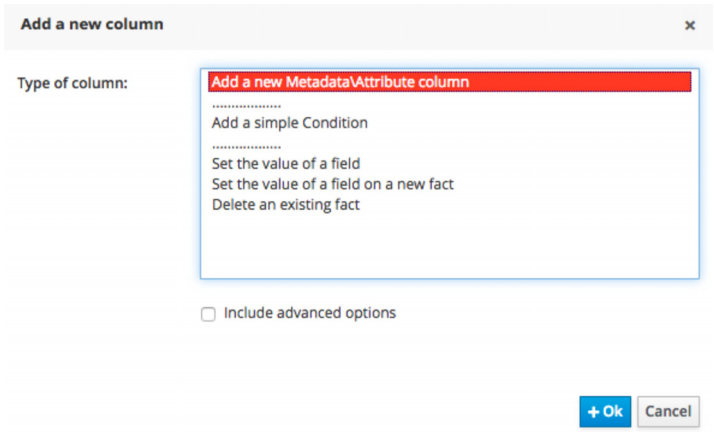


**Figure 6.20** **Add a new column popup's used to add options to your guided decision table.**

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 6.21.

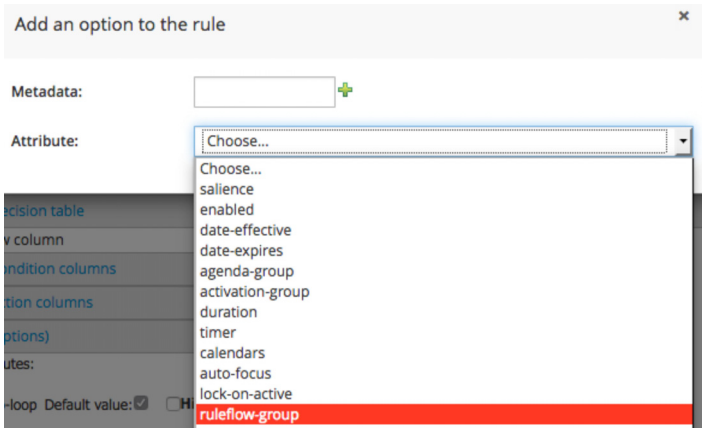


**Figure 6.21** **Add attributes to your decision table from the drop-down menu, such as ruleflow-group.**

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 6.22.

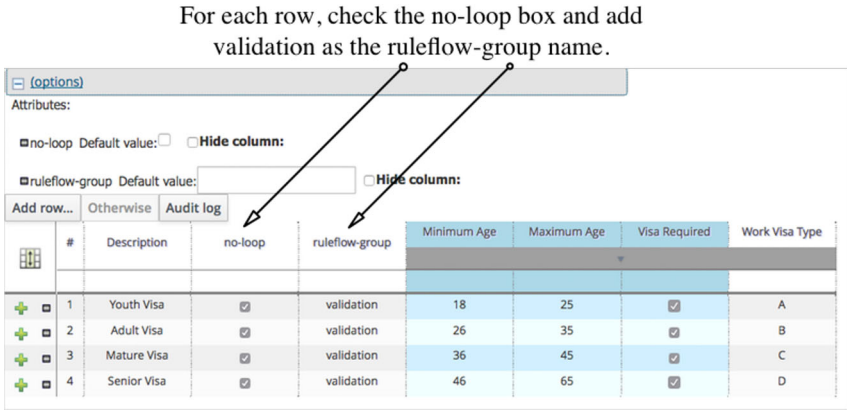


**Figure 6.22** **There attribute values should be filled in for each row and for each rule in the decision table.**

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 6.23.

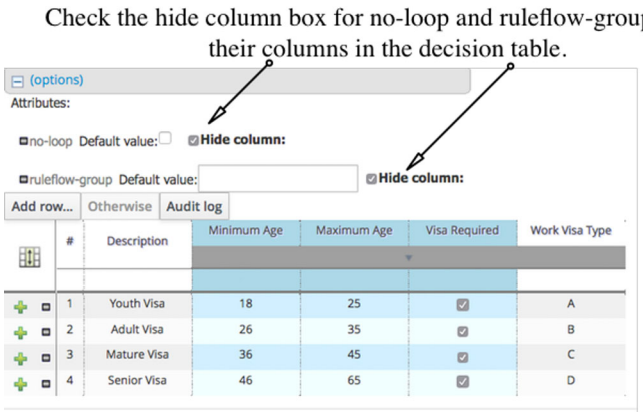


**Figure 6.23 To enhance readability of the decision table, hide the attribute columns.**

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.

## 6.3 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.

# 7 Designing business processes

***This chapter covers***

- Understanding basic usage of JBoss BPM process design tooling
- Implementing a business process with JBoss BPM
- Using an iterative process to implement a business process
- Validating the implemented business processes
- Exploring advanced business process design concepts

When browsing books with a title that includes the words "business process management" or "BPM," you expect a chapter to be devoted to designing and implementing processes. This book laid the groundwork needed to start implementing processes. You've created data models and business logic, and the next step's to capture the process using the data model, tie in the business logic, and deliver on the promise of business value to your customers.

The goal in this chapter isn't to show every task possible for implementing a business process using JBoss BPM Suite process designer. I'm teaching an iterative approach for using the process designer to implement an example process in this chapter which introduces all the basic tasks. By showing you how to implement the next task in a process, then validating the results before continuing onwards, builds confidence in using the process designer. This iterative approach breaks up your learning into well-defined steps, each resulting in a new working process which is a step closer to completion. By the end of this chapter, delivering more advanced

process tasks and functionality's a matter of using the skills taught and a bit of exploration within the process designer tooling. Business process design and implementation's an ongoing learning experience, and expect to continually learn new ways to effectively implement business process solutions.

Although all JBoss BPM Suite processes are implemented using the BPMN standard, this chapter isn't a complete tour of BPMN process elements. Working with the most common elements when building business processes ensures the building blocks expand your processes step-by-step into an eventual completed solution. Providing these basic tools to get you started are meant to trigger a desire to explore more advanced BPMN elements on your own.

Your journey starts in the domain of a Human Resources (HR) onboarding process, the example process which you're implementing in this chapter. Onboarding's the concept of processing a new employee in a company; registering in payroll systems, granting them access to resources, and more. In chapters four through six you can read about designing domain models, rules and test scenarios that all support this story of onboarding new employees in your company. Figure 7.1 shows an overview of a single iteration of activities needed to create a step in this example process. Adding a new step at a time allows for verification of the progress and ensures the process is working as expected before proceeding.

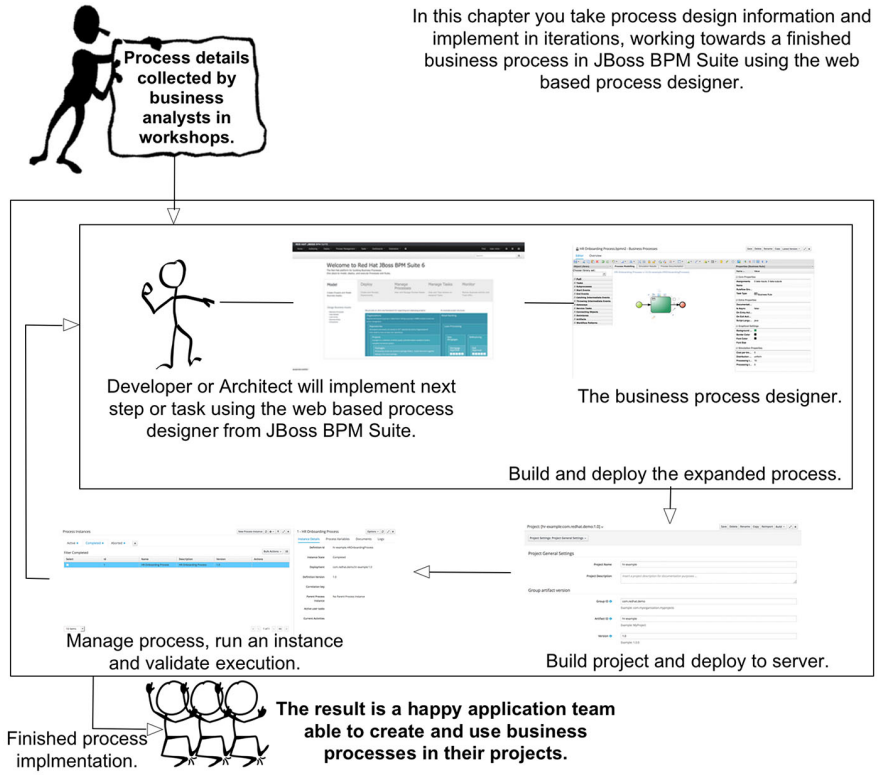


**Figure 7.1 The overview of the iterative path taken to develop, deploy and validate a step in your business process implementation.**

Let's dive right in and start putting together the first business process as you begin to tackle this HR on boarding project.

## 7.1 Meet the business process designer

In this section, you learn to navigate the JBoss BPM Suite menus to create a new business process artifact, see where you can find it in your project, and see how to open it in the process designer.

To get started, an example project's provided for this chapter containing the artifacts and business process created in this chapter. The example project uses rules and test scenarios previously constructed in chapters five and six. It's unnecessary to have previously completed those chapters, but helps in understanding how rules and tests work. This section uses the provided rules and tests without discussing their details.

Install the provided code project (https://github.com/effectivebpmwithjbossbpm/chapter-7-business-process-demo) for this chapter before continuing. If you need to review setting up any of the example projects as local installs or as containerized projects, please refer to chapter three. Once the project's installed, start the JBoss BPM server as listed in the instructions.

To get started implementing a process, a new process must be created. This work starts in the Project Authoring aspect, which you can find in the Authoring menu in the home screen as shown in figure 7.2.

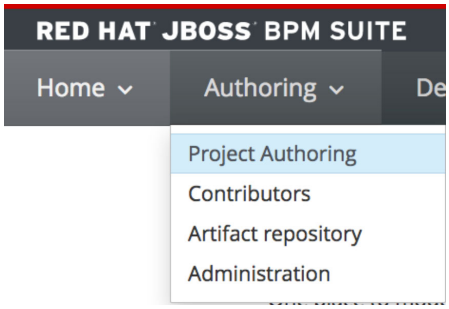


**Figure 7.2** From the Authoring menu, select Project Authoring to open the project example prepared for this chapter.

Once in the Project Authoring perspective, create a new business process from the *New Item* menu as shown in figure 7.3. This opens a pop-up asks the details needed to create a new business process.

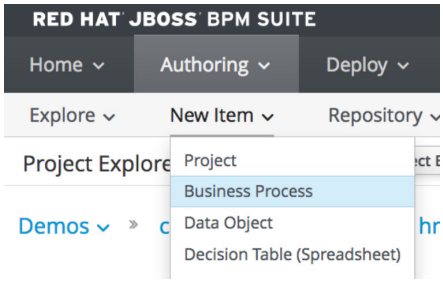


**Figure 7.3** Create a new business process from the New Item menu.

Fill in the pop-up menu with the name of the business process being *HR Onboarding Process* and leave it in the *<default>* package for this project. Figure 7.4 shows you the filled in details of a new business process before clicking on the *OK* button to create it.

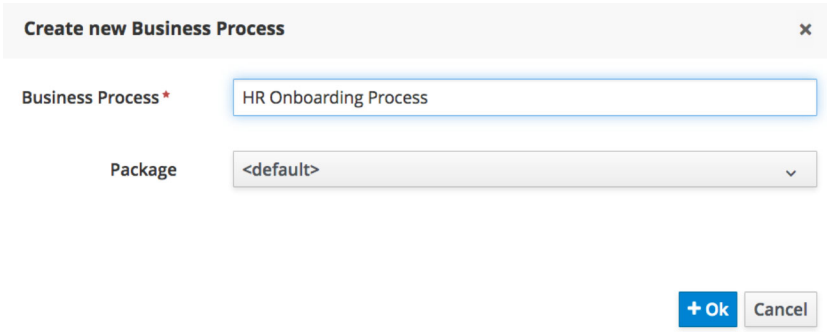


**Figure 7.4 After filling in the name and package level for the new business process, submit the pop-up by clicking on the OK button.**

This creates the business process file in the *Project Explorer* pane on the right under the menu *Business Processes* and opens it in the process designer. Figure 7.5 shows the process designer with the HR Onboarding Process open. Like all new business processes, the designer opens with a single start node on the screen and awaits further instructions from you.

With the process created, let's get started with the process designer by taking a first pass at this business process.

## 7.2 Get started with the process designer

The next step on your journey's to use the process designer and incrementally expand the process from its initial state to a completed process. This isn't something you want to do in a single try, as the complexities involved are easier to test if you take this step-by-step.

In this section, you take a first pass at the process by adding a single step and then end the process. Learning the basics of working with the process designer allows you to successfully build and verify your process design as you work towards a final solution.

This section teaches you how to do all the following tasks:

- Navigate the process designer
- Design a first working process
- How to add and manage debug helper statements in your process
- Activate the auto save features to protect your work in the process designer
- Validate the process by checking for errors in message pane
- Build and deploy the process

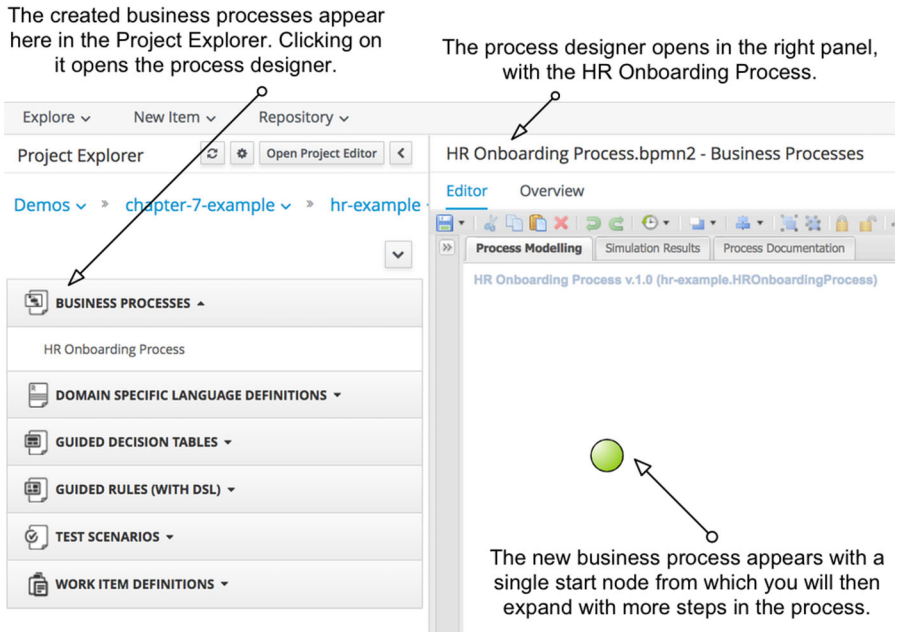


**Figure 7.5 The process designer's open showing the start of the HR Onboarding Process.**

- Verify that the process has been deployed correctly in the process management tooling
- Start an instance of the deployed process
- Inspect the process instance for completion in the process management tooling
- Verify that the script task node produced the expected welcome message in the server log

It's important to work through each of these steps to ensure a successful process implementation. Let's kick this off by expanding the initial process by adding a new step to the process.

### 7.2.1 Extending the process

A process has steps and each step's represented in the diagram by a node type. If you need to review concepts such as process steps and node types, please refer to chapter one. Currently you've a process with a single node, the start node. To start extending this process and make it runnable requires, at the least, one node of any type and an end node.

Start adding task nodes by using the main process palette. You select the start node by clicking on it to activate the mini-icons that act as speed options to building your process. The next step from this start node's added by selecting any of the icons shown in figure 7.6.

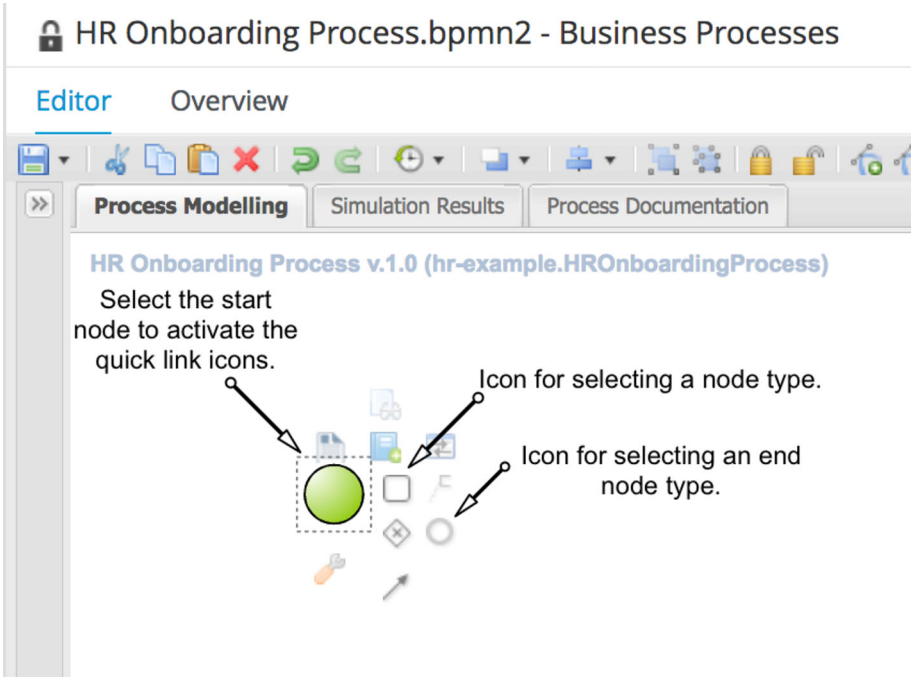


**Figure 7.6** **Select the start node. Mini-icons appear that are short-cuts to building out a process.**

If you click on the node type icon (small box) it adds a blank node with a transition (directed arrow) between the start node and this new node. The new node's automatically selected, which allows you to see the mini-icons to select your next process step you require. Select the end node mini-icon (double circle) and the process is extended with a transition to an end node. Figure 7.7 shows the process in its current state with the diagram completed, but nothing named and no functionality defined for the nodes you've added. Right now, they're visual placeholders for functionality or actions to be taken in that step of the process.

A good habit to get into is to turn on the automatic save option. It's found in the process designer on the top left of the menu bar, shown in figure 7.8. Click on *Enable autosave* to activate this and worry no more about saving your work.

Now it's time to put some actions in the node you created and label the nodes for documentation clarity when anyone reviews the process diagram.

### 7.2.2 Adding functionality to nodes

You've extended the initial process and now you can specify the node type. By giving them names and adding other specific details, these node types can deliver desired functionality.

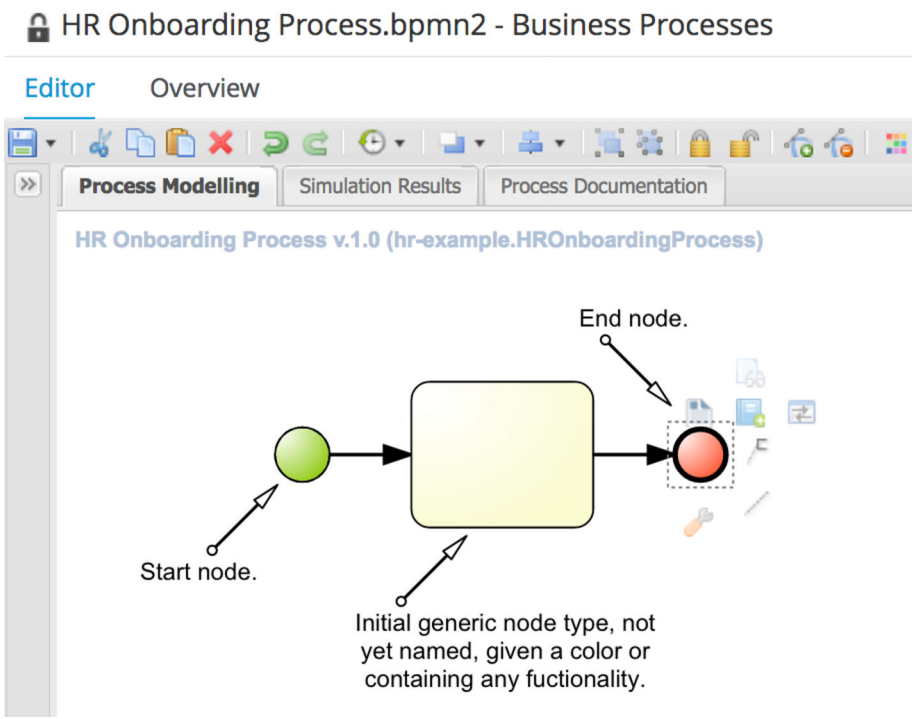


**Figure 7.7** The first pass you've extended the process to contain a start node, a yet to be defined generic node and an end node.

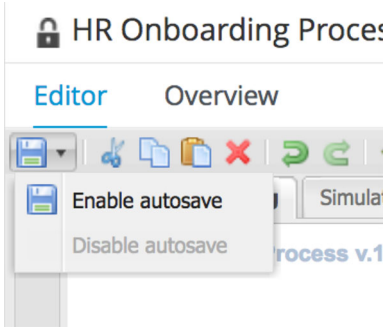


**Figure 7.8** The menu under the disc icon enables the autosave feature.

Starting with the first node, the start node, you might notice they're green by default and named if you desire. Let's leave the start nodes in their original state in the designer and move onto the next node.

This node isn't yet defined as a specific type, but you can see what options a node type has by clicking on the node to activate the mini-icons and then on the wrench icon, which is used to configure a node, as shown in figure 7.9.

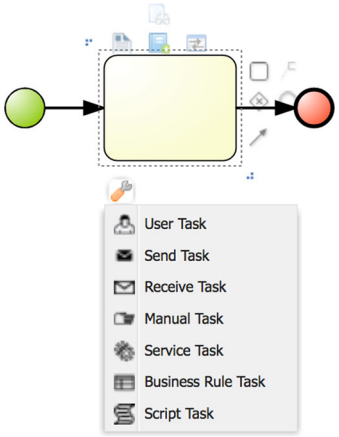


**Figure 7.9 Click on the wrench icon to see the list of node types available for you to configure this node.**

To get this process started, select the *Script Task* option and you can add some print statements to the script as an indication that this node was executed. Any print statements in a script task produce entries in the JBoss BPM Suite:server log by default. After selecting this node to be a script task, the small icon next to that entry's added to the node to allow you to identify what node type it is.

Next up's giving this node a name, adding a color to identify it, and adding the functionality to print a statement to the logs. This can all be done by selecting the node and opening the *Properties* pane on the right by clicking on the double arrow to slide it out.

Figure 7.10 shows the *Properties* pane open for the node with the details filled in as follows:

- Name: Hello Reader
- Script: This field shows partially the functionality you add next, using the pop-up *Expression editor* to send a message to the logs.
- Graphical Settings -> Background: Select color green from pop-up options when you click on this field.

You can add the name and choose the color for the node type, but to fill in the expression you need requires Java statements. These get added by clicking on the *Script* field to activate the pop-up *Expression editor*, which you can then add the statements as shown in figure 7.11. These print statements, some line feeds to help format the end results and write to the JBoss BPM Suite server log. This provides a means of validation that this node's been reached.

Developers have low-level debugging tools to help them, but at the process level you can use some tips to help you gather more information as you move through a process and can discover where it isn't working as expected.

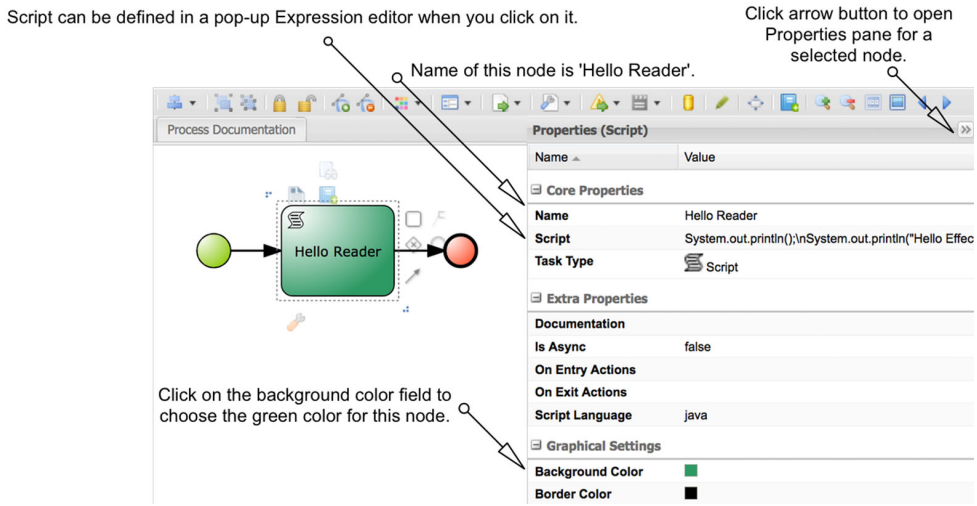


**Figure 7.10** **The Properties pane opens for the selected node when you double-click the arrow button on the top right. The attributes for this node are set to ensure a name, functionality in the script and a color.**

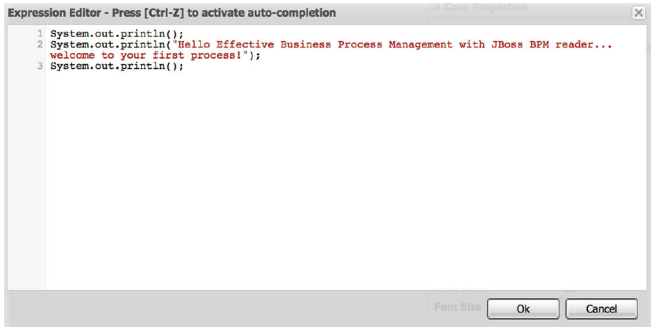


**Figure 7.11** **The Expression editor's used to add Java language statements in this node for printing text to the JBoss BPM Suite server log.**

### SIMPLE PROCESS LEVEL DEBUGGING TIPS

In a process, you can't set a debug break point like you can in a piece of code. The JBoss BPM Suite provides you with the visual tools shown in this chapter for post run-time investigations, but this only shows the path taken. Other helpers allow you to see process variables at the end of a process instance and inspect their state information. You see how to do this later in this chapter.

One of the easiest tips to apply's adding print statements to task nodes. Some tasks have a property called *On Entry Actions* and *On Exit Actions*, which give you the freedom to add Java language commands to print to the JBoss BPM server log. Not all task

nodes use these properties, for example, a script task executes only the script code from the *Script* property field.

When you put them in the *On Entry Actions* field they're the first things executed when a node in a process is entered. The *On Exit Actions* are the last things executed before the node's finished and the process moves on to the next step. A simple print statement to notify you of your location in the process is often handy to isolate problems when they occur.

As mentioned, in your script task node you can add the extra lines as shown in figure 7.12 to first print the name of the node being entered and then again at the end of the script to denote leaving the node. You learn about the *kcontext* API variables and methods later in this chapter, for now understand that this looks up the current task node name.

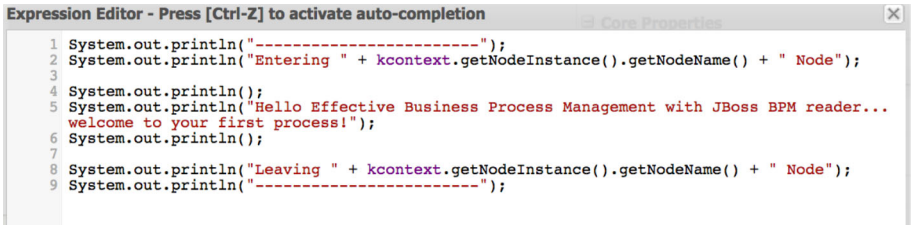


**Figure 7.12 The expanded script task node *Script* field with enter and exit helper statements.**

Getting back to your process, the final node's an end node that you can select, then open the *Properties* pane and update with the name *End Successful Onboarding*. Another way to give a task node a name's to double-click on the node, which opens a text field for you to enter the name. It's always a good practice to get in the habit of naming end nodes, as you often have processes with various end states. They can end on a desired path or in one of several error states. You want the process to end and reports to be generated that detail how many processes have ended and in what state become much easier to formulate when you've every single end node labeled with a clear name.

Figure 7.13 shows this first iteration contains a valid process, but to be sure you can check the messages pane at the bottom of the screen.

Because no obvious errors or issues exist, let's validate this extension by building, deploying and running it. This verifies that the process works as expected.

### 7.2.3 Validate your process extensions

Validating your process is achieved by building the project using the *Project Editor* and executing it. Click on the *Open Project Editor* button found on the left side in the Project Explorer panel. Select the menu item in the top right labeled *Build* followed by *Build & Deploy* to start a deployment of the project. If all goes well, you'll see a green bar pop-up on the top of the screen that says *Build successful*. To verify that it's indeed

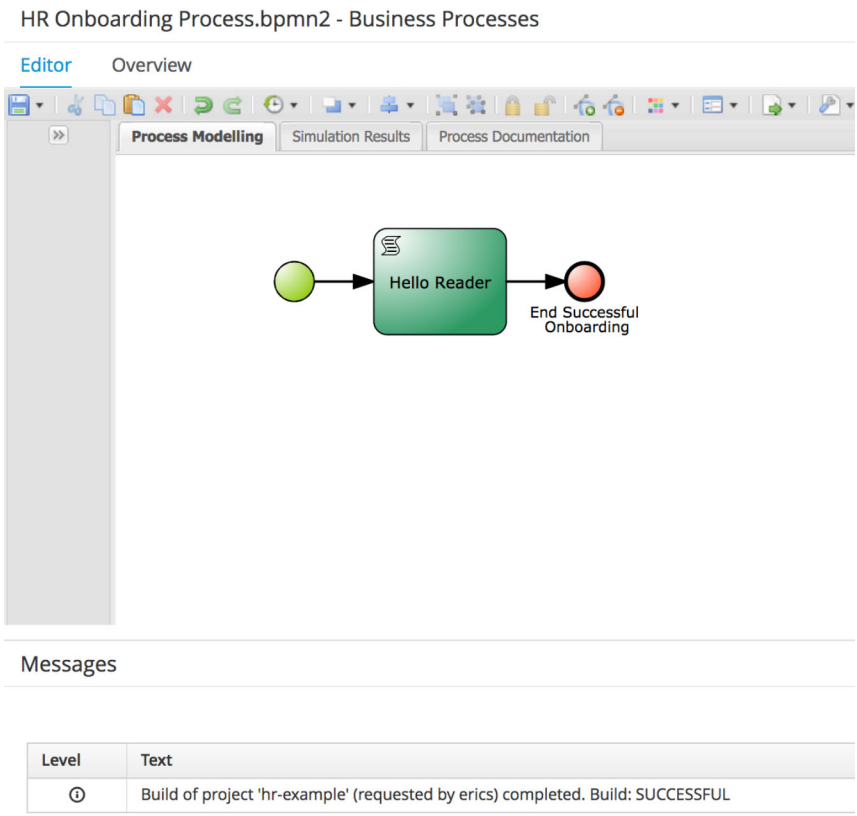


**Figure 7.13 The messages pane at the bottom of the process designer can contain errors and issues if any are found during your process design activities; keep an eye on this.**

deployed, click on the top menu item named *Process Management* and then *Process Definitions*. Here you make sure that the process definition's listed as show in figure 7.14.


**Figure 7.14 When you deploy the project, you've a process definition ready be used.**

Looking closely, notice a *Start* button located next to the process definition. If you click on this button, it starts a process instance of the HR Onboarding process. A pop-up

appears which is normally a process start form that you design to accept process data based on what your process might need. In this case, it's an auto generated empty form as shown in figure 7.15 that you need to submit via the button of the same name.


**Figure 7.15 The auto generated start process form's empty and needs to be submitted to trigger a new process instance.**

For now, it's sufficient to use the auto generated empty form to start your process. In chapter eight you'll learn more about form use and how to design a start process form for your processes.

You see a pop-up green bar that states the process was started successfully and shows its process id. It's an identifier unique to that process instance that you can look up in the process instances view. Click on the top menu item named *Process Management* and then *Process Instances* to see the list of process instances. Figure 7.16 shows the view of the *Completed* tab of process instances which are finished.

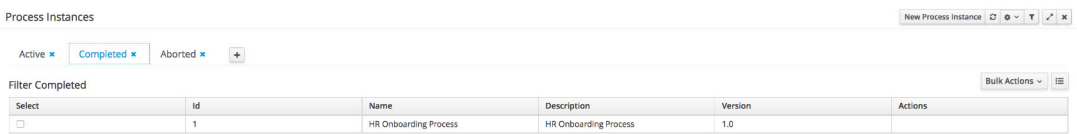


**Figure 7.16 The HR Onboarding process has completed instance with id 1.**

You can examine the details of this process instance by clicking on it to open the more detailed view. A pane opens on the right that contains more process instance details and other tabs as shown in figure 7.17, but you need a visual representation of the process instance to verify that it completed.

Select the drop-down menu *Options* as shown and click on *Process Model* to open the process model viewer. You see in figure 7.18 the exact same model as the one in the *Process Designer*, except it shows the nodes greyed out as they've executed.

This is how you can verify the path taken through a process. If the process had more paths not taken, they'd have their original node colors (not greyed out).

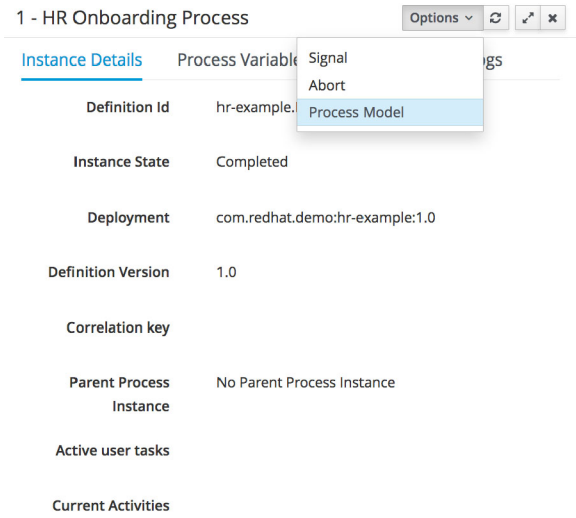


**Figure 7.17** The process instance details are shown with the *Options* menu open, allowing you to select the *Process Model* to visually verify what the process status is.

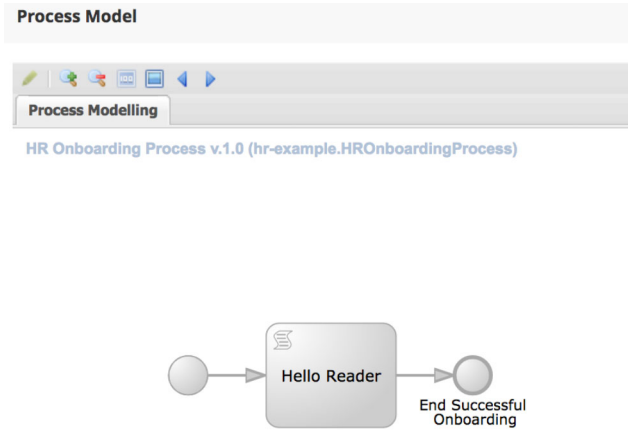


**Figure 7.18** The process model's viewed to validate that it's completed, indicated by the greyed-out nodes on the path already take.

If a process was uncompleted, the active node(s) would have a red circle to identify where that(those) instance(s) was(were) now. As this process instance's finished, the nodes traveled have but one path and it's greyed out.

Finally, you can verify that the script task did the work you asked it to do by inspecting the JBoss BPM Suite server log, which is in the console where you started at the beginning of this chapter. You should see output something like this:

```
16:08:27,160 INFO  [stdout] (default task-21) ------------------------
16:08:27,161 INFO  [stdout] (default task-21) Entering Hello Reader Node
16:08:27,161 INFO  [stdout] (default task-21)
```

```
16:08:27,161 INFO  [stdout] (default task-21) Hello Effective Business
     Process Management with JBoss BPM reader... welcome to your first
     process!
16:08:27,161 INFO  [stdout] (default task-21)
16:08:27,162 INFO  [stdout] (default task-21) Leaving Hello Reader Node
16:08:27,162 INFO  [stdout] (default task-21) ------------------------
```

The welcome message you placed in the script property of the script task node's working perfectly. You also notice that the debug helper statements are telling you that you entered the *Hello* Reader node and left it after doing the task assigned to it. Although it's a nice simple process to get started with, you've accomplished all the tasks listed at the start of section 7.2. It's not bad for your first time working with the process designer, but more interesting's to expand your process to create a more realistic HR Onboarding Process.

Let's move quicker now using what you've learned to expand on this basic process design.

## 7.3 Expanding on the basic design

The next iteration in this section extends the process to start working with process data. The process is going to manage data as it flows through the process, applying that data for decision making using a rule task and mapping data from the process into and out of tasks.

This section teaches the following:

- Add an object from your data model to the process
- Create and manage process variables
- Map incoming data from the generated start process form to process variables and data model objects
- Map process variables into and out of tasks, in this iteration in and out of a rule task
- Validate submitted data using rules in your project
- Map decisions made in a rule task back out to the main process variables
- Validate that this all works to expectations by inspecting process variable states upon process instance completion

To do all of this you need to expand the process by adding input data, map this to your data model objects, and apply that submitted data to your projects validation rules. It might sound intimidating, but step-by-step the previous process expands by only one task node at a time; this is all you need to learn in this section.

### 7.3.1 Adding process variables

The world of BPM that you've been exploring's devoid of the ability to keep track of dynamic data as you manipulate it through the process. You might feel that the data models you designed are involved, but up to now you haven't interacted with your model.

That changes now. Adding *process variables* to your project creates links between the data manipulated in your process and the data model used in your project. Process variables are the same as global variables in programming languages; they reside at the top level of your process and are passed as values into individual task nodes. When a process variable's passed into a task node, a local copy of that process variable's created to work with inside that task node. Task nodes process data and copy that data from inside a task node back out to the originating process variable. Let's look at how to do all of this by picking up where you left off and adding process variables.

Ensure that the HR Onboarding Process is open in the process designer and click on the white background canvas. Now open the *Properties* pane on the right to find that the process properties are available to you for editing. Click on the *Variable Definitions* field to access the pop-up *Editor for Variable Definitions* and you find an empty list of process variables which are defined for this process as shown in figure 7.19.

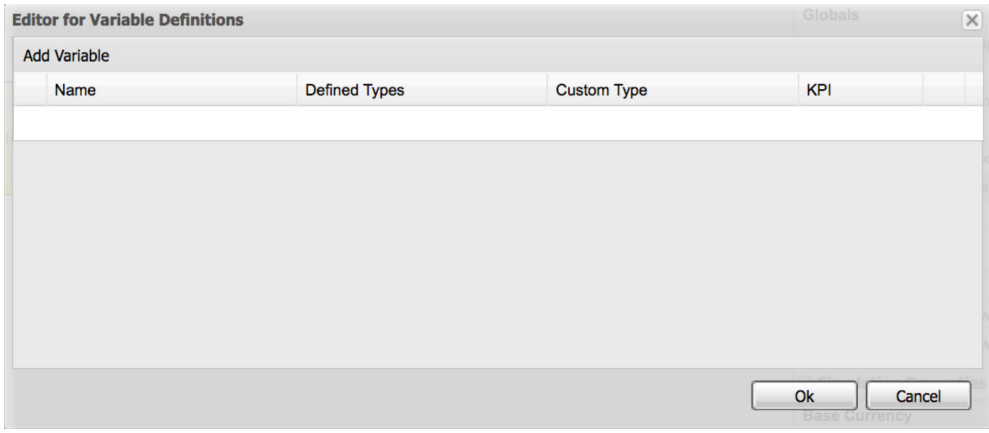


**Figure 7.19** **Process variables are added in this pop-up editor.**

When you click on the *Add Variable* button a new blank entry's added to the list and you can start clicking on the fields to add the following variable:

- Name: **employee_id**
- Defined Types: **String** (select from drop down menu)

Ignore the other fields as they aren't needed now. A question you might ask's why define the type as a *String* instead of an *Integer*? If you look at the *Person* object from this projects data model, you find the id field's defined as an *Integer*. Also, remember that when you've a process deployed and have yet to define a start form, one's automatically generated for you. This form only generates *String* as the type for variables unless it's specified as an object. This means, for now, you define all basic variable types as String unless specified otherwise.

In the next chapter, you learn about form generation and how to design form fields to take other types than strings. For now, this process explicitly shows the true data types entered and uses script tasks to map them into your data model objects.

Now you need to add two more process variables, each having a name and their types defined as strings. First define the employee age:

- Name: **employee_age**
- Defined Types: **String**
- Next define the employee visa type:
- Name: **employee_visa**
- Defined Types: **String**

Finally, you need to have an employee to add all these attributes to. It's time to make use of a *Person* object from your data model. When you created the previous process variables and defined their types, you might have noticed that at the bottom of that drop-down menu the two objects from your data model are listed. This is because the data model's defined in this project and it allows you to make use of any of them as a type. Define your employee process variable as follows:

- Name: **employee**
- Defined Types: **com.redhat.demo.hr_example.Person**

Your process variable editor should look like the one in figure 7.20, with all four process variables listed and their types. Ignore the other fields in the list for now, you won't be using them in this chapter.

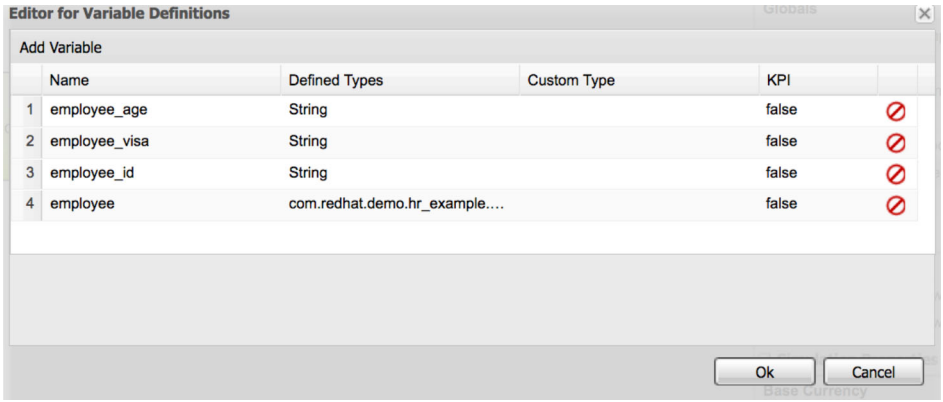


**Figure 7.20 The process variables, all four, defined in the process editor.**

Working with the automatically generated start form for this process means all existing input process variables are shown in the generated start form. This isn't how real projects work, as usually they provide a custom front-end application that submits

process data into your process. To keep things focused on the JBoss BPM Suite and what's available to you, use the provided start form and leave the more advanced front-end aesthetics for exploration in the more advanced examples found online.[1]

Be sure to save your process before you move on to expand the *Hello Reader* script task to map the process variables into the employee process variable.

Now that the process variables are added, starting a process instance makes the data available in the *process context*. You can think of the context of a process as all the data and state information that the BPM engine maintains for the process instance. You can access this information, such as the current node name as shown in section 7.2.3 and process variables. Let's look at how to do this to copy the submitted data from their process variables into the employee object.

Open the Hello Reader script task properties and access the script editor. Add the following lines of Java code as shown in figure 7.21.

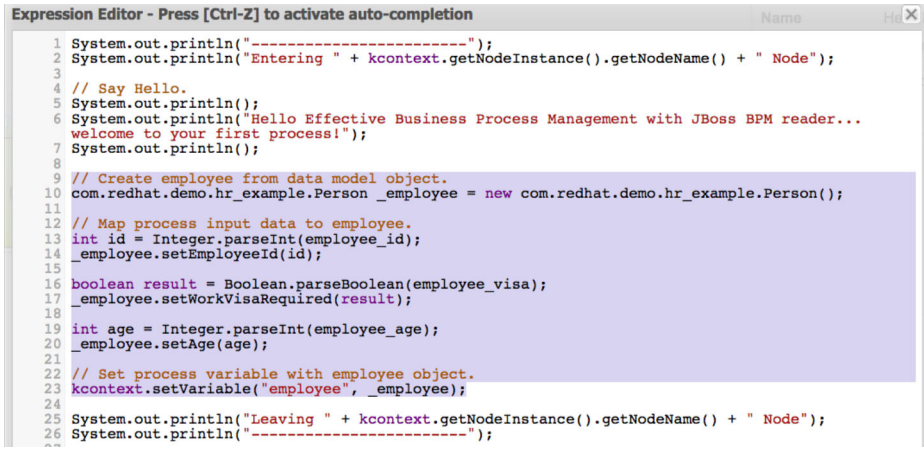


**Figure 7.21 Expanding the initial script task to map submitted data into the employee object.**

To understand what's being done here, let's look at each part of the highlighted section in figure 7.21.

```
// Create employee from data model object.
com.redhat.demo.hr_example.Person _employee = new
     com.redhat.demo.hr_example.Person();
```

The comment states that a local empty copy of the Person object's created and called *_employee*. As a good standard practice, denote local process variables by starting them with an underscore. As you map them to and from process variables of the same name it's helpful to keep things straight in what's a local variable to the task and what's the main process variable.

[1] See chapter two for an example of a Travel Agency booking process with a web front-end used to submit process data.

Now that you've an empty employee object, start mapping in the submitted age, id and whether a work visa's required. Remember, the submitted process variables come from an automatically generated task form to start things off. This means converting any submitted types to the employee object model attribute types, as required. In this case, it's the type *Integer* which is needed for the employee id.

```
// Map process input data to employee.
int id = Integer.parseInt(employee_id);
_employee.setEmployeeId(id);
```

This code converts the *employee_id* string into an integer type before using the proper method to set the employees age. The attribute that determines if an employee needs a work visa's a Boolean value, and here you convert the *employee_visa* String type to a Booleans value before using the proper method to set the employee attribute *workVisaRequired*.

```
boolean result = Boolean.parseBoolean(employee_visa);
_employee.setWorkVisaRequired(result);
```

The age attribute's also an Integer and it's converted and mapped to the employee object like the id attribute.

```
// Set process variable with employee object.
kcontext.setVariable("employee", _employee);
```

To map local process variables from a specific task node you need to access the process variable back in the process context. This provides you one of the programming interfaces and it's known as the *kcontext*. Helper methods exist that you can use, one being the *setVariable* as shown here to map the local *_employee* process variable back to the *employee* process variable.

Once finished, be sure to save your process. Now let's validate the process to ensure these process variable changes work before moving on.

### 7.3.2 Validate your process variables

Validating the process variable extensions is no different than what was shown in section 7.2.3. Check for error messages and deploy the process by building the project from the *Project Editor*.

Because this process was deployed once already, when you validated in section 7.2.3 as version 1.0, it either increases the project version number or *overrides* the existing deployment artifact when the pop-up appears by clicking on the *Override* button shown in figure 7.22.

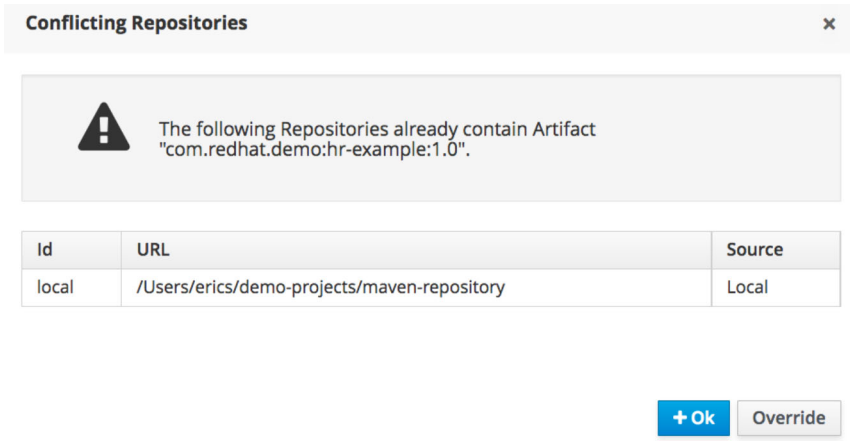


**Figure 7.22 When deploying a project which has been previously deployed, you can override the existing deployment by clicking on the *Override* button in this pop-up.**

From the Process Management menu open the Process Definitions and start the HR Onboarding Process. Enter data into the fields of the presented start process form as shown in figure 7.23. Leave the employee field blank to ensure your script creates the mappings correctly to populate the attributes.

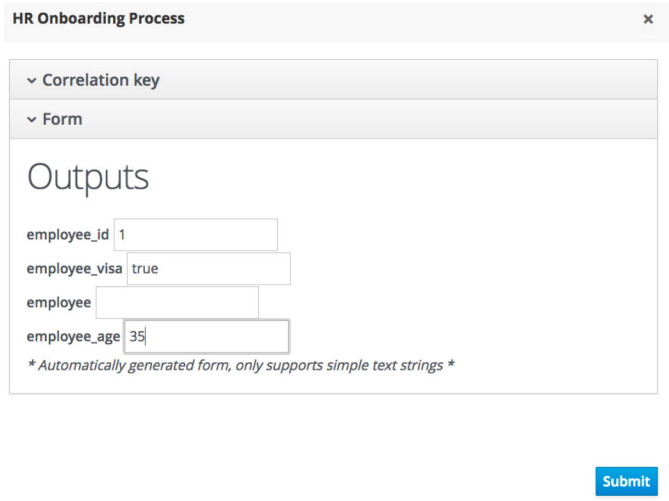


**Figure 7.23 The auto-generated start process from for HR Onboarding process with fields showing sample data. Note that the warning's presented about this form only submitting String types.**

After submitting the form, the process instance runs to completion and the view of the instance's displayed in the panel on the right. Click on the *Process Variables* tab to view the process variables as shown in figure 7.24.

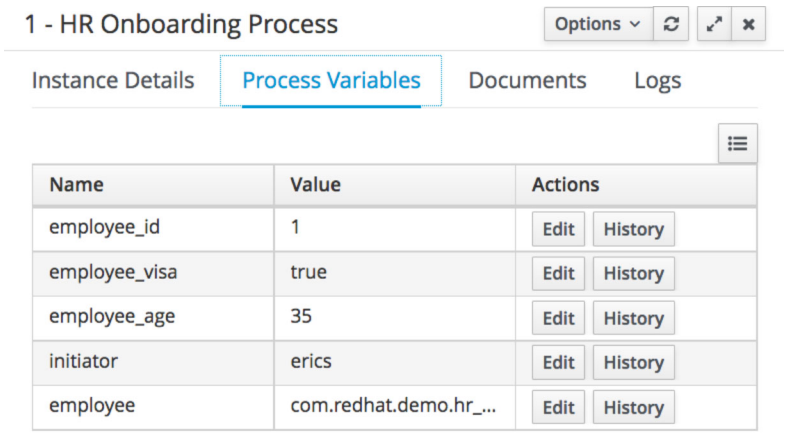


**Figure 7.24 The view of the process instance variables for inspecting that your script did the work of mapping them into the employee object.**

By selecting any variables *History* button under the column *Actions* to view variable change activity. This still doesn't help you to view a complex variable like the employee object, and you need to add a bit more validation in a script task. Before you do this at the beginning of your process, let's assume that this works fine and add that validation after your next extension.

In the next section, let's add validation checks to the submitted employee data using a rule task. These rules determine employee type and if a work visa's needed.

### 7.3.3 Extending the process to integrate rules

This example project came to you with rules for validating the data submitted for a new employee to the HR Onboarding process. In this section, you insert a rule task in your process to apply these given rules for validation against the initial data submitted. You can validate that rules have modified your incoming data with another script task that prints the data that should've been modified. These actions set up the use of a rule based decision to determine an eventual path to take later in the process. More on this after you extend the process with your rules.

To add a rule task between the existing script task *Hello Reader* and the end node, you first need to make some room by removing existing sequence flows. To do this select the *sequence flow* between the *Hello Reader* task and the end node. Click on the red 'X' to remove it as shown in figure 7.25.

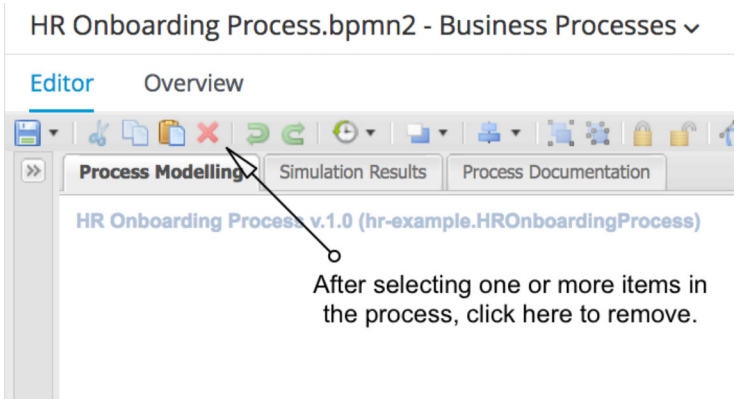


**Figure 7.25** **To remove items from the process designer, select them and click on the red 'X'.**

This button on the task bar's how you remove any item from the process designer, by selecting and then clicking on the red 'X'. If you hover on this icon, it displays a message "Delete all selected shapes." By selecting more than one item in your process and clicking on this deleted button, all are removed.

Now that you've room to add them, insert a rule task and a script task. To connect them use the methods you learned, either dragging from the sidebar or from previous tasks, to connect all tasks with sequence flows. Give the rule task the name *Validate Employee* and let's call the script task *Verify Validation*.

A short note on the use of script tasks as validation for rule execution. It's a visual teaching tool to ensure that process variable or data models are updated as you design a process. Although it's useful and recommended in your daily usage to adopt these practices, realize that they'd be removed once everything's working as expected. Production ready processes won't always benefit from extra script tasks if performance's a concern.

Once you've completed the insertion and naming, your process should look the same as shown in figure 7.26.

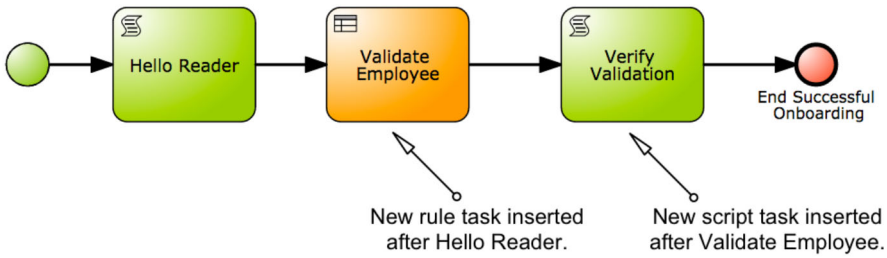


**Figure 7.26** **The extended process with a rule task and a script task inserted.**

The next step's to ensure that the rule task's configured to use the right set of rules. First open and view a few of the rules provided, such as the *Validate Employee Senior* found under the *GUIDED RULES* menu in the *Project Explorer*.[2] Click on the *show options* menu at the bottom of the rule and note that the *ruleflow-group*'s named *validation*. For your design convenience, all the rules in this example project have been assigned to the *ruleflow-group* named *validation*. By assigning *validation* as the group of rules the rule task *Validate Employee* uses, the decision table and guided rules shall be applied.

Checking the age of an employee and assigning them a category by modifying the field *type's* achieved with the guided rules. The decision table contains a set of rules to check each submitted employee to determine the type of work visa needed. The decision table rules modify an employees *Work Visa Type*. These modified fields are exactly the ones you want to check in the script task *Verify Validation*, to ensure they're working.

To assign the *validation* group to your rule task, select *Validate Employee*, open the *Ruleflow Group* property editor by clicking on that field in the *Properties* pane on the right. A pop-up appears and you select the *validation* rules in the *Ruleflow Group Name* field. To ensure that you've the right rules or to view what rules are part of the *validation* group, select the *Rules* drop down menu in the *Editor for Ruleflow Groups* as shown in figure 7.27.

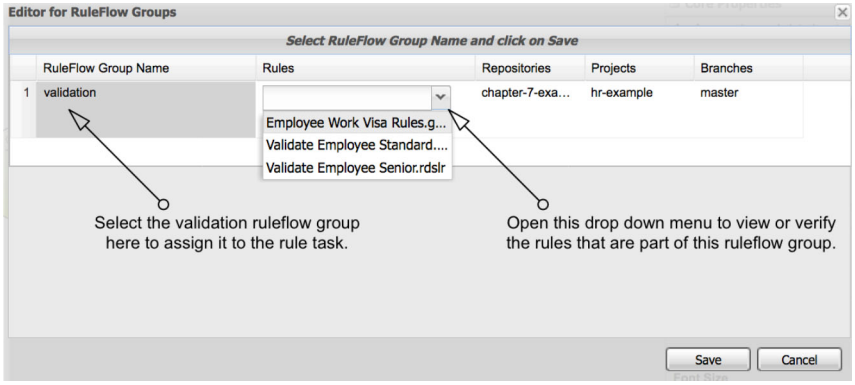


**Figure 7.27 Editor for Ruleflow Groups used to select the rules to be assigned to your rule task.**

The only question that remains is, how can you ensure process variables are updated by the rule task *Validate Employee*?

### MAPPING PROCESS DATA IN AND OUT OF A TASK

It's important to determine the process data to be mapped in to and out of the *Validate Employee* rule task. As previously mentioned, process variables are the global variables and when you work in any task, you either work directly on the global process variables or you create local variables. A rule task's one of the tasks that requires you

[2] For more details on rule creation and rule navigation, see chapters five and six.

to pass in process data that it needs by mapping global process variables in. When it's finished, in this case after applying rules to the data passed in, it needs to know the mapping back out from local variables back to global process variables.

For your rule task, an employee's passed in to the task which creates a local copy for use within the rules themselves. This local copy's modified, depending on the rule outcome, and mapped back out to the employee process variable for use in the rest of your process.

To achieve this, you use the *Assignments* property editor associated with the rule task by:

1. Select Validate Employee.
2. Open the *Properties* editor.
3. Open the pop-up editor by clicking on the *Assignments* field.

Figure 7.28 shows the editor, where you create local variables by using the same global name but marked with a leading underscore[3] and link them to their global process variables. You also define the variable data type from the drop-down menu provided. Once you've completed this activity, save your work and close the editor.

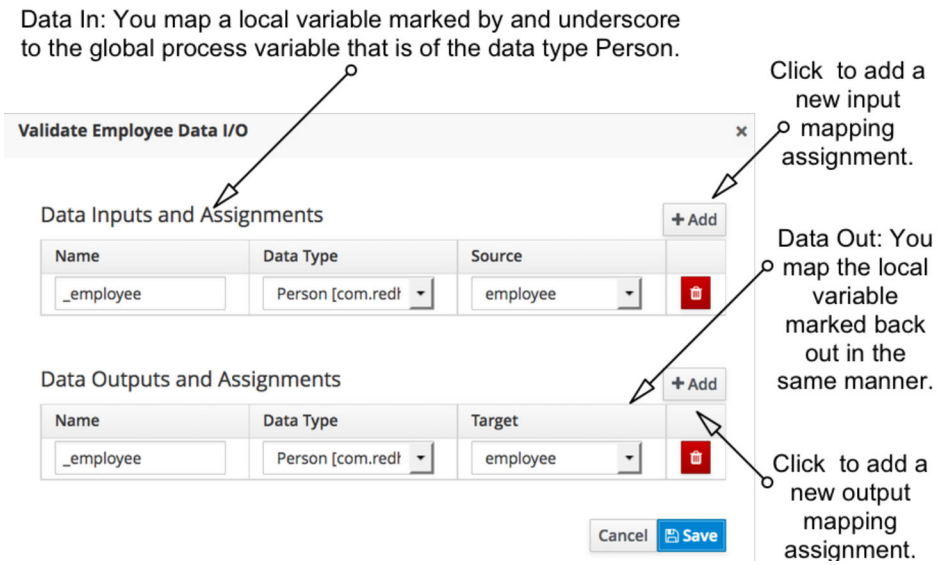


**Figure 7.28 The *Assignments* editor's where you can map global process variables in to local copies and back out.**

[3] The use of the same global process variable names, but marked with a leading underscore for local process variables, is a suggested way of consistently being able to understand the variables you might be tracing during testing of your processes.

Finally, add the same type of logging statements that you did in the Hello Reader task by using the *On Entry Actions* and *On Exit Actions*. If you're unsure how to do this, review the example project as it's been provided for your exploration at points like this.

Save your process work before moving on to add the validation script task work to inspect some of the process variables in your employee to ensure the rules are working in this process as expected.

### 7.3.4 Validate your rule task integration

To validate your rule task process extension, you're adding a few statements like you did in the *Hello Reader* task. This allows you to print the values of process variables that you expect have changed values due to the rule task processing.

First select the script task *Verify Validation*, open the *Properties* pane on the right and click on the *Script* field to open the *Expression Editor*. As previously done in the *Hello Reader* script task, start by entering the statements to log entering this task.

```
System.out.println("-------------------------");
System.out.println("Entering " + kcontext.getNodeInstance().getNodeName() + "
     Node");
System.out.println();
```

The next section creates a variable which is used as a copy of the process variable holding our employee. The first line's a comment and the second copies the process variable *employee* to a local variable. For the local variable, start with an underscore, accessing the *employee* through the process context helper method *getVariable*.

```
// Display current employee fields to validate business rules.
com.redhat.demo.hr_example.Person _employee =
     (com.redhat.demo.hr_example.Person) kcontext.getVariable("employee");
```

Now that you've a copy of the employee you can start to log a few items to verify that the attributes within the employee were changed as expected. The *validation* rules contain checks to set the type of employee (either Senior or Standard) and to set the type of work visa that the employee needs (A, B, C or D). The following shows how you can access the fields:

```
System.out.println("Employee type: " + _employee.getType());
System.out.println();
System.out.println("Work visa type: " + _employee.getWorkVisaType());
System.out.println();
```

Finally, you add the now well know statements to log leaving this task as follows.

```
System.out.println("Leaving " + kcontext.getNodeInstance().getNodeName() + "
     Node");
System.out.println("-------------------------");
```

Figure 7.29 shows the completed statements in the *Expression Editor*. Be sure to save your work by clicking on the *OK* button, saving your process before you test your work.


**Figure 7.29 The final script for *Verify Validation* task.**

If you've been following along, there's already one deployment of this project, and you're now re-deploying. By first bumping the project version number, you're indicating a new version's deployed. Increase the project version number as follows:

1. Open the Project Editor.
2. Increase the *Version* number from 1.0 to 1.1.
3. Save it with a statement about releasing validation rule improvements.
4. Build and deploy the project.

You should see a green pop-up with the message that the build was successful.

To ensure a new process version was deployed, you open the *Process Deployments* perspective from the *Deploy* menu at the top to see a list of deployed units on your JBoss BPM Suite server. There should be an entry that lists the projects being worked on and include a version number at the end as follows.

```
com.redhat.demo:hr-example:1.1
```

With that in place you can now start a new process instance by first opening the *Process Definitions* from the *Process Management* menu at the top. What you might notice is that the *HR Onboarding Process* is listed, but with a version number of 1.0.

How can that be?

The thing to keep clear in your mind's that the process is an artifact and that the project contains all the artifacts, such as rules, processes, and your data model. The process artifact has a version number but we didn't increase that to 1.1, we increased the project version number to 1.1. It's possible to deploy a new version of the same process, but you need to increase the process artifact version number in the *Properties* editor in the process designer. Let's leave the process version number at 1.0 and continue on to validating the business rules.

Start a process instance of the *HR Onboarding Process* as you did before, via the *Start* button and submit the same data you did before or try new data as you explore the workings of the validation rules.

Here I submit the form using the same data as shown in figure 7.23 and view the server logs from the console where you started the JBoss BPM Suite and you should see something like the following.

```
14:15:41,838 INFO  [stdout] (default task-61) ------------------------
14:15:41,838 INFO  [stdout] (default task-61) Entering Validate Employee
     Node...
14:15:41,838 INFO  [stdout] (default task-61)
14:15:41,906 INFO  [stdout] (default task-61)
14:15:41,906 INFO  [stdout] (default task-61) Leaving Validate Employee
     Node...
14:15:41,906 INFO  [stdout] (default task-61) ------------------------
14:15:41,907 INFO  [stdout] (default task-61) ------------------------
14:15:41,907 INFO  [stdout] (default task-61) Entering Verify Validation Node
14:15:41,907 INFO  [stdout] (default task-61)
14:15:41,907 INFO  [stdout] (default task-61) Employee type: Standard
14:15:41,907 INFO  [stdout] (default task-61)
14:15:41,907 INFO  [stdout] (default task-61) Work visa type: B
14:15:41,907 INFO  [stdout] (default task-61)
14:15:41,907 INFO  [stdout] (default task-61) Leaving Verify Validation Node
14:15:41,907 INFO  [stdout] (default task-61) ------------------------
```

Note that I've removed the output from the *Hello Reader* node for brevity, but you see the rule task *Validate Employee's* processed and that the rules have identified that this employee's *Standard* and needs a *B* work visa.

Let's extend the process by adding a split in the processing path, allowing for a decision to be made as to the right path to take. Decisions are reached by evaluating data attributes from our employee and in this example, take one path for senior employees and another for standard employees.

### 7.3.5 Extend process with a gateway

The last extension in this chapter's splitting the process execution path in two; one path for senior employees and another for standard employee onboarding. Adding *gateway*, which is a special element which is diamond shaped and contains an 'X', allows a single incoming path to split into one or more exit paths. The process designer offers several types of gateways, but not all are covered in this chapter. By using an XOR gateway the process exits only one path when it continues processing. The gateway you're using is shown in figure 7.30.

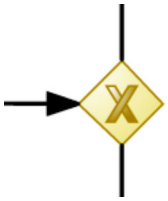

**Figure 7.30 A gateway's used to split the process execution flow from one path into several exit paths.**

As you've become a bit of an expert at using the process designer, I'm going to provide you with a list of steps to design this extension and only detail the new gateway element:

1. Remove the exit sequence flow from the script task *Validate Employee* and the end node from your process.
2. Add a gateway, either using drag-and-drop or using the mini object library.
3. Select the gateway by clicking on it.
4. Activate the configuration menu by clicking on the wrench icon below the gateway as shown in figure 7.31.

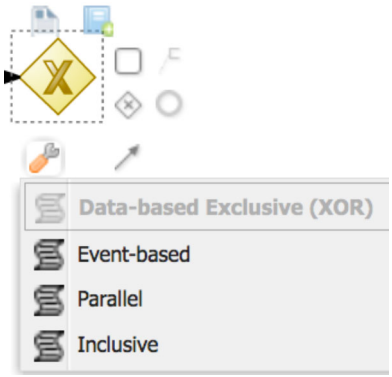


**Figure 7.31 The configuration menu for a gateway allows you to choose the type you need.**

5. By ensuring the gateway's a *Data-based Exclusive (XOR)*, only one exit path's taken.

Before coming back to configuring the gateway, finish connecting the process paths that exit the gateway. Here's the task list completing the design for both exit paths from this gateway:

1. Use drag-and-drop from the *Object Library* on the right or use each element's mini icon object library to add new elements
2. Start with the senior path by exiting the gateway with a transition to a script task, be sure the task isn't generic by using the configuration wrench to select type *Script*.
3. Name the script task: *Path for Senior Employee*

4 End the senior path by exiting the *Path for Senior Employee* script task and transitioning to an end node
5 Name the end node: *End Senior Onboarding*
6 Back at the gateway, start a second path for the standard employee by exiting the gateway with a transition to a script task
7 Name the script task: *Path for Standard Employee*
8 End the standard path by existing the *Path for Standard Employee* script task and transitioning to an end node
9 Name the end node: *End Standard Onboarding*

Now you must drag around the new items in your process to resemble figure 7.32, making this process pleasing to view.

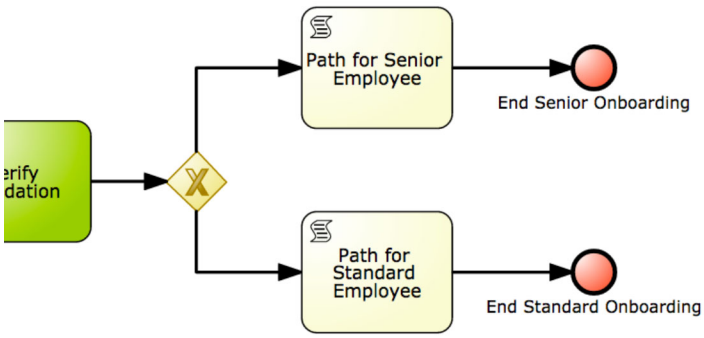


**Figure 7.32 By dragging the tasks around you can end up with an aligned process which is extended with a gateway and two possible paths towards completion of the HR Onboarding Process.**

A nice feature to add now's colors to each node. A nice visually pleasing effect's to color the path you desire to be taken, sometimes referred to as the *happy-path*, green. This doesn't affect actual processing functionality, but does make business discussions around the process more effective. Now the only thing missing's to define script statements in their respective *Script Expression Editors*. The code provided here works for both paths.

```
System.out.println("------------------------");
System.out.println("Entering " + kcontext.getNodeInstance().getNodeName() + "
     Node");
System.out.println();

System.out.println("Leaving " + kcontext.getNodeInstance().getNodeName() + "
     Node");
System.out.println("------------------------");
```

The only thing left to do's to ensure that the gateway has a way to decide which path to take, which isn't determined in the gateway node itself, but on each of the outgoing transitions. These contain an *Expression* field when you open the *Properties* pane on the right after selecting a transition that can process a Boolean value (true or false). This is how a decision's made as to which path to take in your process, by using the guided editor which automatically lets you choose from all available process variables.

To ensure a Boolean process variable exists, you need to go back to the process properties by clicking on the background of the process and opening the *Properties* pane on the right. Using the *Variable Definitions* add a new Boolean variable *employeeIsSenior* as shown in figure 7.33.

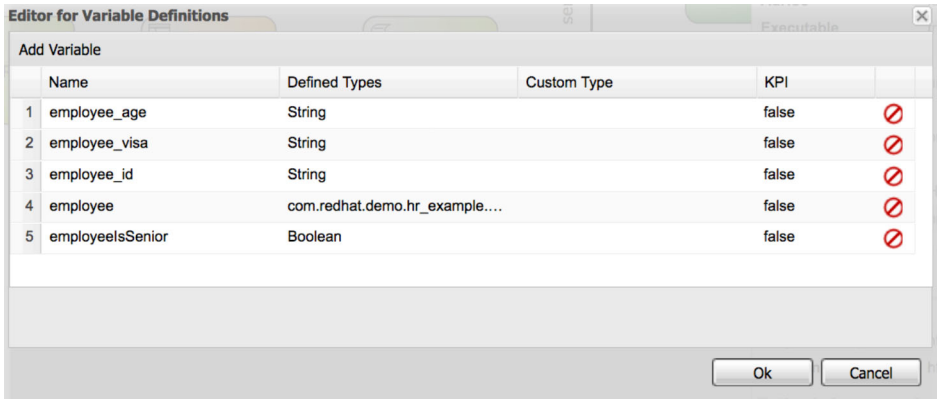


**Figure 7.33 Added variable employeeIsSenior for evaluation in your process gateway.**

Now that you've a process variable to work with, you need to ensure that, once an employee type's determined, you can set this process variable. This happens after the rule task *Validate Employee*, and in the subsequent *Verify Validation* script task you can add the following to ensure the process variable *employeeIsSenior* is properly initialized based on the employee data submitted to your process.

```
// Set employeeIsSenior for gateway path extension.
kcontext.setVariable("employeeIsSenior", new Boolean("False"));
if (_employee.getType().equals("Senior")) {
  kcontext.setVariable("employeeIsSenior", new Boolean("True"));
}
```

Adding these messages before the closing statements marks leaving this script task. Now open the transition properties by clicking on the transition leading to the senior path, update the name of this transition to *senior* and open the *Expression* editor by clicking on that field. In figure 7.34 you see that you can select *employeeIsSenior* from the existing process variable in the drop-down menu and select the *Condition* to be *is true*.

Do the exact same thing for the transition leading to the standard path, but name it *standard* and set *Condition* for *employeeIsSenior* to *is false*. This ensures that this path's

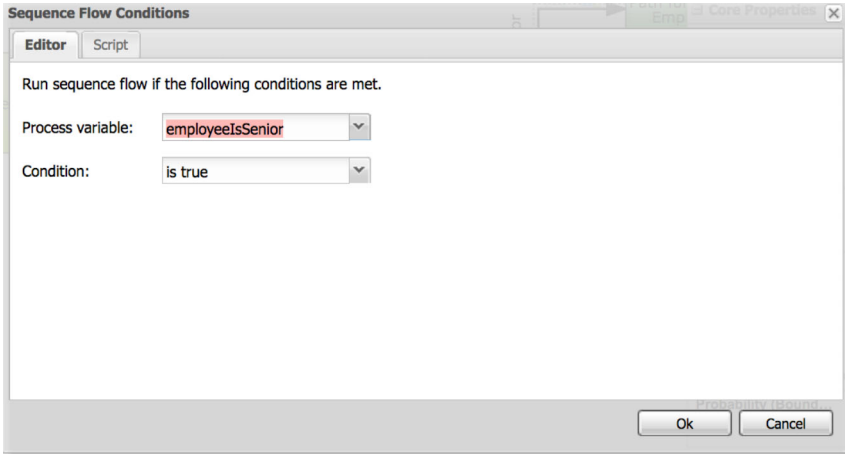


**Figure 7.34 The senior transition path's taken if the process variable employeeIsSenior is true.**

only taken if the process variable's set to false, which happens when the employee data determines that it's a standard type employee onboarding. This completes the process extensions where you added a gateway and two paths to your process for onboarding two different categories of new employees.

Save the process, build and deploy it. Next, start several process instances with data as shown before in figure 7.23. Notice the new field for the added process variable? This gets automatically generated by the build and deploy step. Leave this new field blank. Attempt to submit new employees to traverse each path, such as setting the age to 30 for the standard path and age to 60 for the senior path.

This finishes up the HR Onboarding Process for now, as you've the knowledge to continue exploring on your own with the process designer. You've not worked with all possible BPM constructions, but it's a matter of experience gained by expanding your process design skills going forward. To help with more complex examples beyond the scope of this book, let's look at compiling a list of example projects you can browse on your own to learn more.

## 7.4 Considering advance process design concepts

In this final section, I give a few pointers to projects that show working examples of more advanced process design concepts. You'll need the following process design concepts in many processes you design, and it's a good starting point to begin your exploration:

- Integration of a standard web service
- Integration of an external spread sheet of rules
- Using sub-processes
- Triggering another action in a process with a signal from the current process flow

Let's discuss each of these briefly and point to examples you can further explore. Your process design toolbox should be heavy enough at this point to allow you to expand into even more advanced process design concepts.

### 7.4.1 Using a web service from BPM process

Organizations might use many services and many types of services from a business process; for example, RESTful services, web services or microservices. In chapter two you explored a travel agency booking process that used two web services that look up hotel and flight information.[4]

You understand now how the process designer allows you to select a task, such as a web service task and view its properties in the *Properties* pane on the right. The hotel or flight service tasks show how a deployed web service's reached from a web service task. Explore how process context data's passed into the web service task. Responses from the web service's mapped back out to process context variables, like you did with a rule task.

If that example isn't enough, I've added a Baggage Delivery example project[5] for you to explore how another web service task was used to integrate a service into that process. The use of a RESTful service or any generic service's now within your reach, as they operate in the same manner as these examples show.

Let's look at integrating an external set of rules, stored in a spreadsheet, which is something often encounter in organizations.

### 7.4.2 Adding an external spreadsheet of rules to a process

This is an example of using an existing set of rules that were maintained in a spreadsheet before JBoss BPM Suite came along. These rules might be easier for their owners to keep in a spreadsheet and continue to use the value put into that spreadsheet. Integrating a spreadsheet into a JBoss BPM Suite process project's preferred to re-implementing these rules into the integrated decision table tooling.

Once the spreadsheet's integrated as an *XLS Decision Table in to your process project*, use the rules using rule tasks. Defining rules as a ruleflow-group name allows their use in rule tasks as you'd expect.

To provide you with a working example project, you can take a look at the Baggage Delivery example project, which has *Non-US Shipping Cost* rules in an external spreadsheet, which you find in the project's *support* directory, and a rule task integrating them called *Calculate Shipping*. Uploading the spreadsheet as a file to the *XLS Decision Table project artifact* ensures changes are applied.

---

[4] The example project's found at https://github.com/effectivebpmwithjbossbpm/chapter-2-travel-agency-demo, see chapter two for installation and detailed walk through where the two web services are discussed.

[5] The JBoss BPM Suite Baggage Delivery demo project is found at https://github.com/effectivebpmwithjbossbpm/bpms-baggage-delivery-demo, it's setup like all the other example projects. Setup and explore this project to learn more about using web service tasks in your BPM processes.

### 7.4.3 Calling a sub-process

When you've a process design, it's common to want to split this into smaller pieces and thereby create reusable processes in your organization. Imagine the process of calculating the price of some article or a process that creates a new customer in your database. Designing these once and reusing them by calling from a *sub-process task*.

Sub-processes are processes, each with its own name and id, called from a sub-process task using its process id. The calling process is suspended and waits for a signal from the sub-process it completed. The sub-process starts a new thread of execution and does all the work that it encounters until it reaches an end node. Upon reaching the end it signals back to the calling sub-process task that it's done. The calling process then picks up and carries onwards in its tasks.

To see a working example of this you can examine two process pieces, the *calculatepriceprocess* and the *compensateService* process, both called as sub-processes from the *specialtripsagencyprocess* in chapter two. By clicking on the sub-process task and examining the *Properties* pane on the right you see it only requires an id of the process you want to call. If needed, the now familiar *Assignments* editor defines the process context variable mappings into and out of the sub-process.

As a final example, let's look at how to send a signal to trigger some action in a process.

### 7.4.4 Sending signals to trigger actions in a process

Triggering a call to start a sub-process is done with a signal. It needs a process id and the calling process then signals that process id to start a new process instance. Sometimes you want a process to take a path to some actions that needs to wait until something happens later in that process. How can you send a signal from later in your process, back to that point where it's waiting, and trigger that process path to continue?

From a currently executing process, a signal's sent that triggers the waiting action. An example of this is provided in the JBoss BPM Generic Loan demo[6], where a process is going to need a user to approve a request later in the process which needs to then trigger a waiting path in that same process to register this activity in an external database system that the customer's approved for a loan.

Figure 7.35 shows a view of the process in question where the signal comes in to play.

In this process, a gateway splits into two parallel paths where both continue processing simultaneously. The lower path has a *signal task* called CC Event, which means it's going to register a customer contact moment in a database but only if the customer's approved for a loan. The other path's where the decision's made by a user in the *Approve* user task to deny or grant the loan to this customer. Use the *Properties* pane

[6] The JBoss BPM Generic Loan demo project's found at https://github.com/effectivebpmwithjbossbpm/bpms-generic-loan-demo, it's set up like all the other example projects. Set up and explore this project to learn more about using signals in your BPM processes.

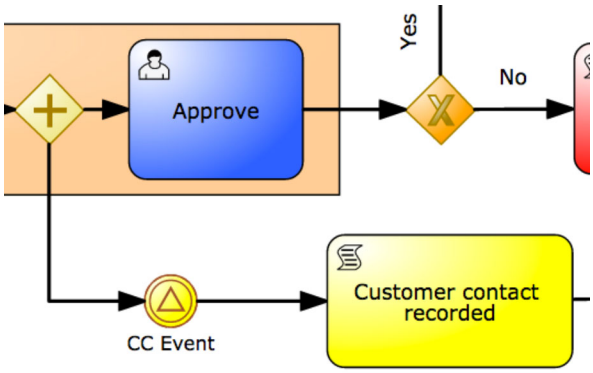


**Figure 7.35 A part of the Generic Loan process where a signal's needed to trigger the *CC Event* to move forward. The user task *Approve* sends a trigger from the *On Exit Actions*.**

after selecting this user task to view the *On Exit Actions* to see how a signal's sent to trigger the *CC Event* task to process further and record a customer contact.

In this chapter, you've worked with the process designer to set up a new process and created your first design. From there you were given tips to help in testing your process design and shown how to deploy the process. Next you've interacted with your process and run instances using the auto generated start forms. When your process instances finished, you visually verified process instance execution and examined process data to validate the expected behavior in your process. After several iterations, your process was well on its way and you explored a few advanced design concepts. These examples have shown the possibilities, now it's up to you to continue expanding your abilities to design business processes with JBoss BPM Suite.

## 7.5 Summary

- Business process design starts with a simple process and is extended with iterations.
- Decisions are made in processes by applying rules using rule tasks.
- Process context variables manage data as it flows through a process.
- Tasks use process data by mapping high level process context variables in to local task variables for use in the task.
- Local task variables are mapped back out to high level process context variables to preserve changes applied in a task.
- A gateway allows you to split the execution flow in a process, such as the XOR gateway that chooses only one of the outgoing paths.
- Use scrip tasks to verify expected data outcomes. This validates task functionality at the process level.
- Logging the entry and exit of a task is achieved by using the *On Entry* and *On Exit* actions in a task where available. This provides process flow tracing for process level debugging.

- Web services tasks, signals, sub-processes and more are available to explore in provided example projects.
- You can explore deployed processes in the Process Management perspective. This provides insights into process deployments, process instance progress, process context and variable states.
- Process instances can be viewed and details of their execution examined, such as process variables and the visual model viewer.
- A process start form's automatically generated based on the process variables and it's used to start a process instance.

# index

## Symbols



## A



## B

## C

## D

## E



## F



## G

## H



## I



## J

## K



## L



## M



## N



## O

## P

## R

## S

## T



## U

## V



## W



## X