Learn by doing: less theory, more results

Activiti 5.x Business Process Management

A practical guide to designing and developing BPMN-based business processes

ATTUNE INFOCOM
Tune into Enterprise Open Source

Beginner's Guide

Dr. Zakir Laliwala
Irshad Mansuri

[PACKT] PUBLISHING open source* community experience distilled

Activiti 5.x Business Process Management Beginner's Guide

A practical guide to designing and developing BPMN-based business processes

Dr. Zakir Laliwala

Irshad Mansuri

[PACKT] open source
community experience distilled
PUBLISHING
BIRMINGHAM - MUMBAI

Activiti 5.x Business Process Management Beginner's Guide

First published: March 2014

Production Reference: 1190314

Published by Packt Publishing Ltd.
Livery Place
35 Livery Street
Birmingham B3 2PB, UK.

ISBN 978-1-84951-706-5

www.packtpub.com

Cover Image by Asher Wishkerman (wishkerman@hotmail.com)

Credits

Authors

Dr. Zakir Laliwala

Irshad Mansuri

Reviewers

Sofiane Bendoukha

Ivano Carrara

Mike Dias

Acquisition Editors

Anthony Albuquerque

Neha Nagwekar

Content Development Editor

Vaibhav Pawar

Technical Editors

Krishnaveni Haridas

Arwa Manasawala

Veena Pagare

Manal Pednekar

Ankita Thakur

Copy Editors

Tanvi Gaitonde

Aditya Nair

Project Coordinator

Leena Purkait

Proofreaders

Maria Gould

Paul Hindle

Indexers

Monica Ajmera Mehta

Rekha Nair

Graphics

Ronak Dhruv

Abhinash Sahu

Production Coordinator

Kyle Albuquerque

Cover Work

Kyle Albuquerque

About the Authors

Dr. Zakir Laliwala is an entrepreneur, open source specialist, and a hands-on CTO of Attune Infocom. Attune Infocom provides enterprise open source solutions and services for SOA, BPM, ESB, portal, cloud computing, and ECM. At Attune Infocom, he is responsible for the delivery of solutions and services and product development. He explores new enterprise open source and defining architecture, roadmaps, and best practices. He has consulted and provided training on various open source technologies, including Mule ESB, Activiti BPM, JBoss jBPM and Drools, Liferay Portal, Alfresco ECM, JBoss SOA, and cloud computing, to corporations around the world.

He has a PhD. in Information and Communication Technology from the Dhirubhai Ambani Institute of Information and Communication Technology. He was an adjunct faculty member at the Dhirubhai Ambani Institute of Information and Communication Technology (DA-IICT) and at CEPT University where he taught students at the Master's level.

He has published many research papers in IEEE and ACM International Conferences on web services, SOA, grid computing, and Semantic Web. He also serves as a reviewer at various international conferences and journals. He has also published book chapters and is writing a book on open source technologies. He is the co-author of *Mule ESB Cookbook*, *Packt Publishing*.

Irshad Mansuri has more than three years of experience in implementing Java, J2EE, Activiti, JBPM, and Liferay solutions. He has successfully migrated the IBM WebSphere Portal to Liferay Portal for a client based in the UK. He has successfully delivered and managed projects in Liferay, Alfresco, jBPM, and Activiti. He has also delivered training on Liferay Portal, Activiti, and jBPM to various clients across the globe. He is responsible for implementing, deploying, integrating, and optimizing portals and business processes using Activiti, jBPM, and Liferay development tools.

About the Reviewers

Sofiane Bendoukha holds a Magister of Computer Science degree from the University of Science and Technology of Oran (USTO), Algeria (2010). Since November 2011, she has been a PhD student at the Theoretical Foundations of Computer Science Group at the University of Hamburg, Germany (`http://www.informatik.uni-hamburg.de/TGI/studenten/bendoukha/bendoukha.html`).

She has always had an interest in workflow modeling and distributed systems. During her Magister thesis, she worked on web service orchestration with BPEL in grid computing. Her PhD thesis revolves around workflow management in cloud computing, especially in Intercloud environments. She focusses more on modeling issues using high-level Petri nets (HLPN) techniques and agents. The outcome covers many aspects of the workflow lifecycle, that is, from modeling to implementation.

She has also worked for two years as a teaching assistant at the Meteorological Institute for Formation and Learning (IHFR) in Oran, Algeria. After that, for one-and-a-half years, she worked as a system engineer at the Remote Sensing Laboratory related to the Algerian Space Agency (`http://www.asal.dz/`). For the actual publication, see `http://www.informatik.uni-trier.de/~ley/pers/hd/b/Bendoukha:Sofiane.html`.

Ivano Carrara was born on January 22, 1960. He has been an expert in electronics since 1979. He started to work with IBM Italia in 1984 and since 1989, he has managed the Italian operations of Galacticomm Technologies, Inc. (FL). He has been working with Liferay since 2004 and with Alfresco since 2007. As Solutions Architect, he cooperated with Accenture in activities based on the Liferay Enterprise Portal. He is currently working on the integration of Activiti BPM, Liferay Portal, Alfresco ECM, Jasper Reports, and Orbeon Forms on a large project related to the travel industry.

Mike Dias has been an Activiti BPM project committer since 2013, contributing to various project areas. He works for a Brazilian company called iColabora as a technical leader of a BPMS built on top of the Activiti Engine.

He is fluent in Java, and also has OCJP and OCWCD certifications. He also dominates JavaScript and has a lot of experience with libraries such as jQuery, Backbone.js, and d3.js.

Recently, he discovered the power of graph databases that gave him a one-way ticket to NoSQL land.

www.PacktPub.com

Support files, eBooks, discount offers and more

You might want to visit `www.PacktPub.com` for support files and downloads related to your book.

Did you know that Packt offers eBook versions of every book published, with PDF and ePub files available? You can upgrade to the eBook version at `www.PacktPub.com` and as a print book customer, you are entitled to a discount on the eBook copy. Get in touch with us at `service@packtpub.com` for more details.

At `www.PacktPub.com`, you can also read a collection of free technical articles, sign up for a range of free newsletters and receive exclusive discounts and offers on Packt books and eBooks.

`http://PacktLib.PacktPub.com`

Do you need instant solutions to your IT questions? PacktLib is Packt's online digital book library. Here, you can access, read and search across Packt's entire library of books.

Why Subscribe?

- Fully searchable across every book published by Packt
- Copy and paste, print and bookmark content
- On demand and accessible via web browser

Free Access for Packt account holders

If you have an account with Packt at `www.PacktPub.com`, you can use this to access PacktLib today and view nine entirely free books. Simply use your login credentials for immediate access.

Table of Contents

Preface 1

Chapter 1: Installing Activiti 7

Getting started with Activiti BPM 8

The Activiti Engine 8

The Activiti Modeler 10

The Activiti Designer 10

The Activiti Explorer 10

The Activiti REST 10

Downloading and installing Activiti BPM 11

Prerequisites for Activiti 11

JDK 6 or higher 11

Eclipse Juno or Indigo 12

Installing the Activiti framework 12

Time for action – installing the Activiti framework 12

Time for action – creating our first process in Activiti 15

Time for action – unit testing to see the result 21

Summary 23

Chapter 2: Modeling Using the Activiti Modeler 25

Understanding Business Process Modeling 26

The BPM lifecycle 26

The design phase 27

Modeling 27

Execution 27

Monitoring 27

Optimization 27

Understanding the BPM standards 28

Getting started with BPMN elements 28

Flow objects 29
Events 29
Activities 30
Gateways 31
Connecting objects 32
Sequence flow 32
Message flow 33
Associations 33
Swim lanes 33
Pool 33
Lanes 33
Artifacts 34
Data object 34
Group 34
Annotation 34
Installing the Activiti Modeler 35
Time for action – Business Process Modeling using the Activiti Modeler 35
Time for action – import and export of a model 45
Time for action – deploying a model into the Activiti Engine 48
Summary 49
Chapter 3: Designing Your Process Using the Activiti Designer 51
Understanding the Activiti Designer 52
Time for action – downloading and installing the Activiti Designer 52
Time for action – designing your first process 54
Time for action – testing your process 66
Time for action – importing a process from the Activiti Modeler to the Activiti Designer 68
Summary 71
Chapter 4: Management and Monitoring Using the Activiti Explorer 73
An overview of the Activiti Explorer 74
A process with the Activiti Explorer 75
Time for action – starting a process instance 76
Managing tasks with the Activiti Explorer 79
Time for action – working with user tasks 80
Managing processes with the Activiti Explorer 83
Time for action – process management 84
Reporting with the Activiti Explorer 88
Time for action – reporting using the Activiti Explorer 89
Administration using the Activiti Explorer 93
Time for action – administration using the Activiti Explorer 96
Time for action – changing to a production-ready database 99

Time for action – deploying a process using the Activiti Explorer **100**
Summary **102**

Chapter 5: Development Using the Process Engine **103**
Understanding the Activiti development environment **104**
The Activiti Spring layer 105
The Activiti Engine layer 105
PVM 105
Setting up the Activiti development environment **105**
Configuring the Activiti Process Engine **109**
Time for action – configuring a database **110**
Time for action – the logging configuration **112**
Time for action – configuring the mail server **113**
Time for action – running the Activiti Engine **122**
Summary **124**

Chapter 6: The Activiti ProcessEngine API **125**
Overview of the APIs and services **125**
Delving into the process engine **126**
RuntimeService 127
RepositoryService 127
Time for action – deploying the process **128**
Time for action – starting a new process instance **133**
TaskService 135
Time for action – querying for user tasks **135**
Time for action – creating and completing user tasks **138**
Time for action – suspending a process **142**
ManagementService 145
IdentityService 145
Time for action – working with users **145**
HistoryService 146
Time for action – querying historical activities **146**
FormService 148
Time for action – unit testing and debugging processes **149**
Summary **154**

Chapter 7: Working with the REST API **155**
Overview of the Activiti REST API **156**
Time for action – implementing the REST service **156**
Working with REST API services 161
Time for action – working with Repository APIs as REST **161**
Time for action – working with processes **168**

Time for action – working with tasks **175**
Time for action – working with users **178**
Time for action – working with management **180**
Summary **185**

Chapter 8: Integrating Activiti with Third-party Services **187**
Using the Liferay Portal with Activiti **187**
Understanding portals 188
Exploring the Liferay Portal 188
Time for action – integrating Liferay with Activiti **188**
Integrating business rules in Activiti **195**
Drools Expert 195
Drools Fusion 195
Drools Planner 195
Time for action – integrating and implementing business rules with Activiti **196**
Deploying Activiti as an OSGi bundle **205**
Time for action – integrating Activiti with Apache Karaf **206**
Summary **212**

Chapter 9: Implementing Advanced Workflows **213**
Understanding parallel gateways **213**
Time for action – implementing a parallel gateway **214**
Getting started with Sub-Processes **220**
Embedded Sub-Processes 221
Standalone Sub-Process 221
Event Sub-Process 222
Time for action – working with BPMN 2.0 Sub-Processes **222**
Understanding multi-instance processes **235**
Time for action – implementing a multi-instance process **235**
Introducing execution and task listeners **238**
Execution listeners 239
Task listeners 239
Time for action – implementing execution and task listeners **240**
Monitoring workflows using BAM and CEP **244**
Understanding BAM 244
Understanding CEP 245
Monitoring using Esper **245**
Time for action – using Esper with Activiti **245**
Summary **252**

Pop Quiz Answers **253**

Index **255**

Preface

Activiti is a lightweight, open source workflow and a Business Process Management (BPM) platform distributed under the Apache license. It can also be clustered and managed on a Cloud environment. It is the best platform to build BPM for people-to-people communication. It can be used very easily in every Java environment. It supports all aspects of BPM in the full context of software development. You can build any custom process language as per your requirement. It is a technology that is welcomed, because it allows people to use the tools that they are accustomed to, instead of replacing them with new ones. You can even involve a person with minimal technical know-how in implementing business workflows. It's easy to get up and running with the setup utility. It also has a superfast and rock-solid BPMN 2 process engine, which provides the best platform for collaboration between non-technical business users and technical developers.

Activiti 5.x Business Process Management Beginner's Guide takes you through a practical approach to using Activiti BPM. This book provides detailed coverage of various BPM notations used for business process development. It focuses on practical examples, instead of just theory. This book offers solutions to various real-time issues in monitoring business processes within a system. The topics are always associated with exercises that will allow you to put the concepts to the test immediately.

This book gives a strong overview of the Activiti BPM with real-time examples. The main objective of this book is to guide you step by step through the practical implementation and integration of Activiti with portals, rule engines, and various other technologies. You will be able to use various services provided by Activiti using REST API based on the practical examples covered in this book. This book will become your trustworthy companion, filled with the information required to integrate BPM with your portal.

What this book covers

Chapter 1, *Installing Activiti*, discusses Activiti's core concepts and terminologies. It also provides an environment setup for Activiti BPM. By the end of this chapter, you will be familiar with the installation of Activiti and the deployment of Activiti Explorer with the Apache Tomcat server. At the end of this chapter, you will learn how to create a simple process in Activiti and perform the unit testing of processes.

Chapter 2, *Modeling Using the Activiti Modeler*, helps you design a business process using Activiti Modeler. You will also learn how to import and export models in Activiti Modeler. At the end of this chapter, you will be aware of BPM standards and BPMN elements, and be able to model your business process in Modeler.

Chapter 3, *Designing Your Process Using the Activiti Designer*, introduces us to the Activiti Designer. You will learn to integrate Activiti Designer with Eclipse, design a process, and test it using Designer.

Chapter 4, *Management and Monitoring Using the Activiti Explorer*, helps you understand the various functionalities of Activiti Explorer. You will learn to manage various tasks, processes, and users. You will also learn to generate reports in Explorer. At the end of this chapter, you will be able to deploy processes in Activiti Explorer and perform various functionalities, such as starting a process instance, monitoring a business process, and completing a task.

Chapter 5, *Development Using the Process Engine*, describes an Activiti development environment. You will learn about the different types of layers in the Activiti engine. At the end of the chapter, you will be able to change the default database of Activiti.

Chapter 6, *The Activiti ProcessEngine API*, explains the different APIs and services of Activiti. It demonstrates how you can start process instances and deploy processes managing user tasks using different APIs and services, such as process engine, runtime, management, and history. By the end of this chapter, you will be able to perform unit testing and debugging for a process.

Chapter 7, *Working with the REST API*, gives an overview of the Activiti REST API. In Activiti, you can use the REST API for managing processes, users, and tasks. You will learn how to implement the REST service in Activiti.

Chapter 8, *Integrating Activiti with Third-party Services*, will teach you about the integration of Activiti with various portals. You will be able to perform integration with Liferay, Drools, and OSGi. You will also learn how to configure Activiti with Apache Karaf.

Chapter 9, *Implementing Advanced Workflows*, explains Sub-Processes, multi-instance processes, and parallel gateways. You will learn about process modeling with execution and task listeners. By the end of this chapter, you will be able to implement event and task listeners in your business processes.

What you need for this book

You will need the following software to be installed before running the code examples:

- Activiti requires JDK 6 or higher version. JDK 6 can be downloaded from `http://www.oracle.com/technetwork/java/javase/downloads/jdk6downloads-1902814.html`.
- There are installation instructions on this page as well. To verify that your installation was successful, run the Java version on the command line.
- Download `Activiti-5.13.zip` from `http://activiti.org/download.html`. Just unzip it in a directory of your choice and then take a look at the `readme.html` file that includes pointers to the documents and the release notes. The user guide (`docs/userguide/index.html`) includes instructions on how to get started in a simple and quick manner.
- Download the Eclipse Juno or Indigo from `http://www.eclipse.org/downloads/`.
- Unzip the downloaded file and then you will be able to use Eclipse. It is used to configure with the Activiti Designer.

Who this book is for

This book is primarily intended for Business Analysts (BAs), developers, and project managers who need to develop a process model for implementation in a BPM system. The book assumes that you have basic knowledge of business analysis and Java; however, knowledge of Activiti is not required.

Conventions

In this book, you will find a number of styles of text that distinguish between different kinds of information. Here are some examples of these styles, and an explanation of their meaning.

Code words in text are shown as follows: "The `Wars` directory contains `activiti-explorer.war` and `activiti-rest.war`."

To give clear instructions of how to complete a procedure or task, we use:

Time for action — heading

Instructions often need some extra explanation so that they make sense, so they are followed with:

What just happened?

This heading explains the working of tasks or instructions that you have just completed.

You will also find some other learning aids in the book, including:

Have a go hero — heading

These set practical challenges and give you ideas for experimenting with what you have learned.

A block of code is set as follows:

```
public class ListTaskDemo {
public static void main(String ar[])
  {
  ProcessEngineprocessEngine = ProcessEngineConfiguration
      .createStandaloneProcessEngineConfiguration()
      .setJdbcDriver("com.mysql.jdbc.Driver")
      .setJdbcUrl("jdbc:mysql://localhost:3306/activiti_book")
      .setJdbcPassword("root").setJdbcUsername("root")
      .buildProcessEngine();
  TaskServicetaskService = processEngine.getTaskService();

List<Task> tasks = taskService.createTaskQuery().taskCandidateGroup("e
ngineering").list();

  for(Task task : tasks)
{
System.out.println("Task available for Engineering group: " + task.
getName());
}
  }
}
```

New terms and **important words** are shown in bold. Words that you see on the screen, in menus or dialog boxes for example, appear in the text like this: "Clicking on the **Next** button moves you to the next screen."

Warnings or important notes appear in a box like this.

Tips and tricks appear like this.

Reader feedback

Feedback from our readers is always welcome. Let us know what you think about this book—what you liked or may have disliked. Reader feedback is important for us to develop titles that you really get the most out of.

To send us general feedback, simply send an e-mail to `feedback@packtpub.com`, and mention the book title via the subject of your message.

If there is a topic that you have expertise in and you are interested in either writing or contributing to a book, see our author guide on `www.packtpub.com/authors`.

Customer support

Now that you are the proud owner of a Packt book, we have a number of things to help you to get the most from your purchase.

Downloading the example code

You can download the example code files for all Packt books you have purchased from your account at `http://www.packtpub.com`. If you purchased this book elsewhere, you can visit `http://www.packtpub.com/support` and register to have the files e-mailed directly to you.

Downloading the color images of this book

We also provide you a PDF file that has color images of the screenshots/diagrams used in this book. The color images will help you better understand the changes in the output. You can download this file from: `http://www.packtpub.com/sites/default/files/downloads/7065OS_ColorImages.pdf`

Errata

Although we have taken every care to ensure the accuracy of our content, mistakes do happen. If you find a mistake in one of our books—maybe a mistake in the text or the code—we would be grateful if you would report this to us. By doing so, you can save other readers from frustration and help us improve subsequent versions of this book. If you find any errata, please report them by visiting `http://www.packtpub.com/submit-errata`, selecting your book, clicking on the **erratasubmissionform** link, and entering the details of your errata. Once your errata are verified, your submission will be accepted and the errata will be uploaded on our website, or added to any list of existing errata, under the Errata section of that title. Any existing errata can be viewed by selecting your title from `http://www.packtpub.com/support`.

Piracy

Piracy of copyright material on the Internet is an ongoing problem across all media. At Packt, we take the protection of our copyright and licenses very seriously. If you come across any illegal copies of our works, in any form, on the Internet, please provide us with the location address or website name immediately so that we can pursue a remedy.

Please contact us at `copyright@packtpub.com` with a link to the suspected pirated material.

We appreciate your help in protecting our authors, and our ability to bring you valuable content.

Questions

You can contact us at `questions@packtpub.com` if you are having a problem with any aspect of the book, and we will do our best to address it.

1
Installing Activiti

The business process is important for any organization, but managing it is of equal importance. Business requirements frequently change for an organization. To change, manage, and understand the current business process is very hard without **Business Process Management** *(***BPM***), where the execution of each system depends on the business process. For developers, implementing a business process without following any standards leads to complexity. To explain a business process to a non-technical person without any graphics leads to misunderstanding. In such situations,* **Activiti** *comes to the rescue, as it is one of the best BPM frameworks for implementing business processes in your organization. Activiti provides a platform to create a business process on a canvas, build it, and execute it on the fly. It is open source, written in Java, and distributed under the Apache V2 license.*

To implement a business process using Activiti, you first have to understand the skeleton of Activiti BPM. In this chapter, we will show you how to download and install Activiti.

In this chapter, we will cover the following topics:

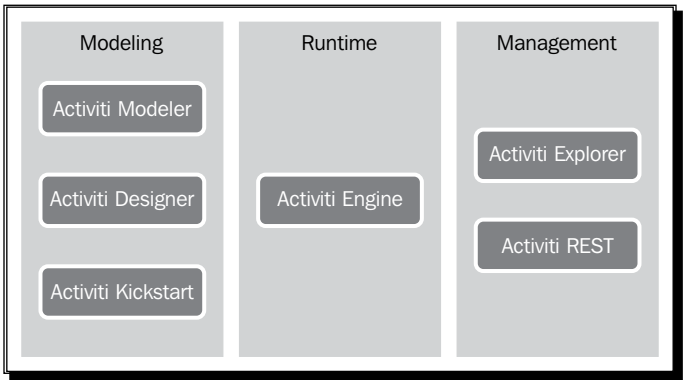
- Overview of Activiti BPM
- Downloading Activiti BPM
- Installing the Activiti framework
- Building your first business process

By the end of this chapter, you will be able to install and access the Activiti framework and implement your own business process.

So, let's take a tour of Activiti, set it up, and implement a business process.

Getting started with Activiti BPM

Let's take a quick tour of the Activiti components so you can get an idea of what the core modules are in the Activiti BPM that make it a lightweight and solid framework. You can refer to the following figure for an overview of the Activiti modules:



In this figure, you can see that Activiti is divided into various modules. **Activiti Modeler**, **Activiti Designer**, and **Activiti Kickstart** are part of **Modelling**, and they are used to design your business process. **Activiti Engine** can be integrated with your application, and is placed at its center as a part of **Runtime**. To the right of **Runtime**, there are **Activiti Explorer** and **Activiti Rest**, which are part of **Management** and used in handling business processes.

Let's see each component briefly to get an idea about it.

The Activiti Engine

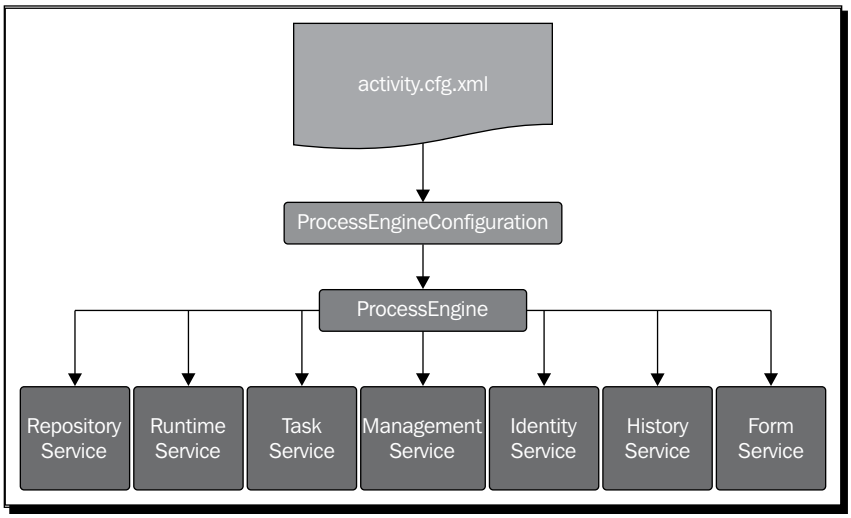
The Activiti Engine is a framework that is responsible for deploying the process definitions, starting the business process instance, and executing the tasks.

The following are the important features of the Activiti Engine:

- Performs various tasks of a process engine
- Runs a BPMN 2 standard process
- It can be configured with JTA and Spring
- Easy to integrate with other technology
- Rock-solid engine
- Execution is very fast
- Easy to query history information
- Provides support for asynchronous execution

- It can be built with cloud for scalability
- Ability to test the process execution
- Provides support for event listeners, which can be used to add custom logic to the business process
- Using Activiti Engine APIs or the REST API, you can configure a process engine
- Workflow execution using services

You can interact with Activiti using various available services. With the help of process engine services, you can interact with workflows using the available APIs. Objects of process engines and services are threadsafe, so you can place a reference to one of them to represent a whole server.



In the preceding figure, you can see that the **Process Engine** is at the central point and can be instantiated using ProcessEngineConfiguration. The Process Engine provides the following services:

- **RepositoryService**: This service is responsible for storing and retrieving our business process from the repository
- **RuntimeService**: Using this service, we can start our business process and fetch information about a process that is in execution
- **TaskService**: This service specifies the operations needed to manage human (standalone) tasks, such as the claiming, completing, and assigning of tasks
- **IdentityService**: This service is useful for managing users, groups, and the relationships between them

- **ManagementService**: This service exposes engine, admin, and maintenance operations, which have no relation to the runtime execution of business processes
- **HistoryService**: This service provides services for getting information about ongoing and past process instances
- **FormService**: This service provides access to form data and renders forms for starting new process instances and completing tasks

The Activiti Modeler

The Activiti Modeler is an open source modeling tool provided by the KIS BPM process solution. Using the Activiti Modeler, you can manage your Activity Server and the deployments of business processes. It's a web-based tool for managing your Activiti projects. It also provides a web form editor, which helps you to design forms, make changes, and design business processes easily.

The Activiti Designer

The Activiti Designer is used to add technical details to an imported business process model or the process created using the Activiti Modeler, which is only used to design business process workflows. The Activiti Designer can be used to graphically model, test, and deploy BPMN 2.0 processes. It also provides a feature to design processes, just as the Activiti Modeler does. It is mainly used by developers to add technical detail to business processes. The Activiti Designer is an IDE that can only be integrated with the Eclipse plugin.

The Activiti Explorer

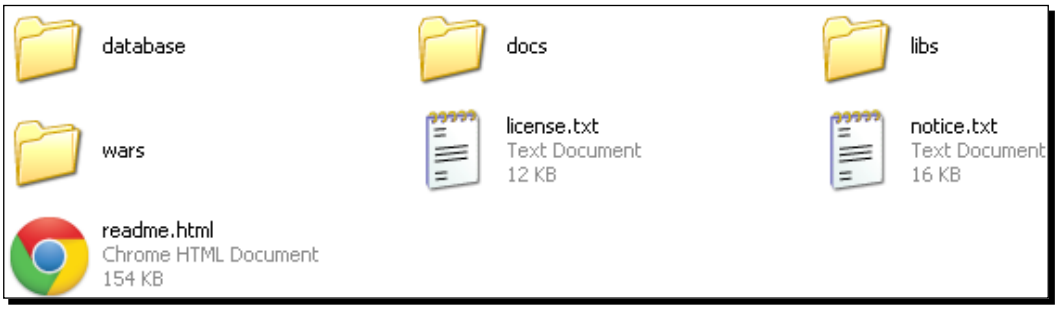
The Activiti Explorer is a web-based application that can be easily accessed by a non-technical person who can then run that business process. Apart from running the business process, it also provides an interface for process-instance management, task management, and user management, and also allows you to deploy business processes and to generate reports based on historical data.

The Activiti REST

The Activiti REST provides a REST API to access the Activiti Engine. To access the Activiti REST API, we need to deploy `activiti-rest.war` to a servlet container, such as Apache Tomcat. You can configure Activiti in your own web application using the Activiti REST API. It uses the JSON format and is built upon Restlet. Activiti also provides a Java API. If you don't want to use the REST API, you can use the Java API.

Downloading and installing Activiti BPM

To start with Activiti, we will first download it. To download Activiti, just go through its website at `http://activiti.org/download.html` and download the latest version, `activiti-5.x.zip`, into your local system. Now extract that ZIP file and you will get all the directories for all the operating systems (the file structures will be the same). You will get the list of directories and files as shown in the following screenshot:



In the `database` directory, you will find the entire SQL scripting file. The `wars` directory contains `activiti-explorer.war` and `activiti-rest.war`. You can also find Java documents and the user guide in the `docs` directory. The `libs` directory contains all the Activiti JAR files.

Prerequisites for Activiti

Before starting the installation of Activiti, there are some prerequisite software applications that should be installed on your system.

JDK 6 or higher

Activiti runs on a version of JDK higher than or equal to Version 6. Go to `http://www.oracle.com/technetwork/java/javase/downloads/index.html` and click on the **Download JDK** button. There are installation instructions on this page. To verify that your installation was successful, run the following command in the command prompt:

```
java -version
```

Execution of this command should print the installed version of your JDK.

Eclipse Juno or Indigo

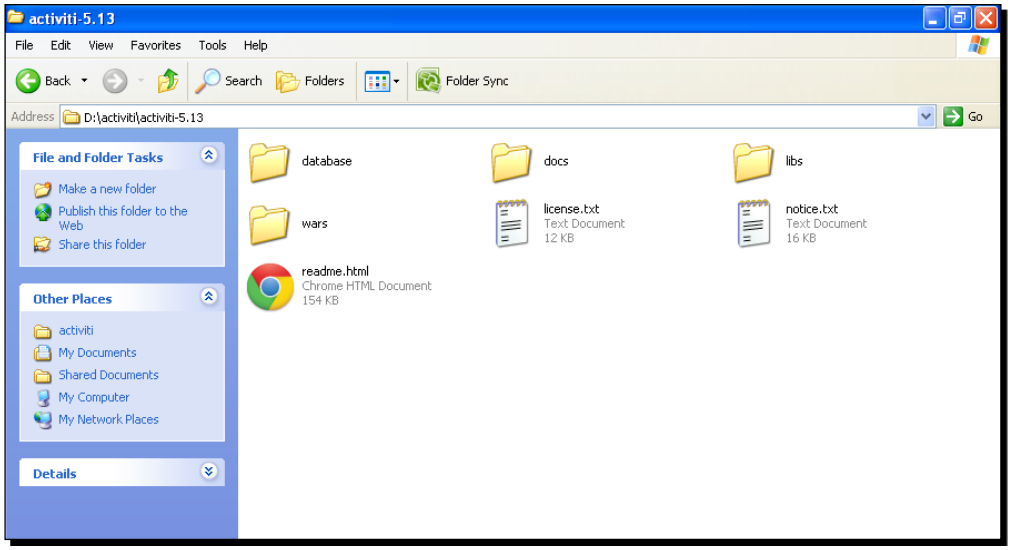
Eclipse will be required when you start working with Activiti Designer as the Activiti Designer only works with Indigo or Juno. Hence, download Eclipse Juno or Indigo from `http://www.eclipse.org/downloads/`, unzip the downloaded file, and you can start Eclipse from within the `Eclipse` folder.

Installing the Activiti framework

Now is the right time to configure Activiti in your system. But, before installing Activiti, make sure that the JDK is configured.

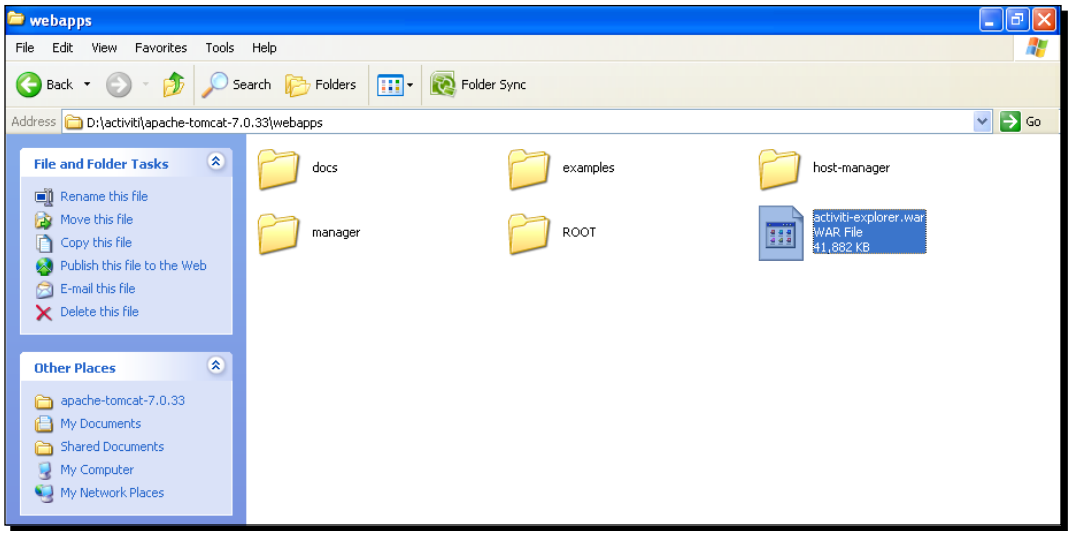
Time for action – installing the Activiti framework

We have already set up all the prerequisites on our system. Let us extract Activiti 5.13 onto one of our drives.



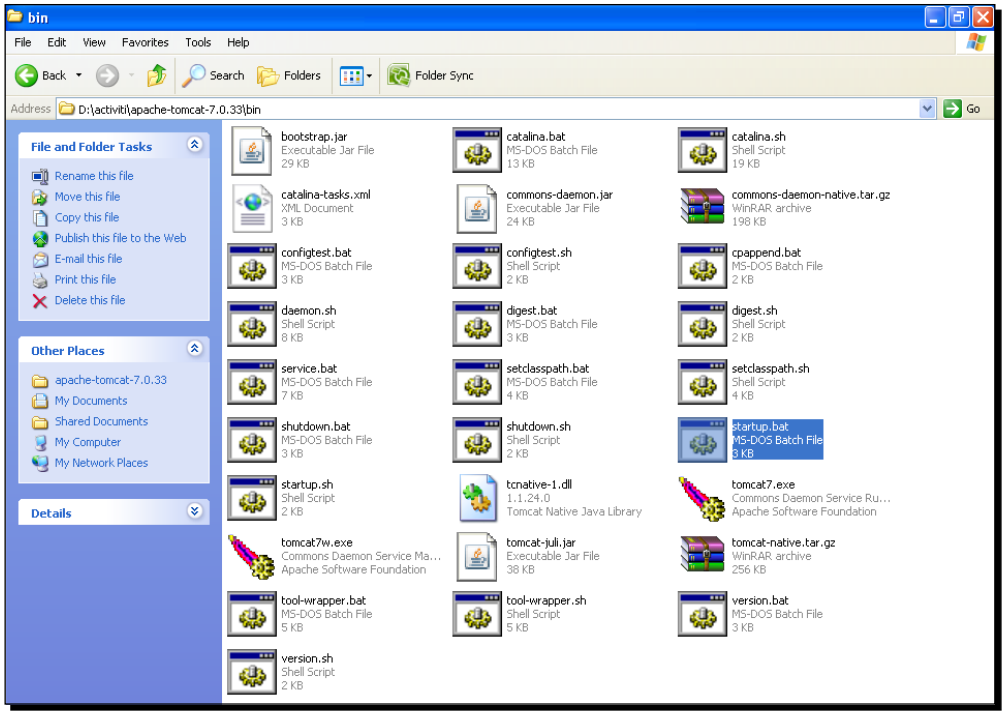
To install the Activiti framework on the system, we will use Apache Tomcat; perform the following steps:

1. Download the latest Apache Tomcat from `http://tomcat.apache.org/download-70.cgi` and extract it onto your local system.
2. Copy `activiti-explorer.war` from the `activiti-5.x\wars` folder and place it in the `apache-tomcat-7.0.33\webapps` folder, as shown in the following screenshot:

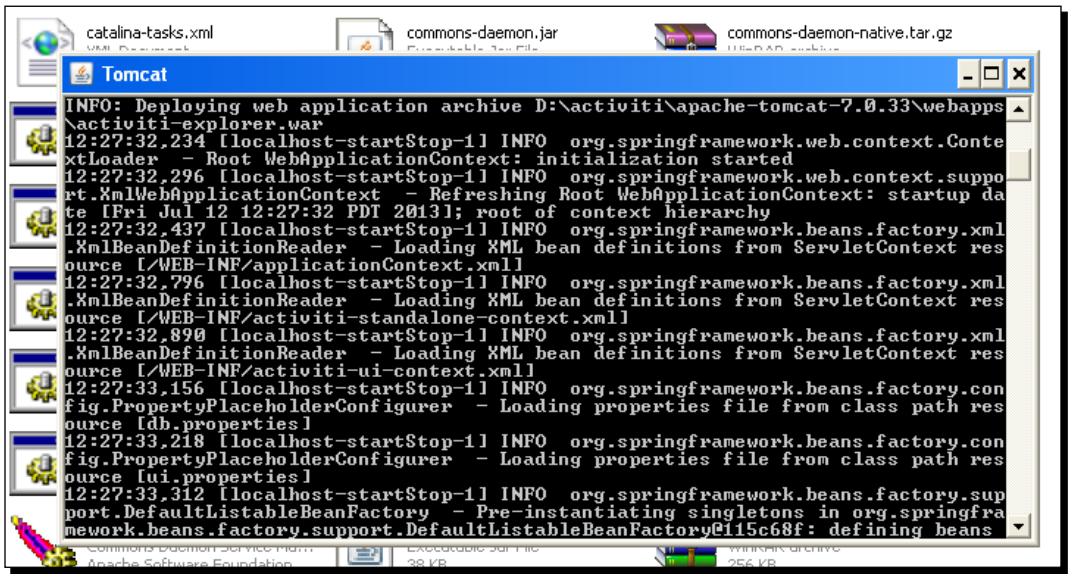


3. Now we will start the Apache Tomcat server so that the `activiti-explorer.war` file is deployed to the `webapps` folder.

4. For the Windows operating system, browse to `apache-tomcat-7.0.33\bin` and double-click on `startup.bat`, as shown in the following screenshot:



5. Now, the `startup.bat` file will start executing and deploying the Activiti Explorer on the system. During installation, it will set up the Activiti Engine, H2 database, default users, and much more. The result of this installation can be seen in the following screenshot:



6. All the directories of the Activiti Explorer will be created on the `apache-tomcat-7.0.33\webapps\activiti-explorer` path once the installation is completed.

7. For the Linux operating system, we can use the command prompt to start Apache Tomcat.

8. The location of the `war` file will be the same as that for the Windows operating system. To start Apache Tomcat, we need to browse to the `../apache-tomcat-7.0.33/bin` path using the command prompt and execute the following command:

```
sh catalina.sh run
```

The result of executing the previous command is shown in the following screenshot:

Activiti is installed; but, to verify whether or not the installation was done properly, open your favorite browser and type the `localhost:8080/activiti-explorer` URL container and press *Enter*. If you get the Activiti login screen, it means you have set up Activiti successfully. If you face any problems, make sure that Tomcat has started and also check whether or not `Activiti-explorer.war` is deployed successfully.

What just happened?

We have successfully downloaded Activiti and installed it on our system. We also had a look at the prerequisites that should be available before installing Activiti.

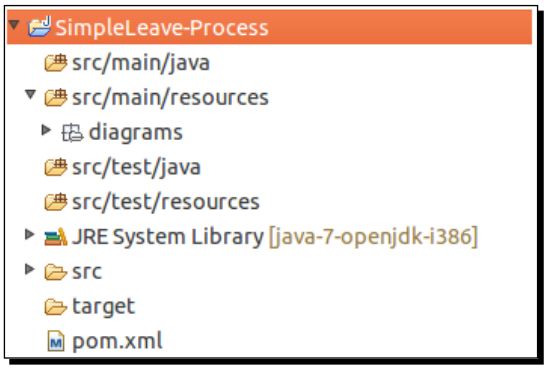
Time for action – creating our first process in Activiti

Now is the right time to look at one simple process in Activiti. We will be using the Eclipse IDE for designing the business process (the details of configuration are covered in *Chapter 3*, *Designing Your Process Using the Activiti Designer*). For now, we will just gain an understanding about creating a process.

We will create a simple leave process for employees in which we will have a script task and a user task. A script task is used for adding some logic within a process and a user task is used to indicate that there is some human interaction required in the process (there is a detailed explanation regarding tasks in *Chapter 2*, *Modeling Using the Activiti Modeler*).

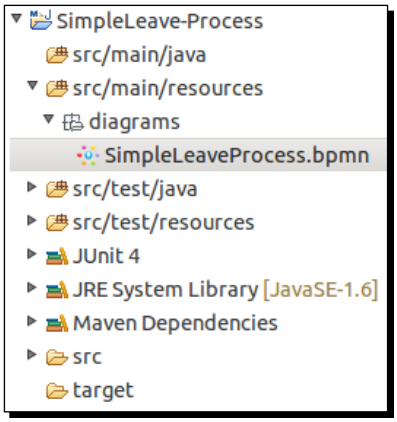
Perform the following steps to create a process in Activiti:

1. Create a new Activiti project in Eclipse by navigating to **File** | **New** | **Project** | **Activiti Project** and name your project `SimpleLeave-Process`.
2. The `SimpleLeave-Process` project will have a folder structure as shown in the following screenshot:



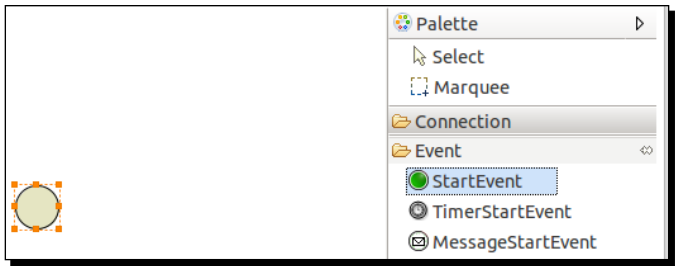
3. Now create an Activiti Diagram in the `src/main/rersource/diagrams` folder structure.

4. To create a diagram, navigate to **File** | **New** | **Other** | **Activiti Diagram** and name your diagram `SimpleLeaveProcess`. The folder structure is shown in the following screenshot:

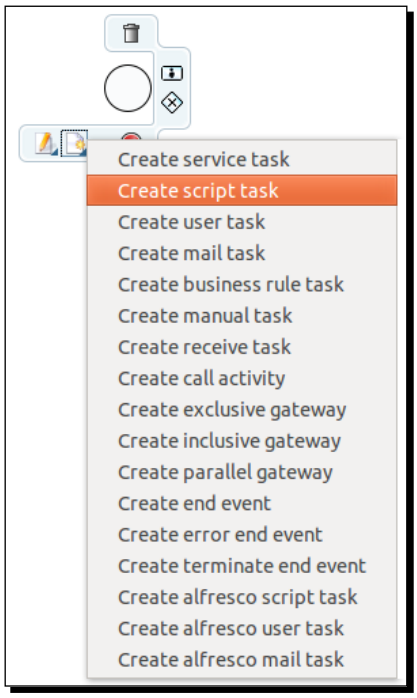


5. On opening the `.bpmn` file, a white canvas will appear on which you can design the business process.

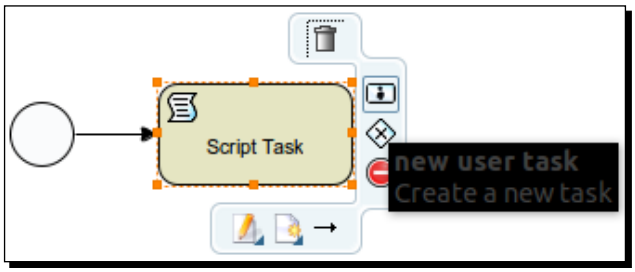
6. To draw a business process, there is a **Palette** option available which contains most of the BPMN diagrams for designing the business process. First, we will be adding a **StartEvent** on the canvas from the **Event** tab of the **Palette** option. All these options are shown in the following screenshot:



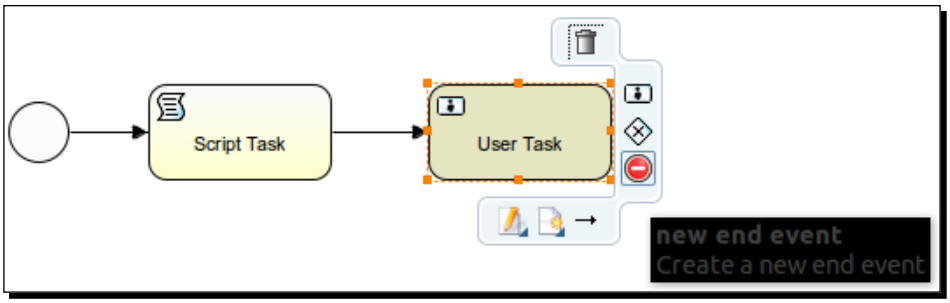
7. To add a script task, click on **StartEvent** and it will display the options as shown in the following screenshot. From those options, select the new element option, which will list down all the elements; from those elements, select **Create script task**, as shown in the following screenshot:



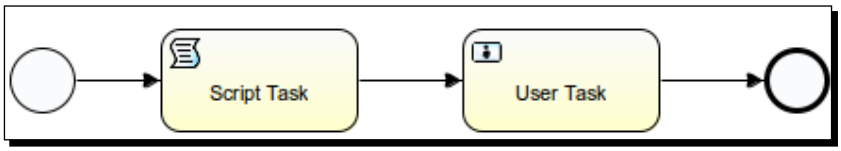
8. To add a user task, click on **Script Task** and select **new user task**, which is the option displayed with a human icon as shown in the following screenshot:



9. As each and every business process should have at least one end event, to add an end event, click on **User Task** and select the **new end event** option, as shown in the following screenshot:

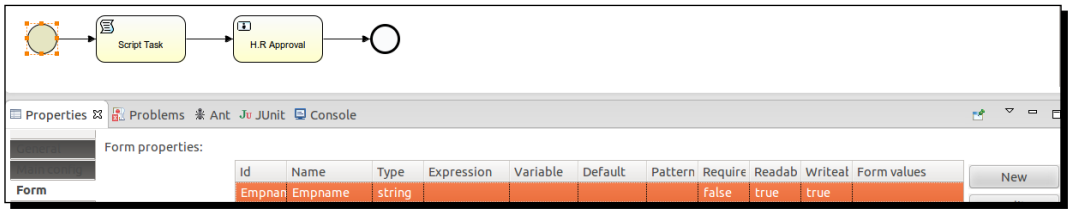


10. After adding the end event to the process, your business process should look similar to the following screenshot:

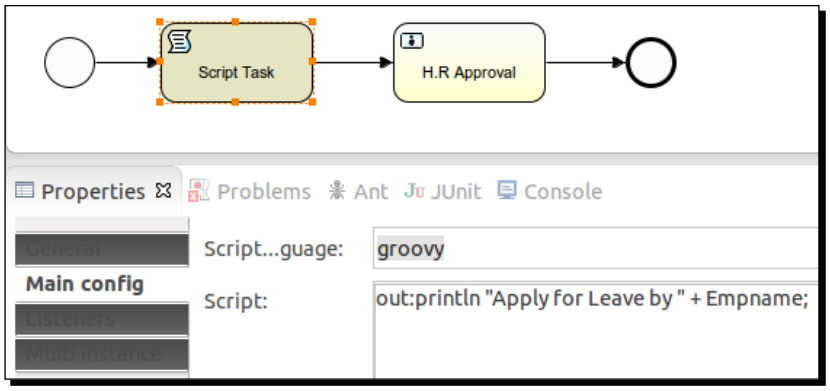


11. Now we have to make some changes to the properties of the process.

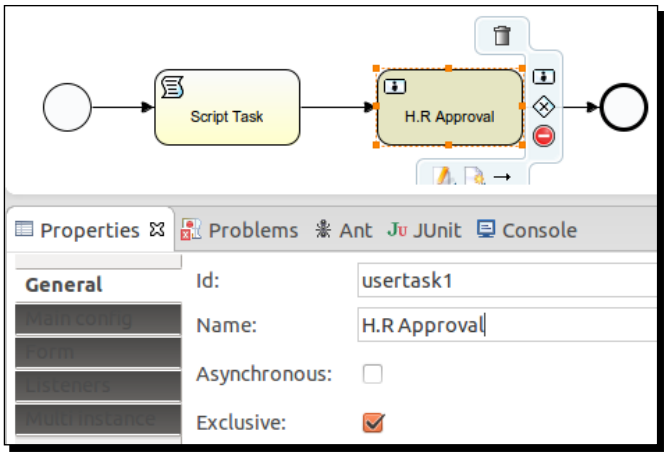
12. First, we will change the properties of the start event. Select the start event, open the **Properties** tab, and add **Form Properties** in the **Form** tab using the **New** button as shown in the following screenshot:



13. Now, edit the properties for the script task. In the **Main config** tab, there is a **Script** field available; within that, insert the print statement `out:println"ApplyforLeaveby"+Empname;`. The `Empname` property is the form property that we created in the start event. The script task will print the message on the console when it is executed, as shown in the following screenshot:

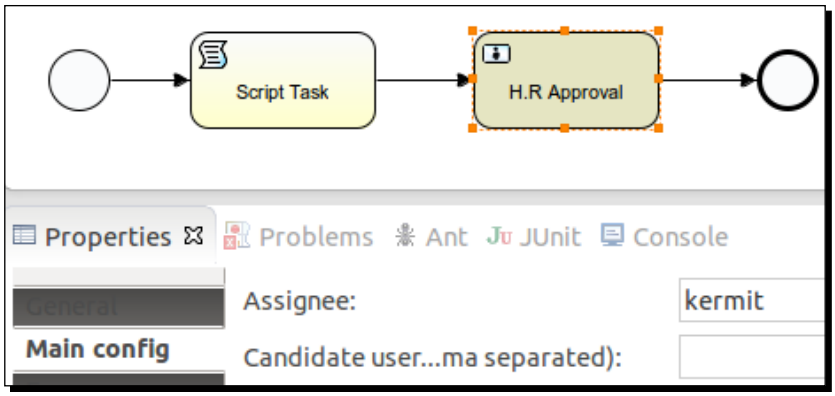


14. Now we will edit the user task properties. In the **General** tab, we will populate the **Name** property, which will be displayed in the user task node, as shown in the following screenshot:



15. As the user task is to be performed by a specific user, we have to provide the name of the user who will be responsible for executing it.

16. We have to populate the **Assignee** property of the **Main config** tab with the name of the person to whom the task should be assigned; as shown in the following screenshot, we will be assigning it to `kermit`:



17. We can also view the `.bpmn` file in the XML format. Right-click on the `SimpleLeaveProcess.bpmn` file, browse to **OpenWith** | **XML Editor**, and it will be available in the XML format. The result of this step is shown in the following screenshot:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xmlns:activiti="http:/
  <process id="leaveProcess" name="My process" isExecutable="true">
    <startEvent id="startevent1" name="Start">
      <extensionElements>
        <activiti:formProperty id="Empname" name="Empname" type="string"></activiti:formProperty>
      </extensionElements>
    </startEvent>
    <scriptTask id="scripttask1" name="Script Task" scriptFormat="groovy" activiti:autoStoreVariables="true">
      <script>out:println "Apply for Leave by " + Empname;</script>
    </scriptTask>
    <sequenceFlow id="flow1" sourceRef="startevent1" targetRef="scripttask1"></sequenceFlow>
    <userTask id="usertask1" name="H.R Approval" activiti:assignee="kermit">
      <documentation>Leave request by</documentation>
    </userTask>
    <sequenceFlow id="flow2" sourceRef="scripttask1" targetRef="usertask1"></sequenceFlow>
    <endEvent id="endevent1" name="End"></endEvent>
    <sequenceFlow id="flow3" sourceRef="usertask1" targetRef="endevent1"></sequenceFlow>
  </process>
  <bpmndi:BPMNDiagram id="BPMNDiagram_leaveProcess">
    <bpmndi:BPMNPlane bpmnElement="leaveProcess" id="BPMNPlane_leaveProcess">
      <bpmndi:BPMNShape bpmnElement="startevent1" id="BPMNShape_startevent1">
        <omgdc:Bounds height="35.0" width="35.0" x="20.0" y="120.0"></omgdc:Bounds>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape bpmnElement="scripttask1" id="BPMNShape_scripttask1">
        <omgdc:Bounds height="55.0" width="105.0" x="100.0" y="110.0"></omgdc:Bounds>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape bpmnElement="usertask1" id="BPMNShape_usertask1">
        <omgdc:Bounds height="55.0" width="105.0" x="260.0" y="110.0"></omgdc:Bounds>
      </bpmndi:BPMNShape>
      <bpmndi:BPMNShape bpmnElement="endevent1" id="BPMNShape_endevent1">
        <omgdc:Bounds height="35.0" width="35.0" x="420.0" y="120.0"></omgdc:Bounds>
      </bpmndi:BPMNShape>
```

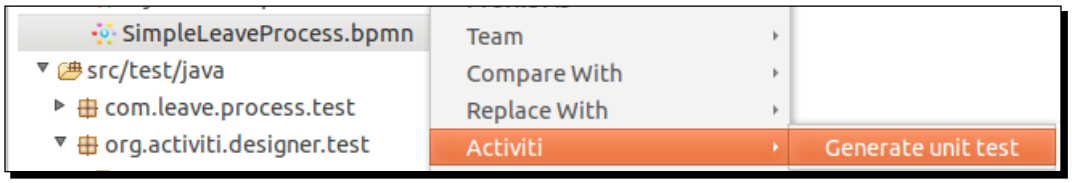
What just happened?

We have created a leave process using the Eclipse editor.

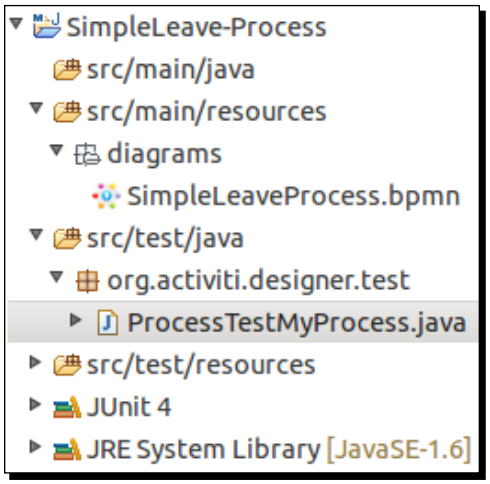
Time for action – unit testing to see the result

Now we will test the business process that we created. For this, we will have to perform the following steps:

1. Right-click on the `SimpleLeaveProcess.bpmn` file and browse to **Activiti** | **Generate unit test**, as shown in the following screenshot:



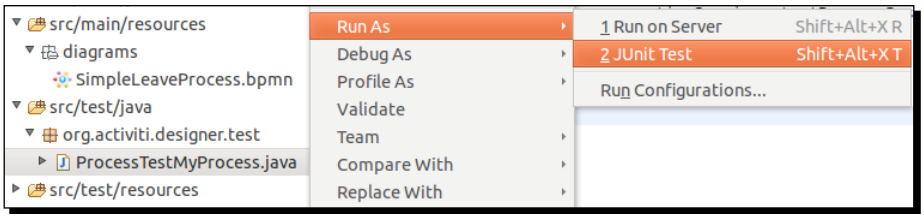
2. On performing the previous step, a test class file is created in `src/test/java package`, as shown in the following screenshot:



3. In the `ProcessTestMyProcess.java` file, replace the code with the following:

```
@Test
  public void startProcess() throws Exception {
    RepositoryService repositoryService = activitiRule.
getRepositoryService();
    repositoryService.createDeployment().addInputStream("SimpleLea
veProcess.bpmn20.xml",
      new FileInputStream(filename)).deploy();
    RuntimeService runtimeService = activitiRule.
getRuntimeService();
    Map<String, Object> variableMap = new HashMap<String,
Object>();
    variableMap.put("Empname", "Irshad");
    ProcessInstance processInstance = runtimeService.startProcessI
nstanceByKey("leaveProcess", variableMap);
    assertNotNull(processInstance.getId());
    System.out.println("id " + processInstance.getId() + " "
      + processInstance.getProcessDefinitionId());
  }
```

4. To view the test result, you can right-click on the `ProcessTestMyProcess` class file and browse to **Run As** | **JUnit Test**, as shown in the following screenshot:



5. The output after executing the test case is shown in the following screenshot:

```
29 Dec, 2013 9:33:20 PM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [activiti.cfg.xml]
29 Dec, 2013 9:33:22 PM org.activiti.engine.impl.db.DbSqlSession executeSchemaResource
INFO: performing create on engine with resource org/activiti/db/create/activiti.h2.create.engine.sql
29 Dec, 2013 9:33:22 PM org.activiti.engine.impl.db.DbSqlSession executeSchemaResource
INFO: performing create on history with resource org/activiti/db/create/activiti.h2.create.history.sql
29 Dec, 2013 9:33:22 PM org.activiti.engine.impl.db.DbSqlSession executeSchemaResource
INFO: performing create on identity with resource org/activiti/db/create/activiti.h2.create.identity.sql
29 Dec, 2013 9:33:22 PM org.activiti.engine.impl.ProcessEngineImpl <init>
INFO: ProcessEngine default created
29 Dec, 2013 9:33:23 PM org.activiti.engine.impl.bpmn.deployer.BpmnDeployer deploy
INFO: Processing resource SimpleLeaveProcess.bpmn20.xml
29 Dec, 2013 9:33:23 PM org.activiti.engine.impl.bpmn.parser.BpmnParse parseDefinitionsAttributes
INFO: XMLSchema currently not supported as typeLanguage
29 Dec, 2013 9:33:23 PM org.activiti.engine.impl.bpmn.parser.BpmnParse parseDefinitionsAttributes
INFO: XPath currently not supported as expressionLanguage
Apply for Leave by Irshad
id 5 leaveProcess:1:4
```

What just happened?

We have created a test file to test a business process in Activiti.

Summary

Oh great! We have finished *Chapter 1*, *Installing Activiti*. We have learned how to install Activiti on our system and how to use the Activiti Explorer for creating a simple business process. We also performed unit testing for a business process. Using the Activiti Explorer, we have deployed the employee leave process. In the next chapter, we will learn how to use the Activiti Modeler to design different business processes.

2
Modeling Using the Activiti Modeler

Business people are used to visualizing the business processes through graphical representations such as flowcharts and UML diagrams. However, when it comes to the execution of the business process, there is a technical gap between the design of a business process shown in the flowchart and the language that executes the business process, for example, WSBPEL.

In the previous chapter, we had an overview of Activiti, covering topics such as downloading and installing Activiti and creating a simple process within Activiti. In this chapter, we will understand what BPM is, its lifecycle and elements, and how to create a business process using the BPM elements in the Activiti Modeler.

In this chapter, we will cover the following topics:

- Business Process Modeling (BPM)
- Modeling principles and lifecycle
- BPM standards
- BPMN elements
- BPM using the Activiti Modeler
- Importing and exporting a model
- Deploying a model to the Activiti Engine

Understanding Business Process Modeling

BPM, with respect to system engineering, provides methods and techniques to make a business process more efficient, adaptive, and effective to accomplish business tasks. We will use the BPMN 2.0 modeling notations for designing our business process with the Activiti Engine. It provides a comprehensive approach to aligning an organization's business process as per the client's requirement. With the help of BPM, organizations can be more effective, efficient, and capable of accepting changes rather focusing on a traditional hierarchical management approach.

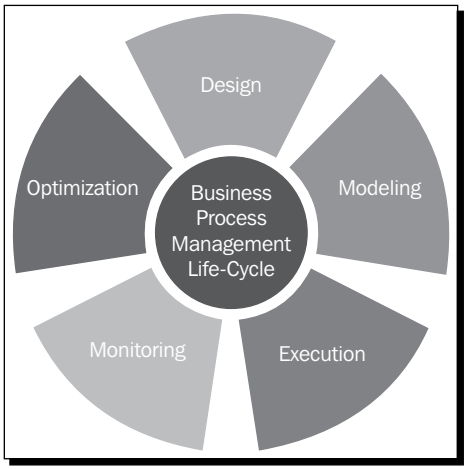
The BPM lifecycle

The creation of a business process comprises the five phases of business process management, commonly known as the BPM lifecycle. The five phases are as follows:

- Design
- Modeling
- Execution
- Monitoring
- Optimization

Each phase is designed to implement a successful process solution.

With the help of the BPM lifecycle, one can understand that implementing a business process is an ongoing process due to the changes occurring within the business environment. The five phases of the BPM lifecycle are as follows:



The design phase

The design phase of a lifecycle is responsible for the identification and design of a business process. The areas focused on within this phase are identifying activities, analyzing possible changes within the organization, defining service-level agreements, and specifying process details, such as actors, notifications, and escalations, within the business process. The main purpose of this phase is to ensure that a correct and efficient theoretical design is prepared. This phase belongs to the process owners who are deciding the process flow for the organization.

Modeling

Modeling is the second phase of the cycle; it's where the business process is validated and fully specified. In the design phase, the theoretical design is prepared, whereas in this phase, the theoretical business process is designed using various BPMN elements. This phase is mainly the responsibility of system analysts and integrators.

Execution

Once the theoretical design is transformed into a model, it can be implemented in the business process application. Now, it's time to automate the execution of the business process. This is the phase where the services are defined to automate the business process. The languages used to create the services are WSBPEL and BPMN 2.0. This phase belongs to the developers who are developing the business process.

Monitoring

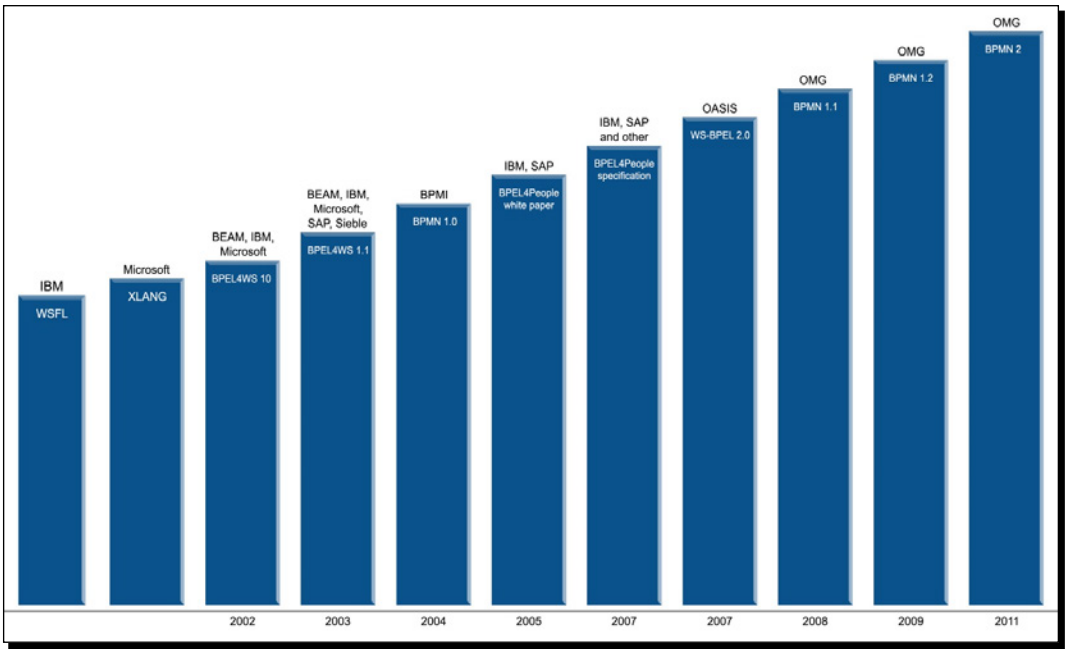
The monitoring phase keeps track of the individual processes. Performance monitoring for each and every process can be provided statistically, and the states of the process can also be maintained. In the monitoring phase, problems within a business process can be identified and corrective steps can be taken to improve it.

Optimization

The optimization phase is the last phase of the lifecycle. In this phase, the performance information of the process is fetched from the monitoring phase and improvements within the process and changes in the business requirement are identified. Once the optimization phase is completed, the business process again goes into the design phase and the cycle is completed.

Understanding the BPM standards

Various BPM standards have been introduced for process development, but all of them have certain limitations. The latest version in use is the BPMN 2.0 standard. This standard supports both designing as well as programming language functionalities. In the previous standards, this functionality was missing. When the business analyst would design a business process, it was difficult for the developers to add certain technical details to it, but the BPMN 2.0 standard made their lives easy. In the following figure, we will be looking at how the BPMN 2.0 standard has made life easier:



Getting started with BPMN elements

The simple diagrams that BPMN consists of to represent the business workflow prove to be helpful for both business users and developers. The four basic categories of BPMN elements are as follows:

- Flow objects
- Connecting objects
- Swim lanes
- Artifacts

Flow objects

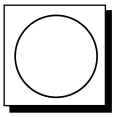
Flow objects consist of various shapes that represent the core elements of business process representation, which characterize the behavior of a business process. To represent the flow of the object, three elements are used: events, activities, and gateways.

Events

Events can be considered as the actions that occur during the business process. They are represented using a circle in a business process diagram. An intermediate event is represented using concentric circles. There are three types of events: start, intermediate, and end.

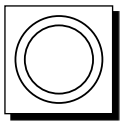
The start event

The start event indicates the start of your business process. It is represented using a single circle. In your business process, you should have only one start event. The start event can be represented as follows:



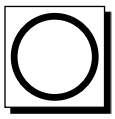
Intermediate event

Intermediate event refers to the events taking place between the start and end of the business process. These events do not affect the business process as they do not start or terminate the business process. You can have *n* number of intermediate events within your business process. The intermediate events are represented using concentric circles, as follows:



The end event

The end event is represented using a thick-bordered circle as follows, and indicates the end of your business process; there should be at least one end event within your business process:

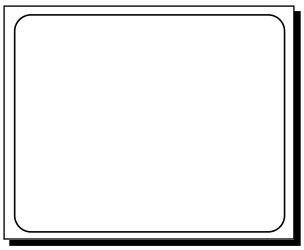


Activities

Activities represent the tasks or work that is being performed within a business process. The shape that represents the activities in a BPMN is a round-cornered rectangle. There are various types of activities, such as task, Sub-Process, and the call activity.

Task

A task activity is used in a business process when there is an activity to be implemented within a process. It is represented by a round-cornered rectangle. Tasks can be used when there are some actions to be performed in your business process. Various types of tasks are provided by BPMN, such as the script task, user task, service task, and mail task. A task is represented as follows:

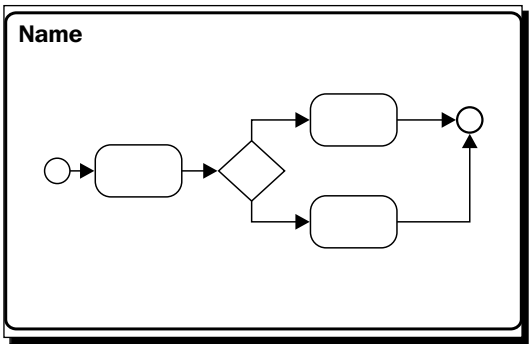


The following are the different types of tasks:

- **Script task**: This task is used for the automation of a task. We can have our logic implemented here.
- **User task**: This task can be used in a business process when human interaction is required for the business, for example, when details are to be filled or verified by a human.
- **Mail task**: This is a type of service task that has services implemented to send e-mails or notifications from the process.
- **Service task**: This refers to a custom task in which you want some specific operations to be done. Similar to the mail task, which is a type of service task that is used to send e-mails from the process, we have a custom service written for the process.

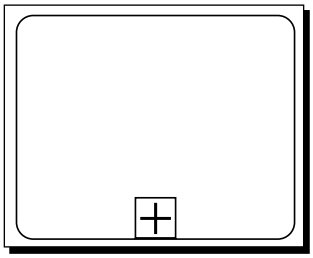
Sub-Process

A Sub-Process is a task that is used to represent levels within our business process. This task is useful when we want to display the leveled diagram of a business process. A Sub-Process will be represented as follows:



The call activity

With the help of the call activity, we can re-use a global process or task that is available within the process engine. On executing the call activity, the control of execution is transferred to the global process or task. A call activity is represented as follows:



Gateways

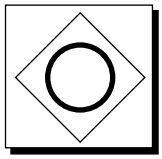
Gateways are used to handle the forking and joining of paths within the business process. The main aim of gateways is to control the flow of a business process. The diamond shape represents a gateway. The types of gateways available are exclusive gateways, event-based gateways, inclusive gateways, and parallel gateways.

Exclusive gateways

We can use an exclusive gateway when we want to proceed with one path from the multiple paths defined. So, we can compare the exclusive gateway to an `if-else` statement of the programming concept. An exclusive gateway can be represented as follows:

Inclusive gateways

An inclusive gateway can be used when we want to execute the paths for which the conditions specified for them are satisfied. We can compare the inclusive gateway to multiple `if` statements. An inclusive gateway is represented as follows:

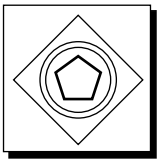


Parallel gateways

In a parallel gateway, all the outgoing paths will be executed without checking any conditions. Hence, in a parallel gateway, we need not specify any conditions for execution.

Event-based gateways

In this gateway, the paths will be executed based on the events that satisfy the conditions.

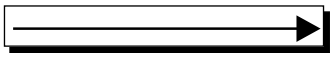


Connecting objects

To represent a basic framework of a business process structure, connecting objects are required. Each flow object can be connected using the connecting objects. These connecting objects can be represented using the following types: sequence flow, message flow, and association.

Sequence flow

Sequence flow represents the order of execution of activities in a business process. It is represented by a solid line with an arrow, as follows:

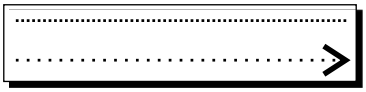


Message flow

Message flow is represented using a dashed line with an open arrow, as follows; it is used to show the messages that are being passed within the business process and cannot be used to connect the activities:

Associations

Associations are used to represent a relationship between the flow object and artifacts or data within the business process; it is graphically represented using a dotted line with an open arrowhead, as follows:

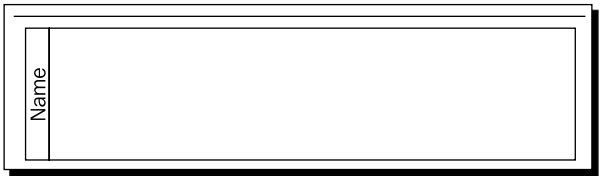


Swim lanes

Swim lanes are just used for visualization purposes. With the help of swim lanes, one can easily organize the activities of a business. Swim lanes are represented using two objects: the lane and the pool.

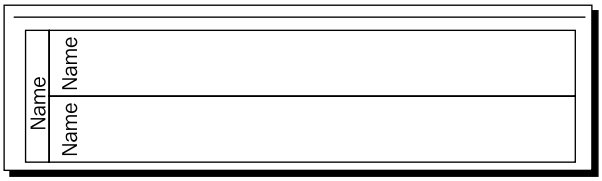
Pool

The pool represents an entity within a business process. It can consist of various lanes. When working with pools, we cannot connect to the activities outside the pool boundary. It is represented as follows:



Lanes

A lane is a subportion of a pool. Lanes are used to organize the business process as per the roles or functions being performed. They are represented as follows:

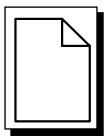


Artifacts

Artifacts are used to provide additional information in BPMN diagrams so that the use of the notation within the business process can be stated clearly and is understood by others. There are three types of artifacts: data object, group, and annotation.

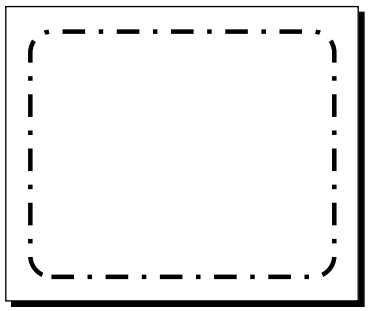
Data object

The data object artifact is useful for the person who is viewing the business process. With the help of this artifact, one can get an idea regarding the data required or produced within an activity. It is represented by an image of a piece of paper folded at the top-right corner, as follows:



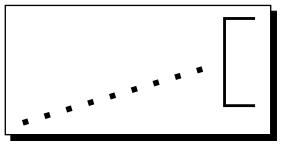
Group

With the help of the group artifact, one can clearly identify the activities or nodes that are grouped together. It is represented using a round-cornered rectangle with dashed lines, as follows:



Annotation

The annotation artifact is used to add some additional comments to the business process. It is represented as follows:



Installing the Activiti Modeler

So far, we have gained only a theoretical knowledge of Activiti. Now it's time to try some hands-on modeling and designing of a business process using the Activiti Modeler. So, to design a process, you will have to configure the Activiti Modeler, which we will be looking at in the upcoming sections.

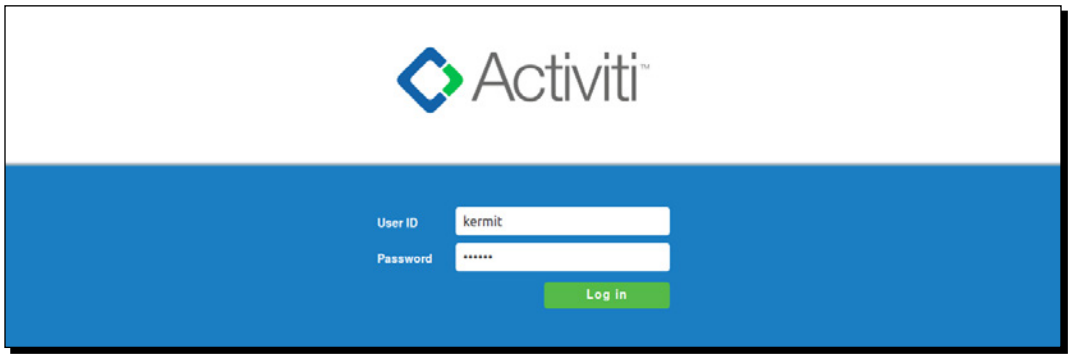
Time for action – Business Process Modeling using the Activiti Modeler

In the previous chapter, we created an employee leave process using the BPMN2.0 XML format. Now we will create a business process using the Activiti Modeler. It is a web-based BPM component available as a part of the Activiti Explorer web application. The main aim of the modeler is to support all the BPMN elements and extensions supported by the Activiti Engine. Perform the following steps to start modeling using the Activiti Modeler:

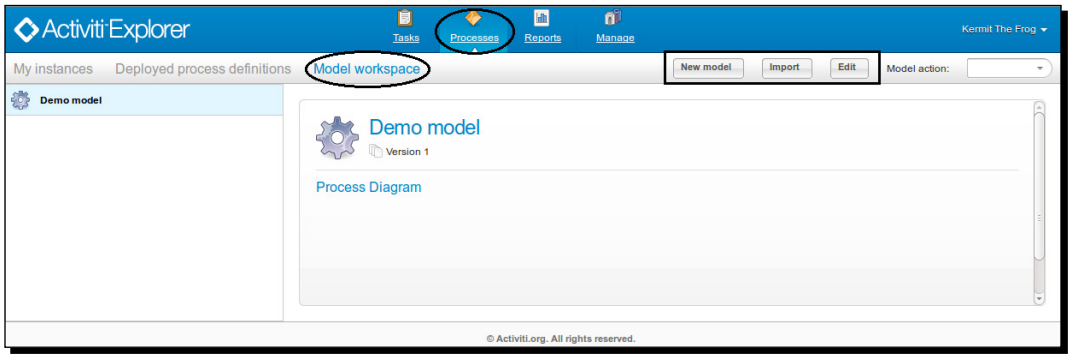
1. To start modeling a business process, you need to start your Activiti Engine if you haven't already.
2. If you are using a Windows operating system, you can browse to your `apache-tomcat` folder where you have installed the Activiti war file (`../apache-tomcat/bin/`) and execute the `startup.bat` file.
3. As we will be using a Linux system, we will start Apache Tomcat from the command prompt. Navigate to `../apache-tomcat/bin` from your terminal and execute the `sh catalina.sh run` command to run your Tomcat server. You will get an output similar to the following screenshot:

```
irshad@irshad:~/Desktop/Activiti-Engine/apache-tomcat-7.0.37/bin$ sh catalina.sh run
Using CATALINA_BASE:   /home/irshad/Desktop/Activiti-Engine/apache-tomcat-7.0.37
Using CATALINA_HOME:   /home/irshad/Desktop/Activiti-Engine/apache-tomcat-7.0.37
Using CATALINA_TMPDIR: /home/irshad/Desktop/Activiti-Engine/apache-tomcat-7.0.37/temp
Using JRE_HOME:        /usr/lib/jvm/java-7-openjdk-i386
Using CLASSPATH:       /home/irshad/Desktop/Activiti-Engine/apache-tomcat-7.0.37/bin/bootstrap.jar:/home/irshad/Desktop/Activiti-Engine/apache-t
omcat-7.0.37/bin/tomcat-juli.jar
Dec 19, 2013 5:22:41 PM org.apache.catalina.core.AprLifecycleListener init
INFO: The APR based Apache Tomcat Native library which allows optimal performance in production environments was not found on the java.library.p
ath: /usr/java/packages/lib/i386:/usr/lib/jni:/lib:/usr/lib
Dec 19, 2013 5:22:43 PM org.apache.coyote.AbstractProtocol init
INFO: Initializing ProtocolHandler ["http-bio-8080"]
Dec 19, 2013 5:22:43 PM org.apache.coyote.AbstractProtocol init
INFO: Initializing ProtocolHandler ["ajp-bio-8009"]
Dec 19, 2013 5:22:43 PM org.apache.catalina.startup.Catalina load
INFO: Initialization processed in 1999 ms
Dec 19, 2013 5:22:43 PM org.apache.catalina.core.StandardService startInternal
INFO: Starting service Catalina
```

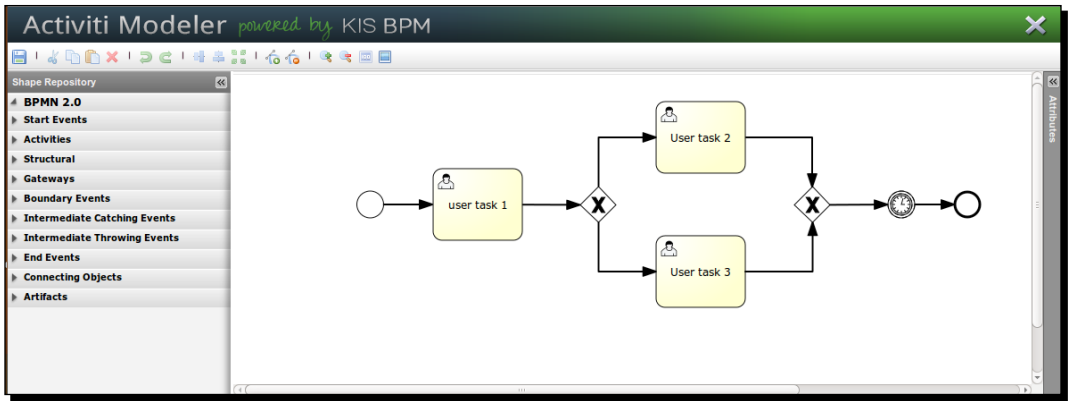
4. Once Apache Tomcat is up and running successfully, you can open a browser and browse to the **Activiti-Explorer** page using the `http://localhost:8080/activiti-explorer/` URL. This URL will open the Activiti Explorer login page. You have to provide the username as `kermit` and the password as `kermit` as well. The login page looks as follows:



5. On successful login, navigate to the **Processes** tab, as you have to model a process using the modeler. Go to the **Model workspace** section in the **Processes** tab. As you can see in the following screenshot, there are various buttons available at the top-right corner, which we will also have a look at:

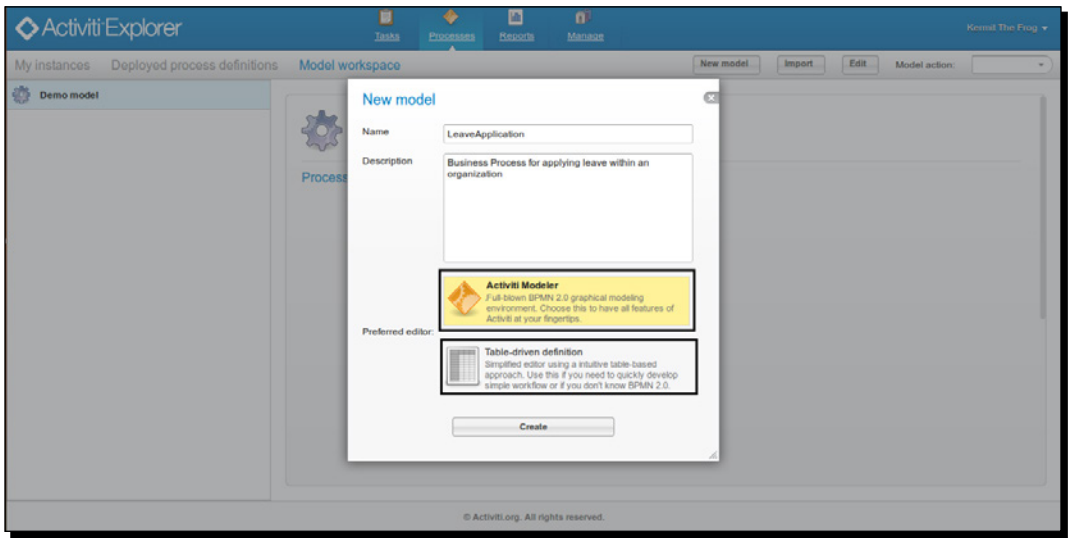


6. In the left pane of the window shown in the previous screenshot, you can see that there is a **Demo model** available for reference purposes. If you want to edit an existing process, you can select it from the left pane and click on the **Edit** button. On selecting the **Edit** button, the Activiti Modeler will open the selected process in the edit mode and you can edit the existing process as per the new requirement, as shown in the following screenshot:

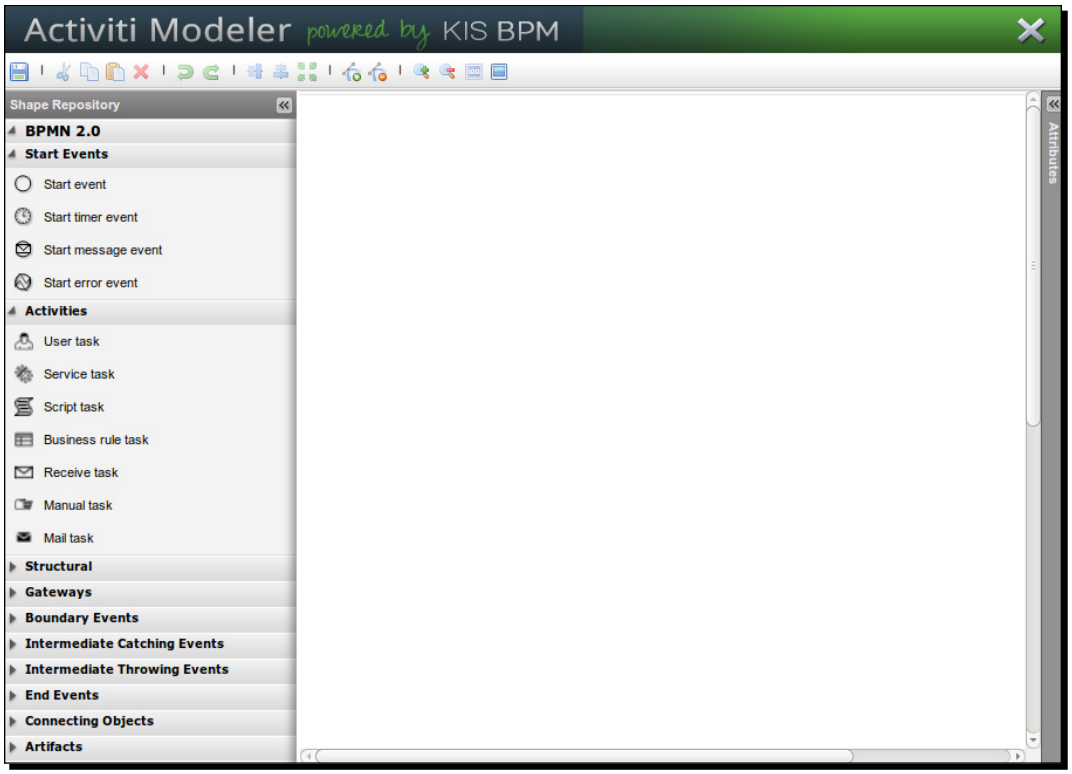


7. But, if you want to model a new process using the modeler, select the new model button from the menu bar shown in the next screenshot.

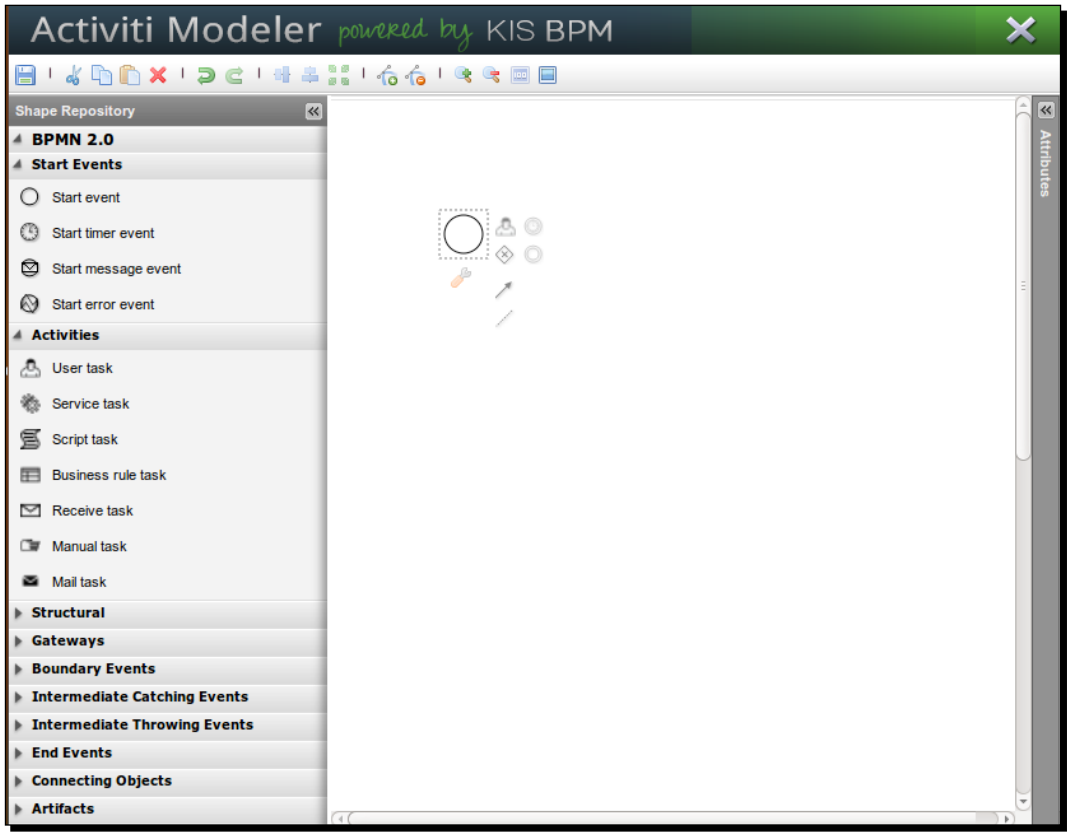
8. On selecting the new model button, a pop-up window opens in which you have to provide the details of your business process. Enter the name as `LeaveApplication` and the description as `Business process for applying leave within an organization` and select the **Activiti Modeler** option, as shown in the following screenshot:



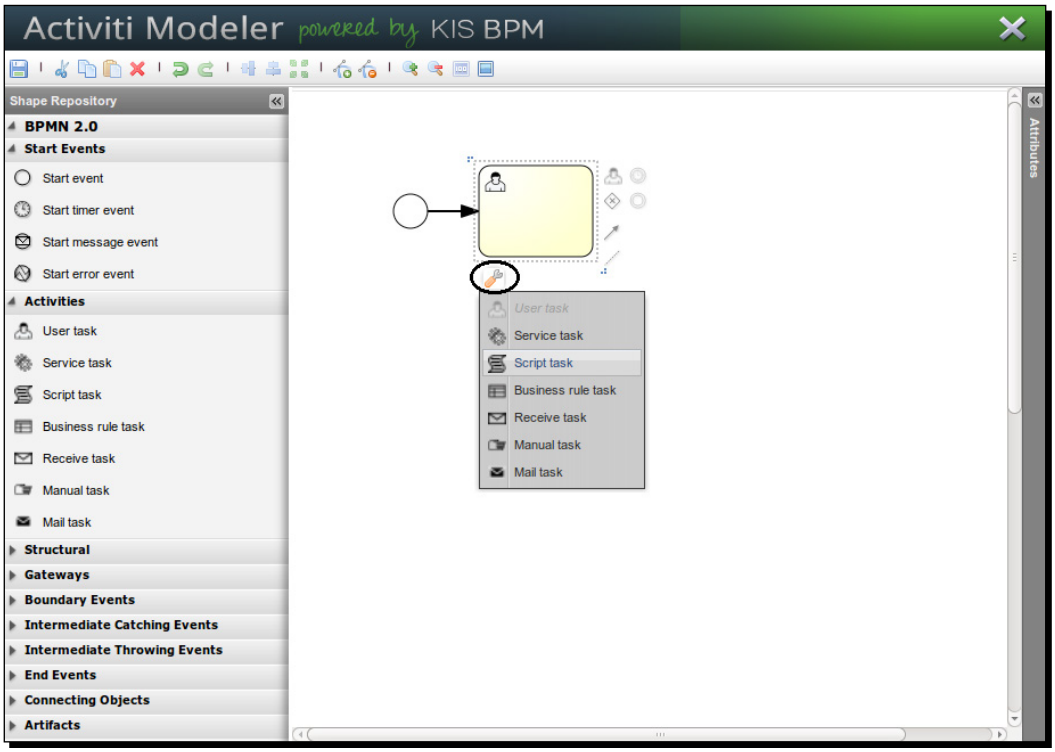
9. On creating the business process using the graphical editor, the Activiti Modeler opens with a blank canvas on which you can design your own new business process using the BPM Notations available in the left panel, as shown in the following screenshot:



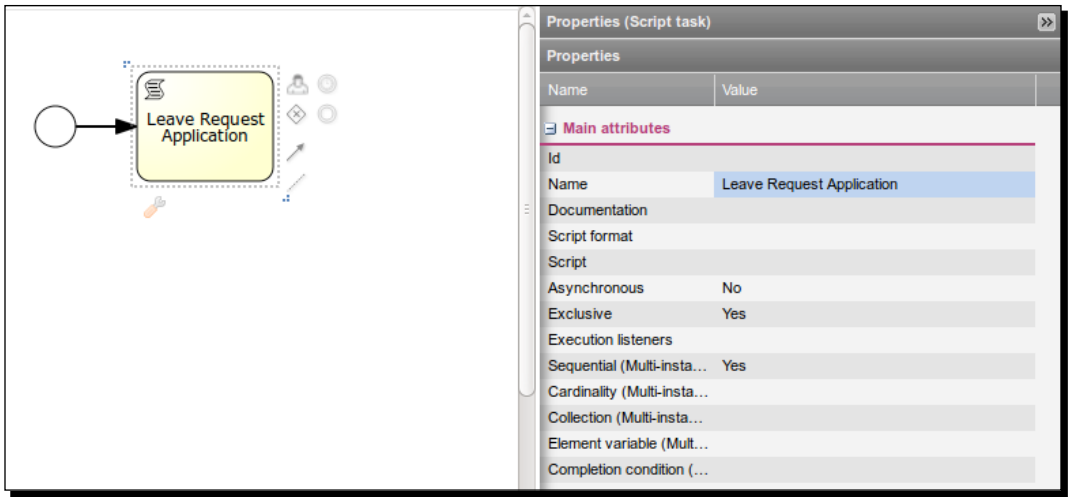
10. To start the process, we require a start node for the process. You can just drag the **Start event** node from the left pane to the white canvas. To add other nodes, you similarly drag them to the canvas. Alternatively, as you can see in the following screenshot, there are various icons available with the **Start event** node; using these icons, we can add the appropriate nodes as required:



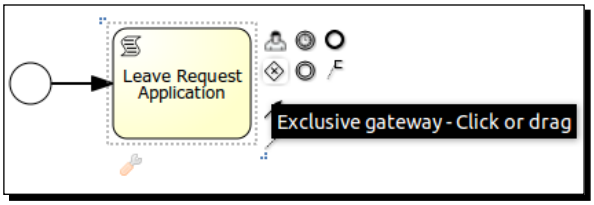
11. Now, to add a script task, you can either drag the **Script task** node from the left pane to the canvas or add a node by selecting the human icon from the **Start event** node and changing its type to script. On changing the type to script, the icon of the node will change from a human to a script. To change the type, you will have to select the **Transform shape** option available at the bottom-left corner of the node, as shown in the following screenshot:



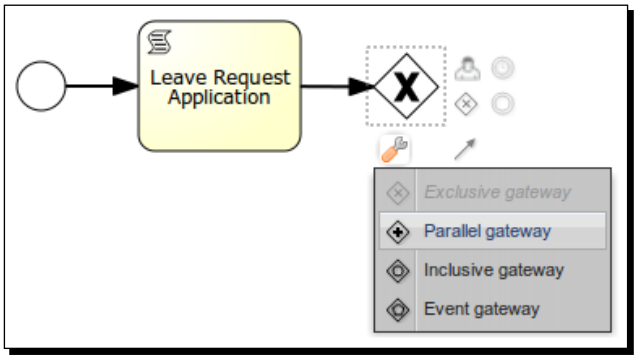
12. To change the name displayed on the node, we need to edit the properties of the node. The **Properties** pane will be available on the right-hand side of the modeler. As seen in the following screenshot, you can select the node and change the **Name** property to `Leave Request Application`:



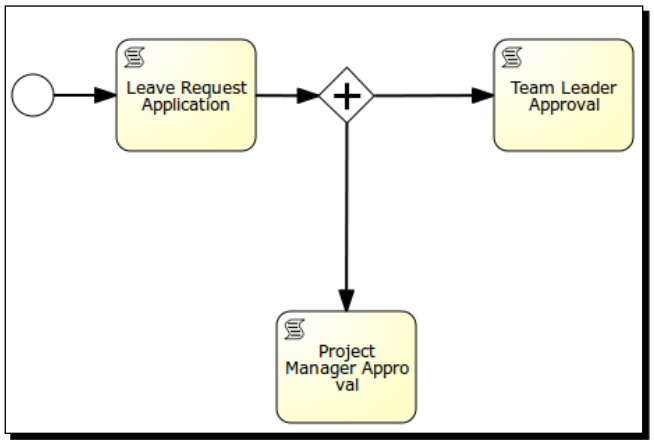
13. Now, we will split our business process by adding a gateway to it. In the following screenshot, you can see how we can split our business process by adding a gateway:



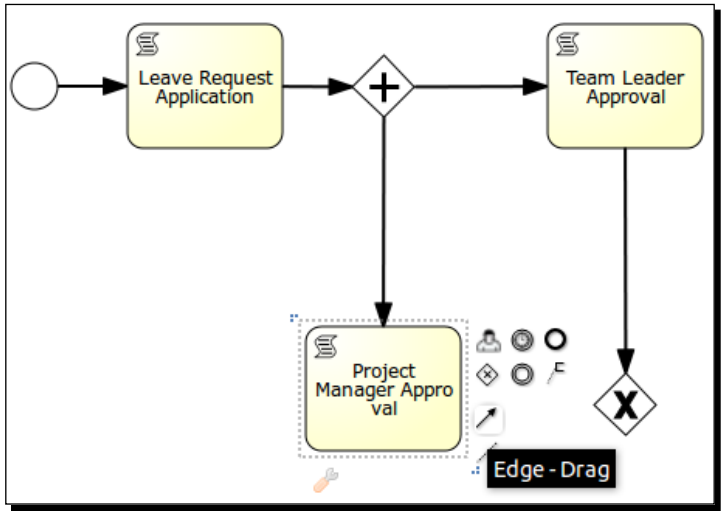
14. As we want to execute paths without checking their conditions, we will change the gateway to **Parallel gateway** from the default, which is **Exclusive gateway**.



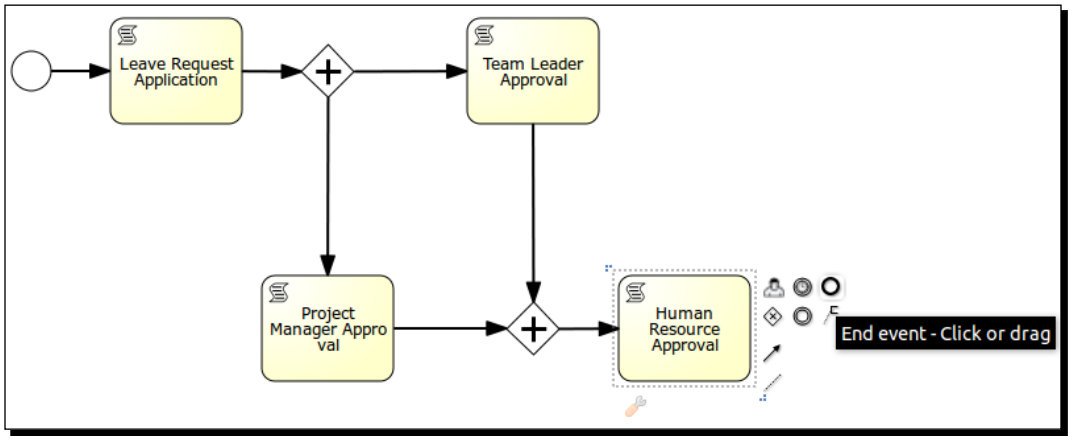
15. Now, we will have two script nodes on the parallel gateway waiting for approval from the team leader and project manager.



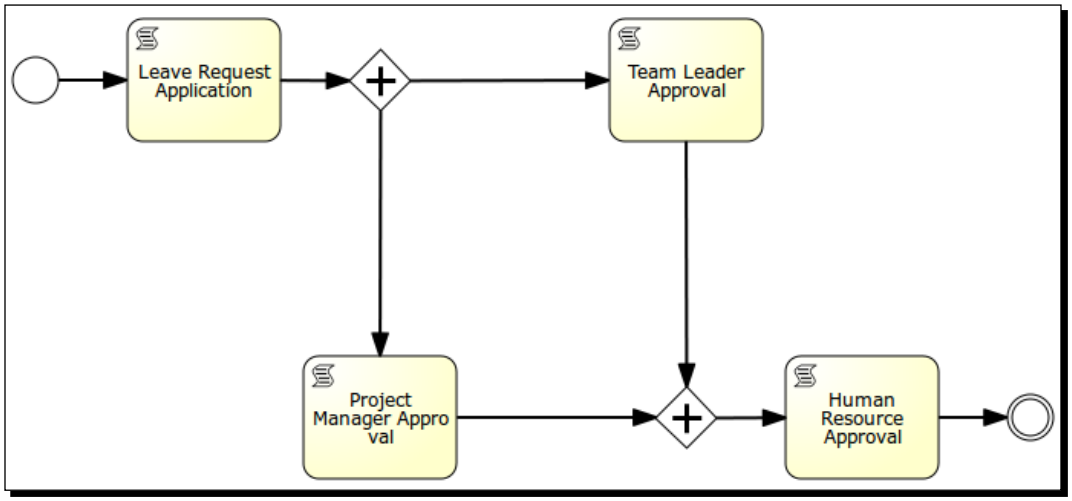
16. As we have used the parallel gateway to split our business process, we also have to use it for the purpose of joining the nodes. From the **Project Manager Approval** script node, we will drag an edge to the gateway. The result of this can be seen in the following screenshot:



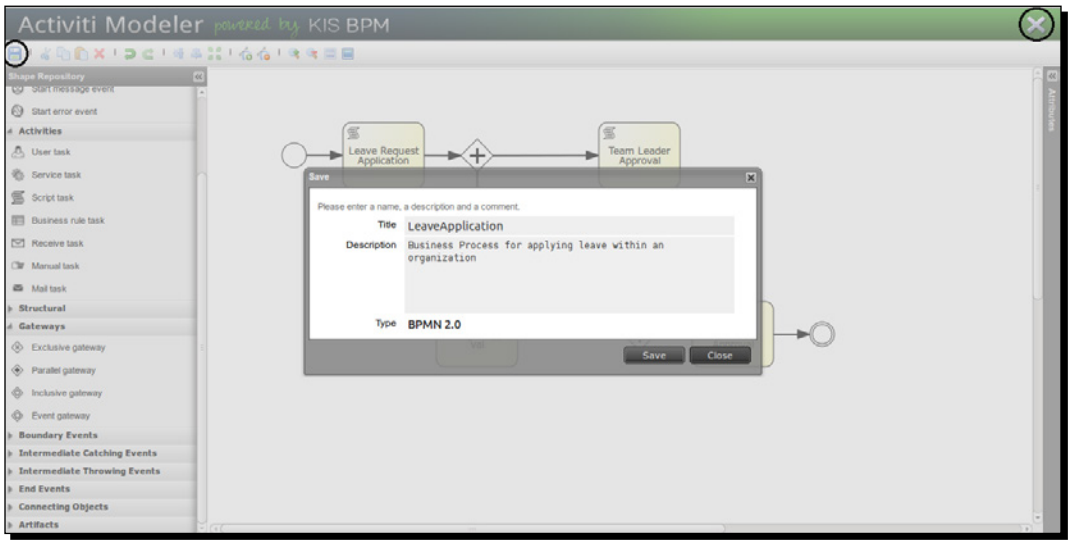
17. After the join gateway, we will add the **Human Resource Approval** script node, and after that script node, we need to add an end event to terminate the business process. The result of this can be seen in the following screenshot:



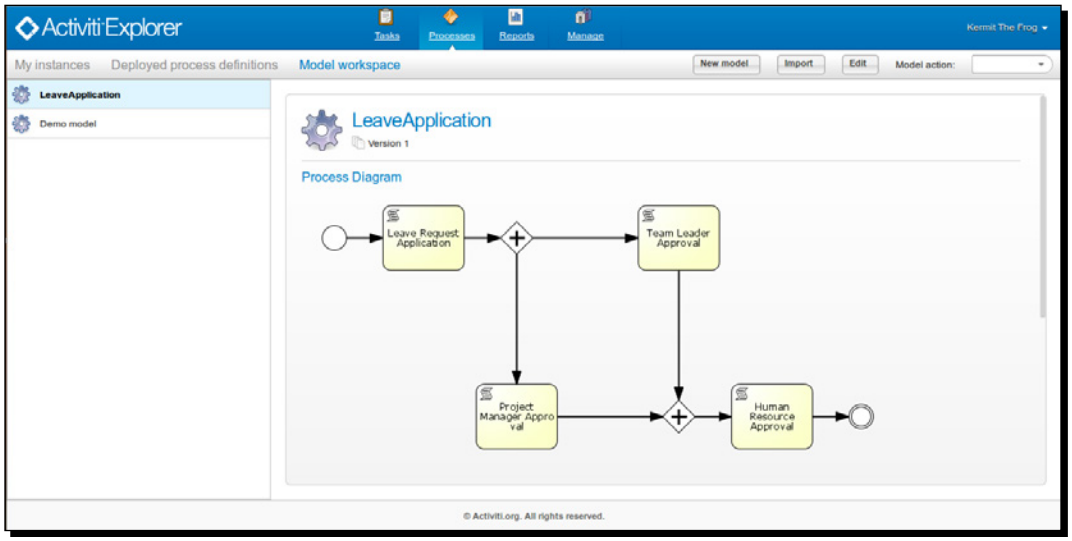
18. On completion of the business process, it will look as shown in the following screenshot:



19. To save the process in a modeler, there is a save icon available at the top-left corner of the modeler window, or you can even press the *Ctrl* + *S* keys. After saving the process, you can close the modeler window by clicking on the close icon at the top-right corner of the modeler window, as shown in the following screenshot:



20. On closing the modeler window, you will find your process available in the left pane along with the **Demo model,** as shown in the following screenshot:



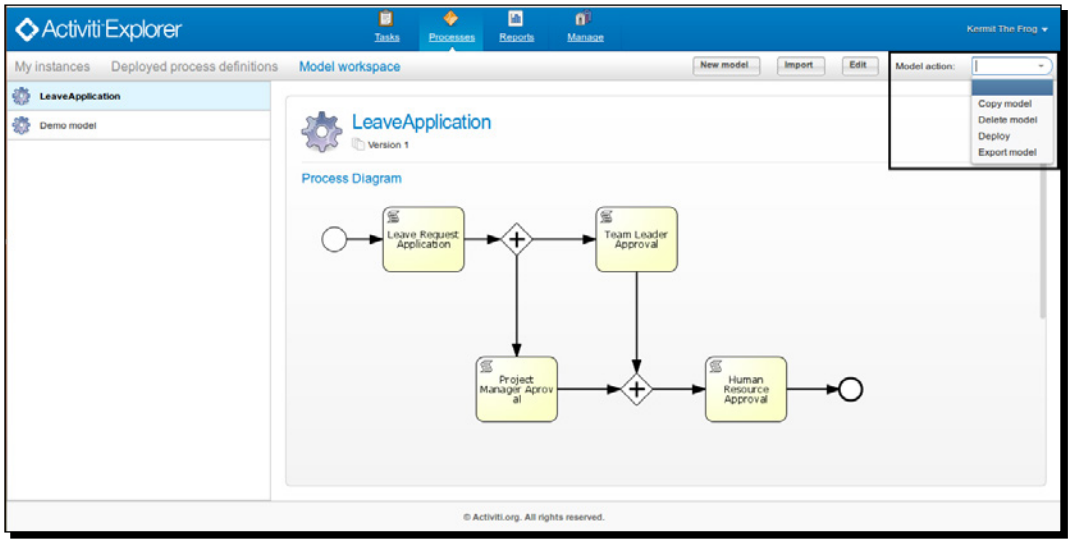
What just happened?

We had a look at the Activiti Modeler and learned how to create a process using it. We also learned how to use the various nodes available for process creation.

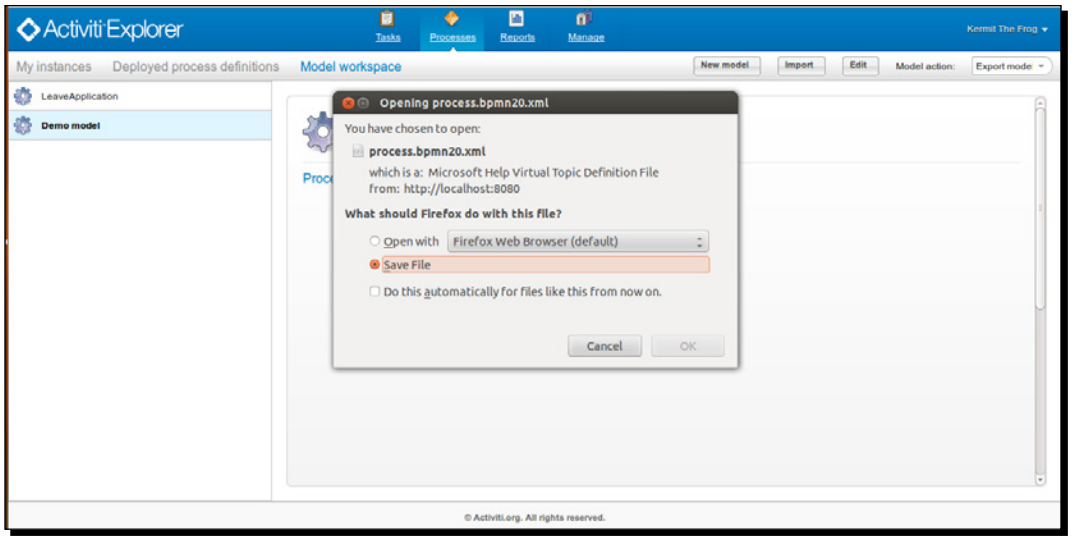
Time for action – import and export of a model

Now, as you are familiar with business process creation using the modeler, you are capable of designing a business process for presentation purposes. The process is approved and now has to be executed by the development team. For this, the modeler provides an export and import functionality so that you can either export or import an existing process from or to the modeler. Perform the following steps to import or export a model:

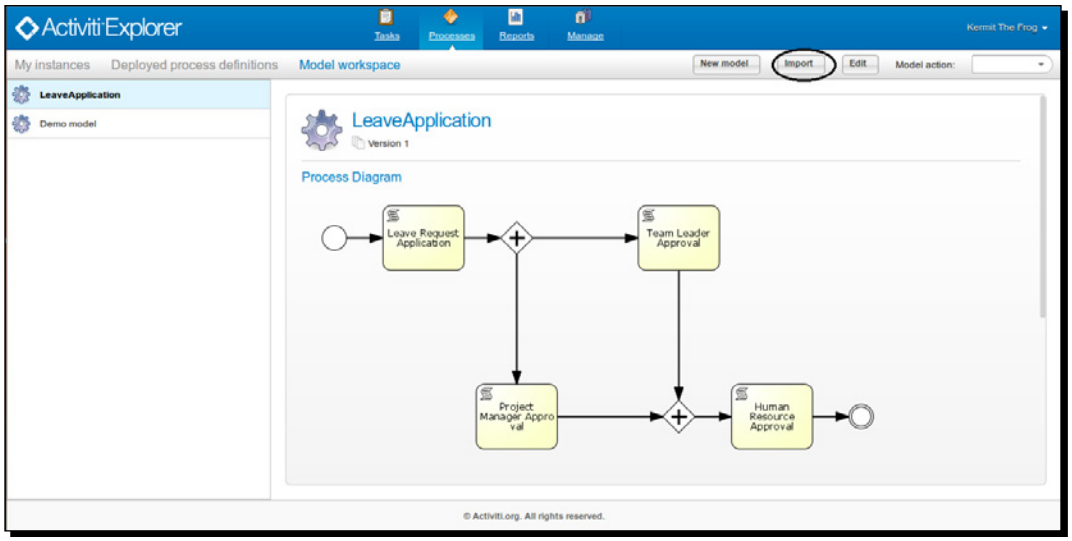
1. We will now export a business process. For performing an export, you can select the process to be exported, and from the **Model action** drop-down menu, you need to select the **Export model** option.



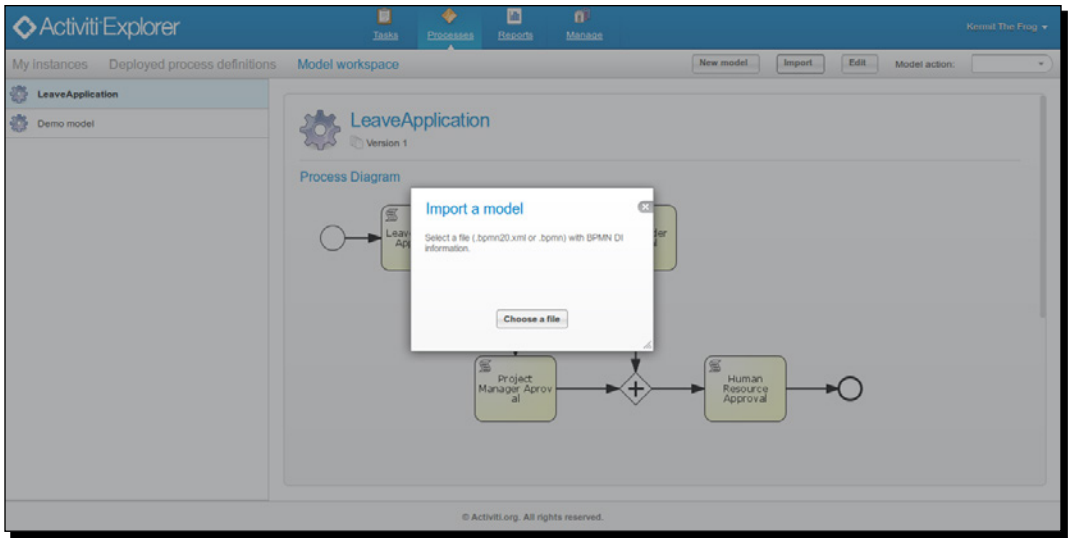
2. On selecting the **Export model** option, the modeler will provide a window that will save our business process in a `bpmn20.xml` format, and we can use the exported file in Eclipse and edit the process for execution:



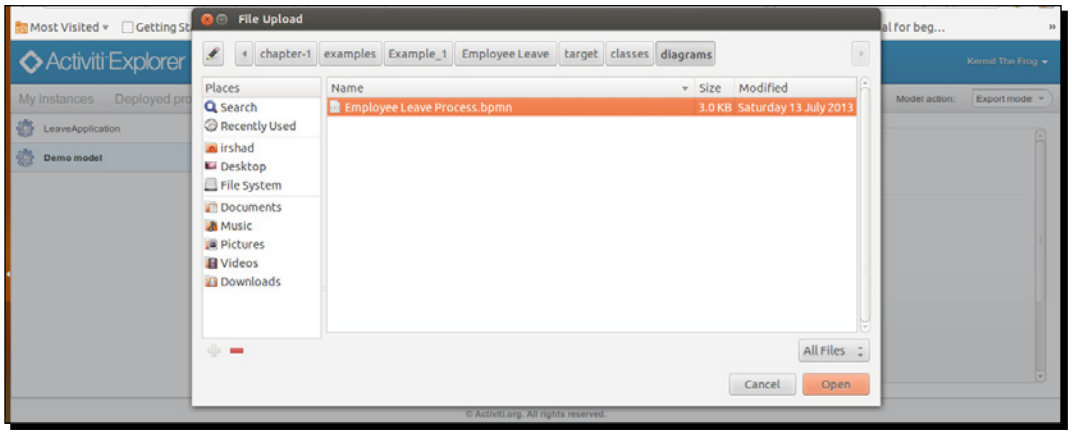
3. If you already have a business process created and want to view that process in the Activiti Modeler, you can use the **Import** button available at the top-right corner of the window:



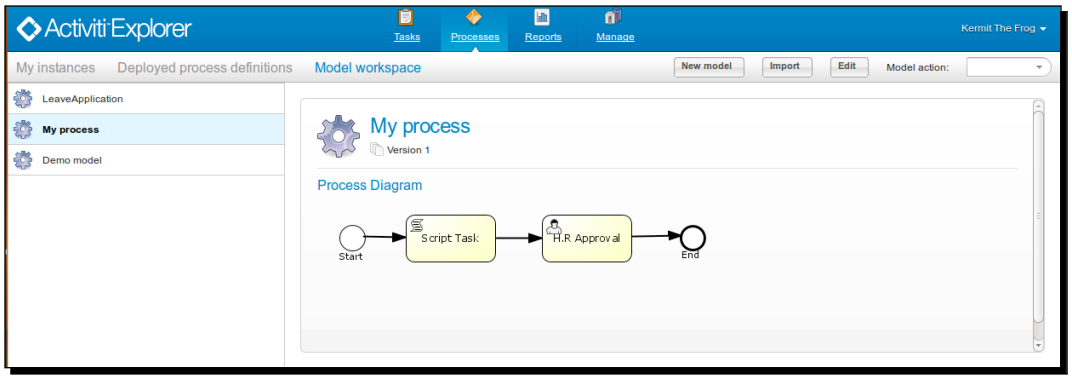
4. On selecting the **Import** button, the following window will pop up that clearly indicates that you can import files only with the `.bpmn20.xml` and `.bpmn` extensions, as shown in the following screenshot:



5. In the previous chapter, we created an example using the Eclipse editor. So, we will import that example to the Activiti Modeler. Browse to the path where the `.bpmn` file is located and import the file as shown in the following screenshot:



6. After the import operation, you can see the process available in the list on the left pane:



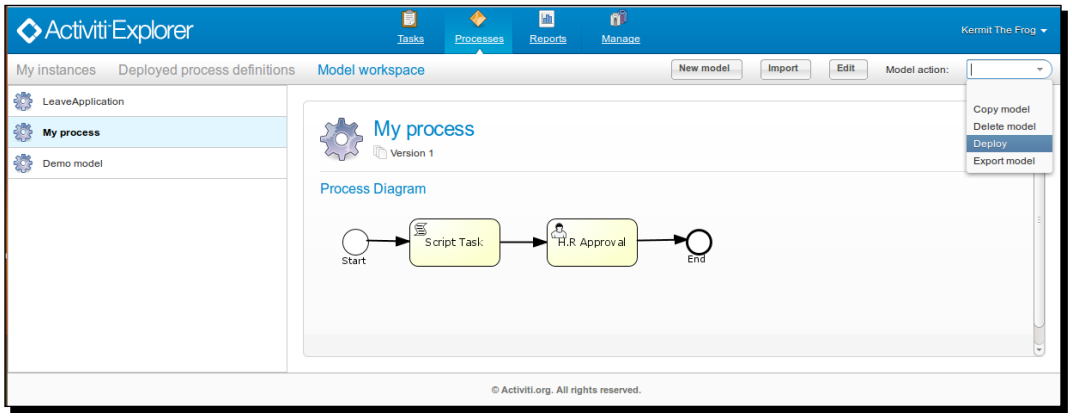
What just happened?

We came across how to import and export the business process from or to the Activiti Explorer.

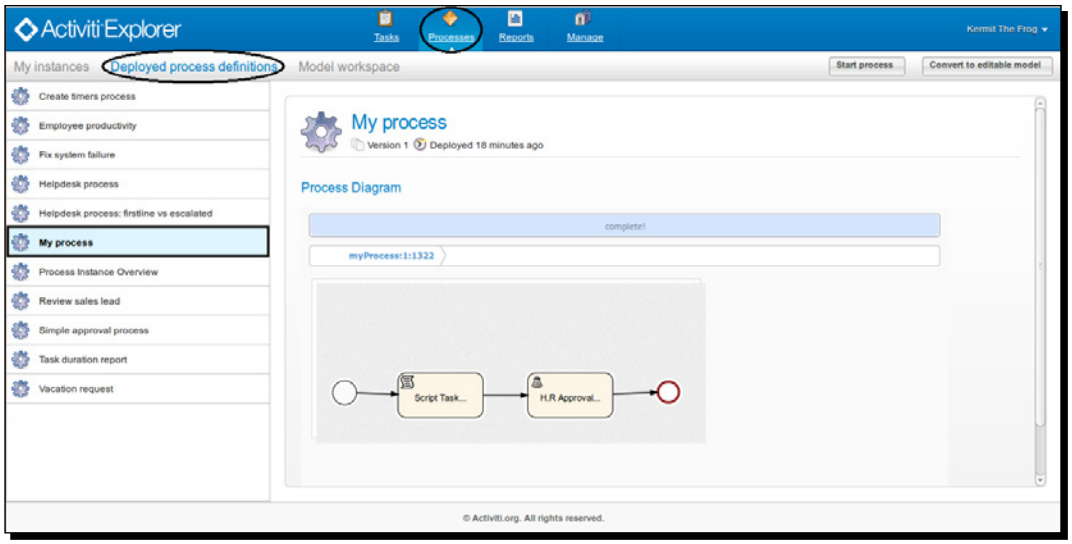
Time for action – deploying a model into the Activiti Engine

After creating the business process, it's now time to deploy it into the Activiti Engine:

1. To deploy a process into the Activiti Engine, you need to select the process, and from the **Model action** drop-down menu, select the **Deploy** option:



2. After the successful deployment of the business process, you can find it in the **Deployed process definitions** menu. You can even start executing the process by selecting the **Start process** button at the top-right corner of the window:



What just happened?

After the creation, import, and export of the business process, there was the curiosity to execute the process in a working environment. So, we deployed the business process into the process engine.

Summary

So far, we have gained knowledge regarding BPM, the lifecycle of BPM, the history of BPM, and the current standards in use for modeling business processes. We also had a look at the various BPM elements used for modeling a business process. In this chapter, we also learned how to install, configure, and use the Activity Modeler.

In the next chapter, we will learn how to configure the Activiti Designer in Eclipse, use the designer to design a business process, and how to test it. So, in the next chapter, we will perform a completely hands-on exercise using an example.

3
Designing Your Process Using the Activiti Designer

In the previous chapter, we learned about the Activiti Modeler and how we can use it to design a process. We are now able to import and export models using the Activiti Modeler. Using the Activiti Modeler to design a business process is mostly preferred by business analysts and architects because it is a web-based designing tool and is easy to use. However, when it comes to designing an executable business process, the Modeler doesn't seem helpful; for this, Activiti provides an Eclipse-based tool called Activiti Designer.

In this chapter, we will discuss the following topics:

- Introduction to the Activiti Designer
- Configuring the Activiti Designer with Eclipse
- Designing the first process
- Testing the process
- Importing the process to the Activiti Modeler

Understanding the Activiti Designer

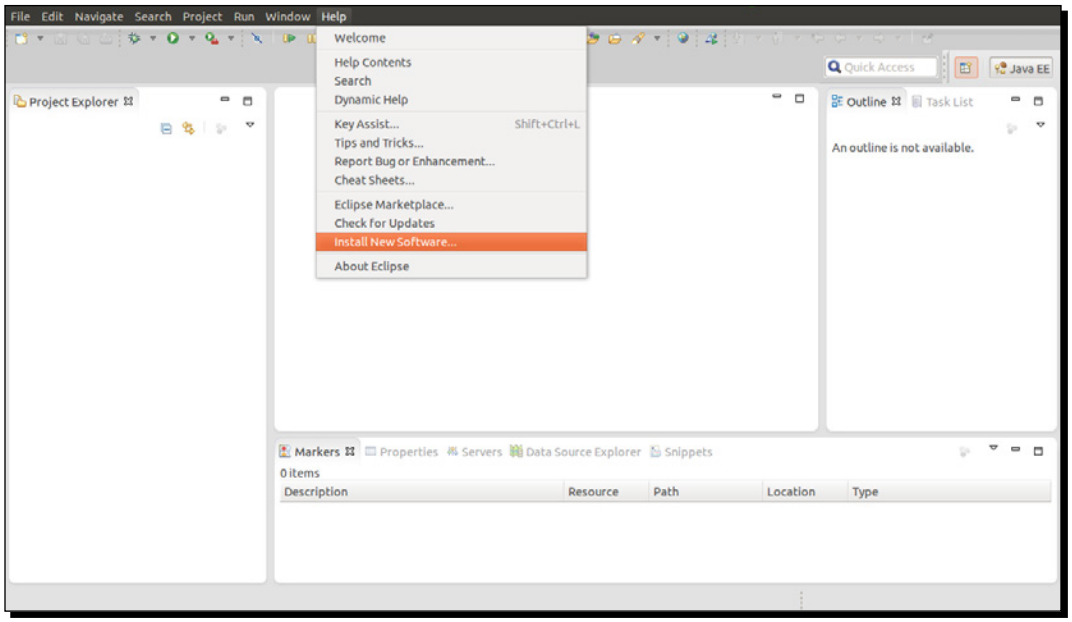
The Activiti Designer is an Eclipse-based workflow designing tool of Activiti. It is only supported by Eclipse Juno or Eclipse Indigo. There is a difference between a Designer and a Modeler, even though both are used for designing business processes.

The difference is that the Activiti Modeler is a web-based process designing tool and cannot be used to provide technical details to the business process, whereas the Activiti Designer is used to design the business process as well as provide technical details to your business process.

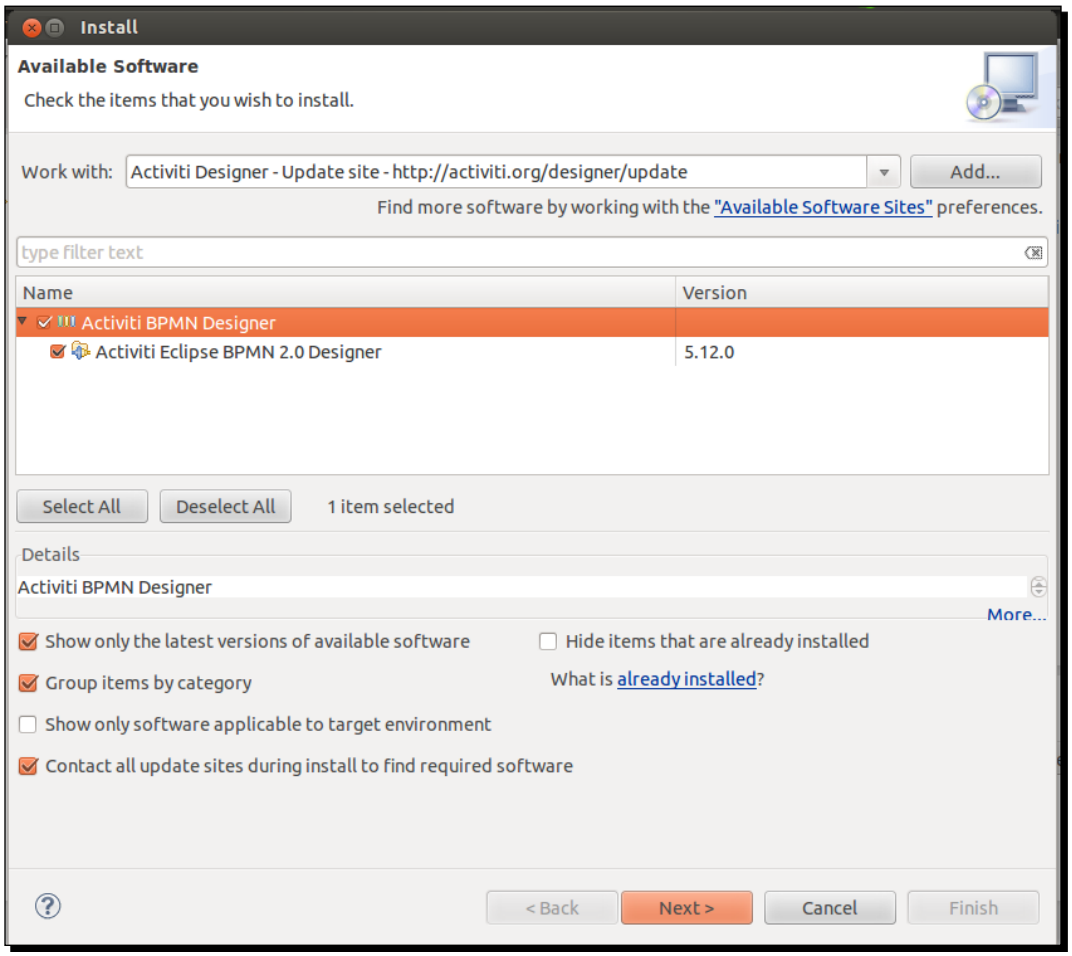
Time for action – downloading and installing the Activiti Designer

It's time to use the Activiti Designer to design a business process. To start designing a process using the Activiti Designer, you need to configure Eclipse with the Activiti Designer by performing the following steps:

1. For configuring Eclipse with the Designer, you need to navigate to **Help** | **Install New Software...**, as shown in the following screenshot:



2. In the following screenshot, you can see the pop-up window where you have to provide the URL for the Activiti Designer, `http://activiti.org/designer/update`, which will fetch the latest Activiti Designer plugin for Eclipse; after the successful update, restart Eclipse:



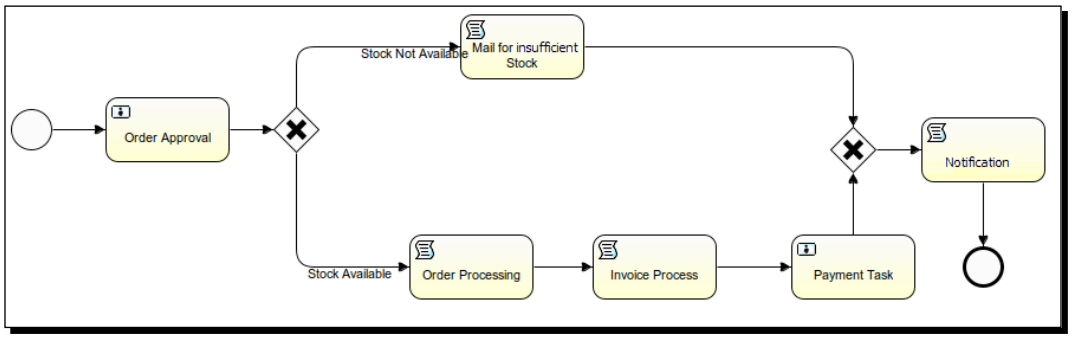
What just happened?

We have just configured the Activiti Designer using the Activiti update site's URL. This URL fetches the updated Activiti Designer.

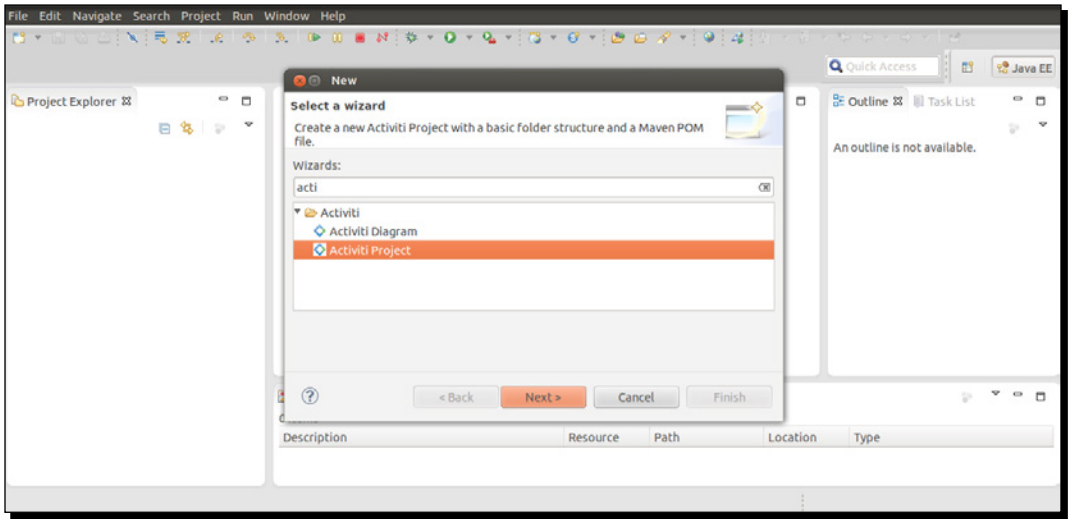
Time for action – designing your first process

We have designed a business process using the Activiti Modeler. To add technical details to a business process, we have to design a business process using the Activiti Designer. So, let's design a business process in the Activiti Designer by performing the following steps:

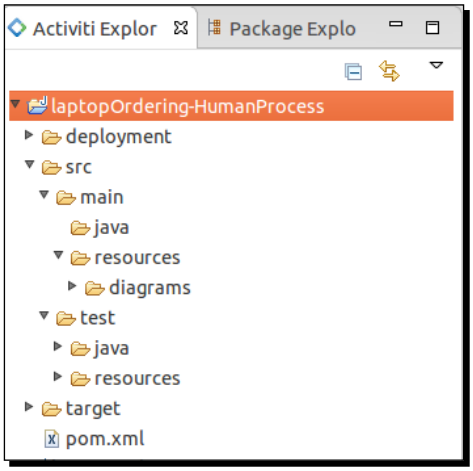
1. After configuring the Activiti Designer in Eclipse, we will design an executable process. So, let's take the example of the laptop ordering process, as shown in the following diagram:



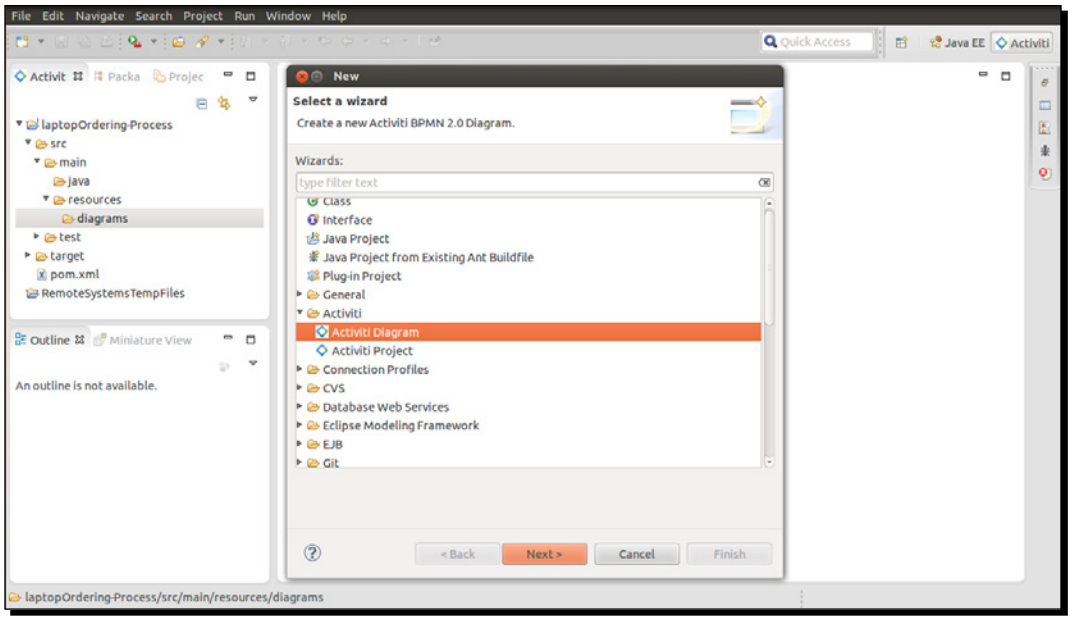
2. In the laptop ordering process, we will have three user tasks and three script tasks.
3. We will create a new Activiti project to start the process creation; for that, we have to navigate to **File** | **New** | **Activiti Project** and provide a name, for example, `laptopOrdering-HumanProcess`.



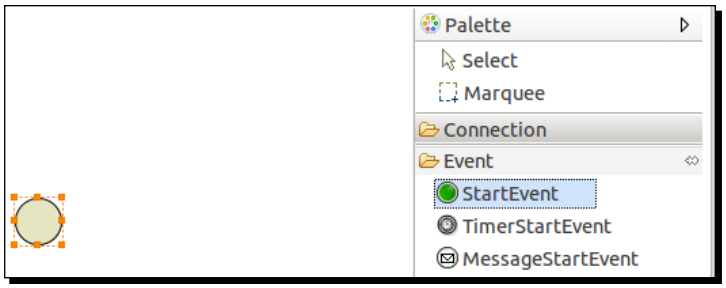
4. Upon the successful creation of the Activiti project, we will have created a directory structure as shown in the following screenshot. In the `main` directory, we can have the production environment's implementation files, and in the `test` directory, we can have the test environment's implementation files; there is also a `pom.xml` file available, which can convert the Activiti project to a Maven project and execute it as such.



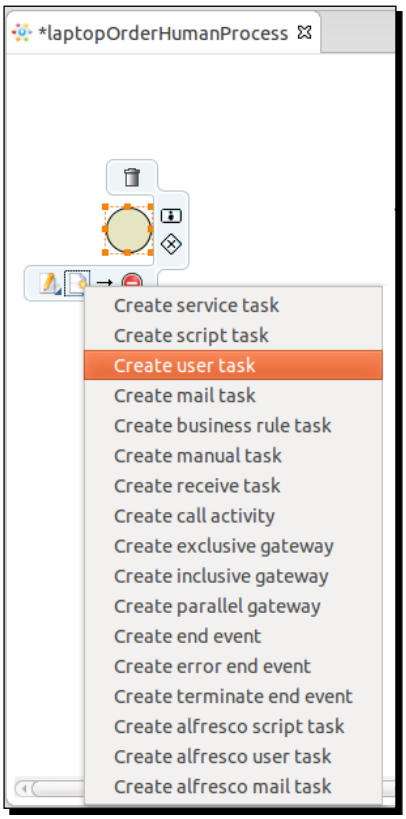
5. Now we will design the process diagram using the **Activiti Diagram** option by navigating to **File**| **New** | **Other** | **Activiti Diagram**, which can be found at **src** | **main** | **resources** | **diagrams** with the name `laptopOrderingProcess`.



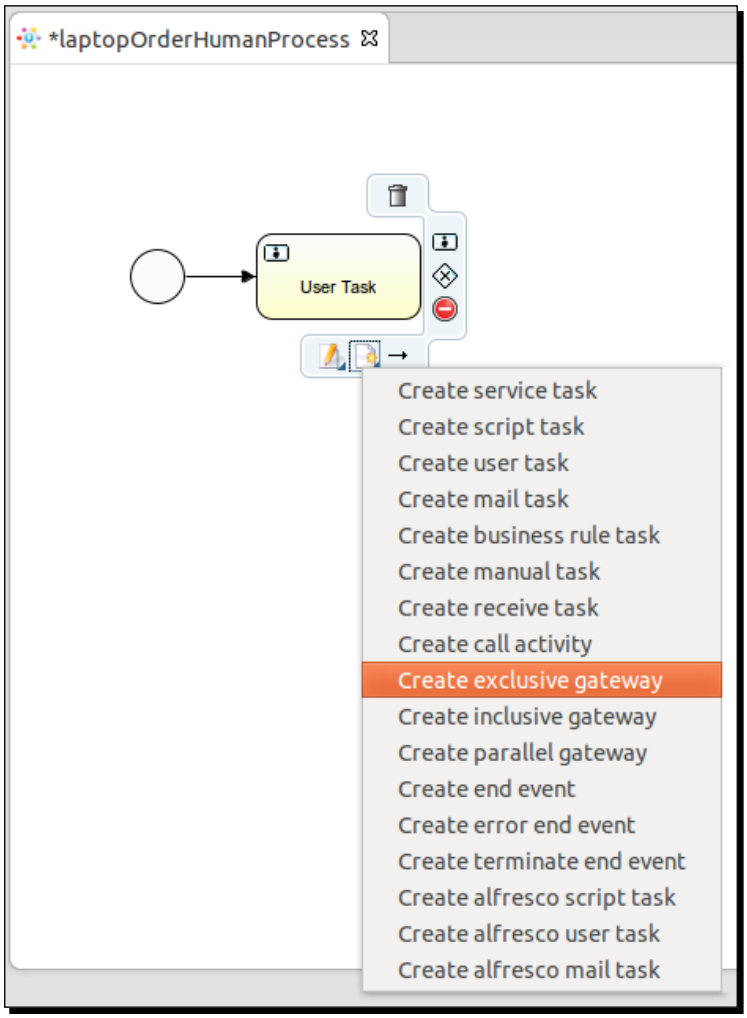
6. Once the `laptopOrderingProcess.bpmn` file is created, drag-and-drop a **StartEvent** option from the **Event** tab available in the **Palette** section on the right-hand side of the screen, as shown in the following screenshot:



7. We can place the next node by selecting it from the start node itself. As we want a user task connected after the start node, we will select the **Create user task** option from the list shown in the following screenshot:



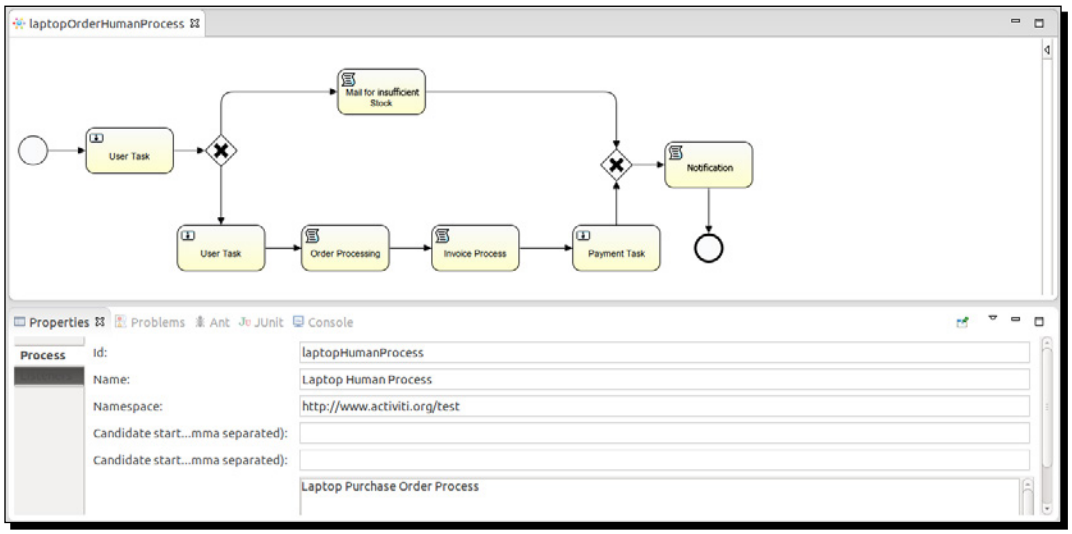
8. Now based on the output provided by the execution of the user task, the process has to decide which path to execute. To decide the path, a gateway is used. There are three types of gateways available: inclusive, exclusive, and parallel; they are explained in *Chapter 2*, *Modeling Using the Activiti Modeler*. In this process, we will be using the exclusive gateway:



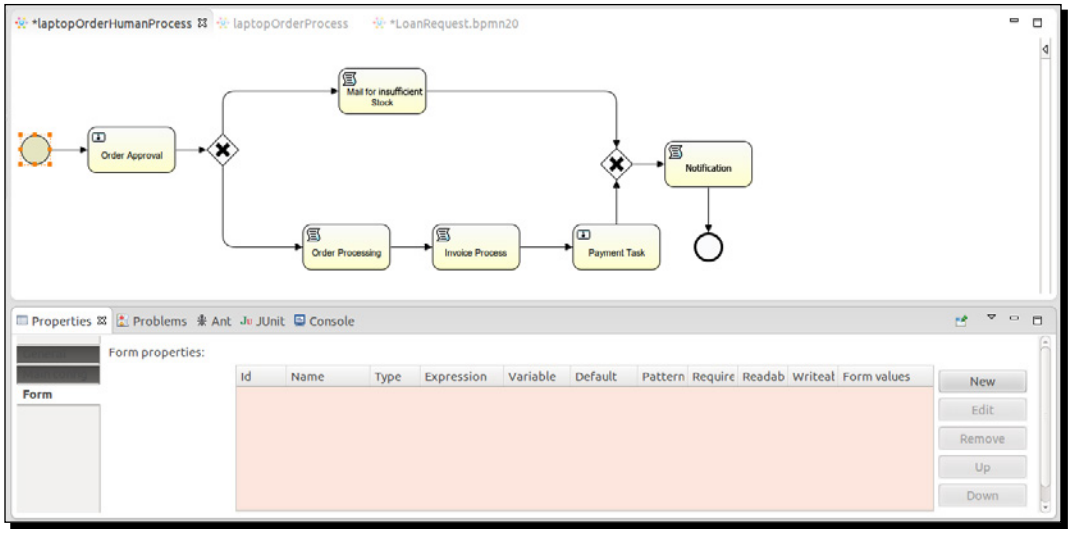
9. We will design the entire process flow using script tasks, user tasks, and the exclusive gateway.

10. After designing the entire process, we will populate the properties of the business process and the nodes used.

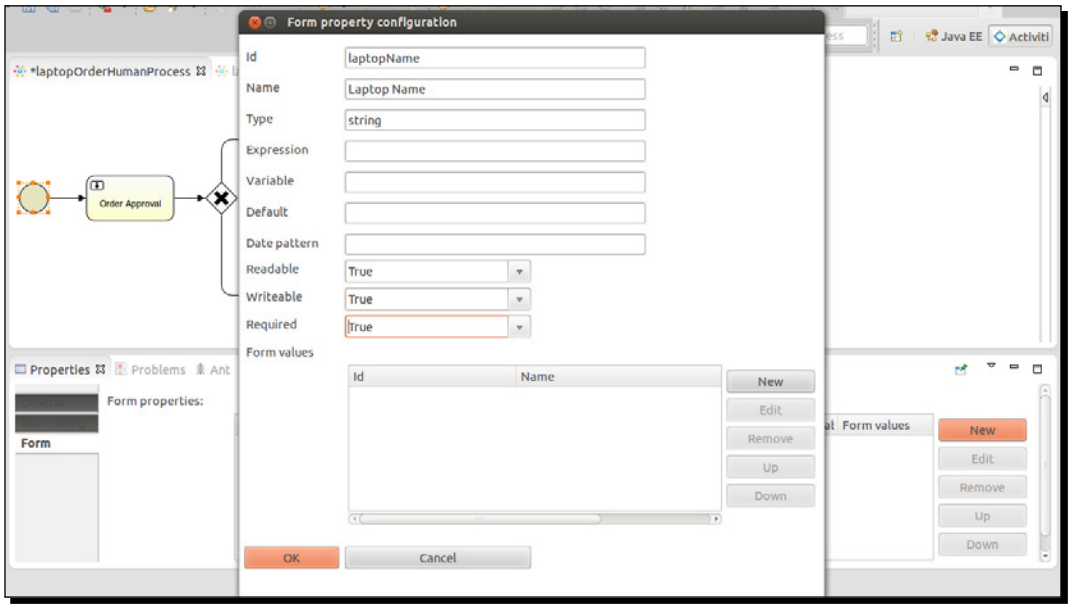
11. We will start by populating the properties of our business process. We will provide a unique name and ID for our process. For that, we need to select the white canvas and open the **Properties** window in which these properties can be specified, as shown in the following screenshot:



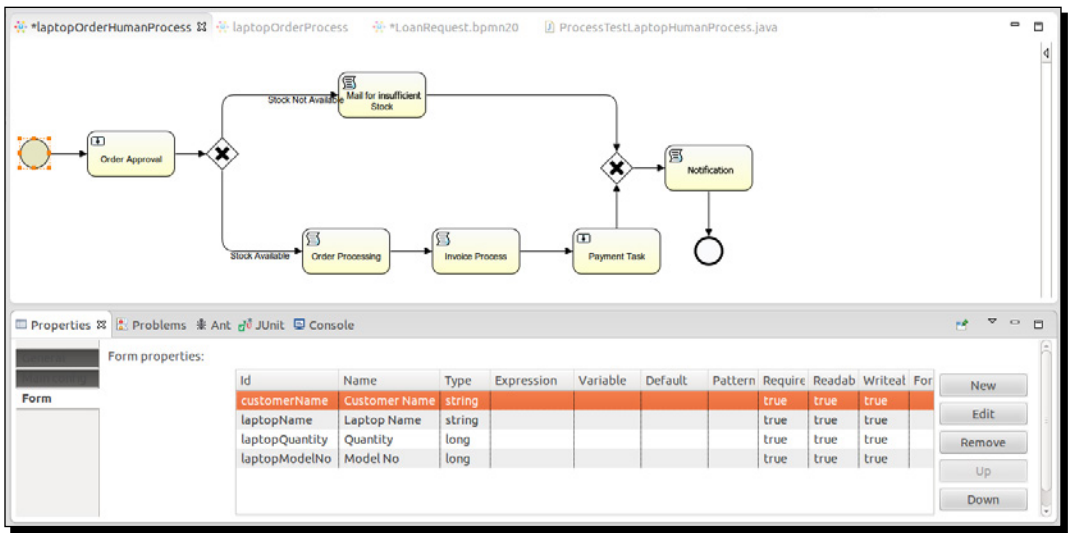
12. We want some values to be provided at the start of the process. For this, populate the form properties of the start node, select the start node, open the **Properties** window, and select the **Form** tab from the **Properties** window, as shown in the following screenshot:



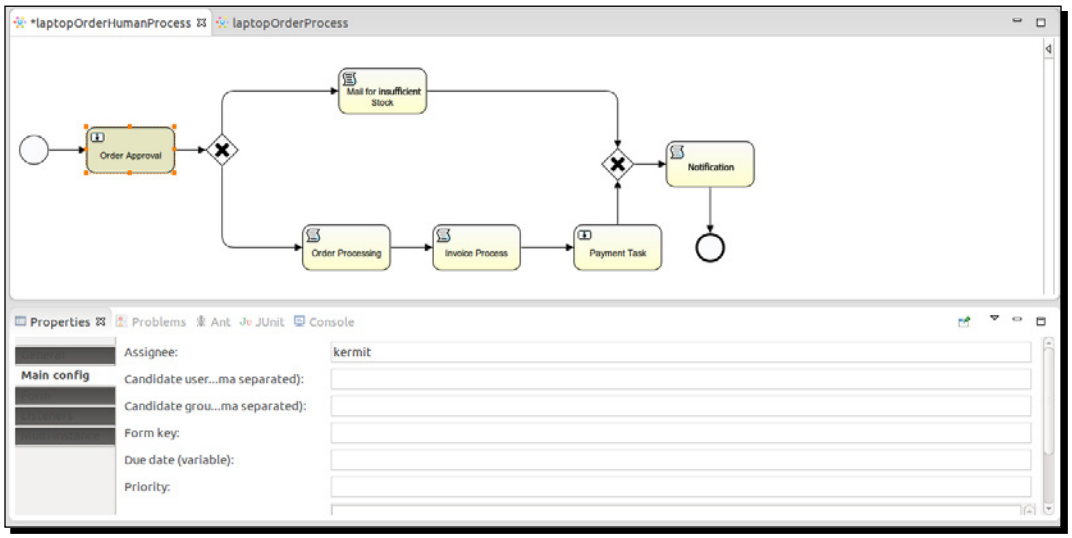
13. To add a new form property, select the **New** option; this will open a form where variable declarations can be made. At the start of the process, the customer will provide some information about the laptop to be ordered, such as the laptop name, quantity, model number, and customer name. We will declare these variables using the form property, as shown in the following screenshot:



14. Activiti supports only certain data types, such as `boolean`, `long`, `string`, `enum`, and `date`. There are other properties, such as **Readable**, **Writeable**, and **Required**, that are by default set to `true`. We will look at the instances in which these properties need to be set to `false`. So, the form properties that we are setting in the start event will be visible to the user when they start the process.

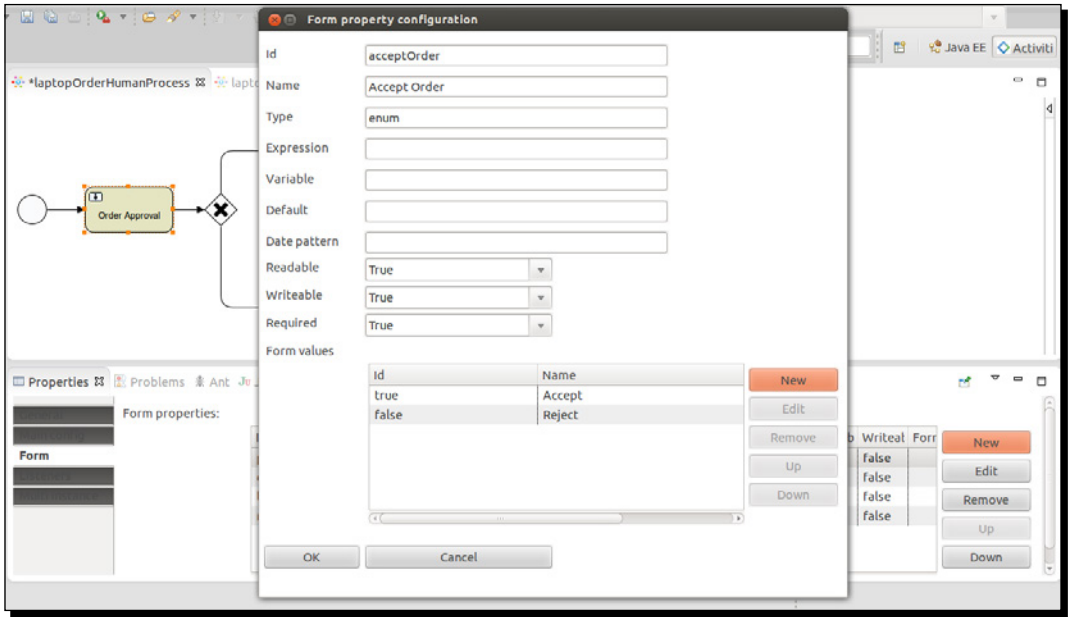


15. Once the customer provides the details regarding the laptop, they should be passed to the **Order Approval** user task. This task will be assigned to a specific user who will either approve or reject the order. To assign this task to a specific user, populate the **Assignee** property of the user task. By default, there are three users available. All these properties are available in the **Main Config** tab of the user task properties.

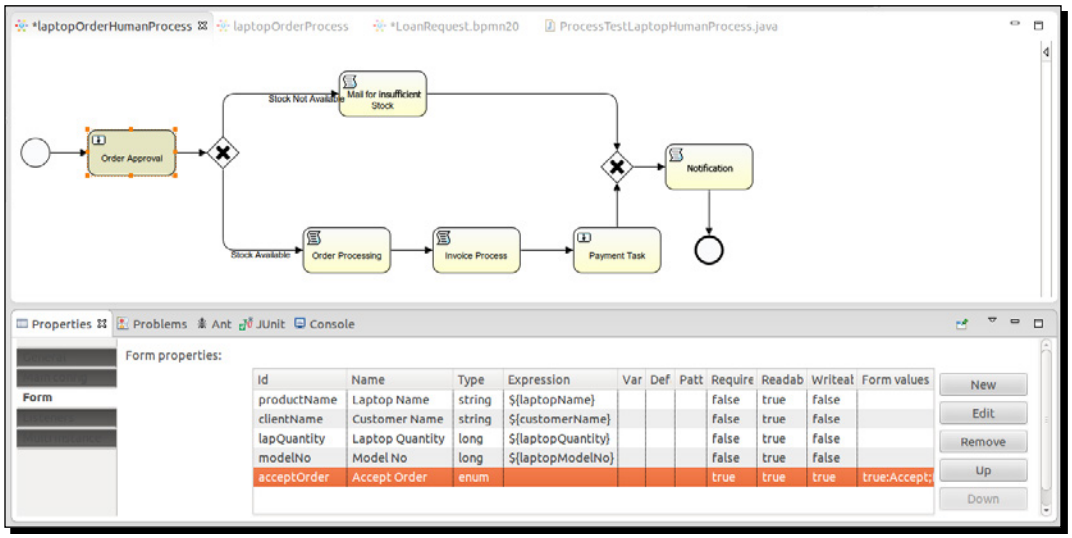


If you want to assign a task to multiple users, you will have to populate the **Candidate user comma separated** property in this field and provide the names with comma-separated values. If you want the task to be assigned to a group, populate the **Candidate group comma separated** property with group names.

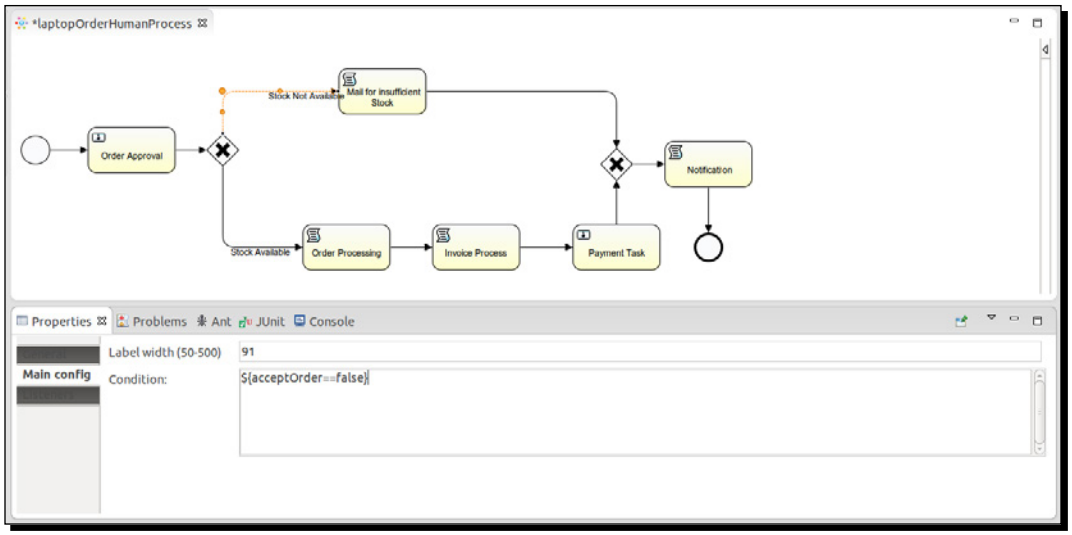
16. The user task is only responsible for accepting and rejecting orders. The user task also has form properties; we will be setting **Type** as `enum` and in **Form values** we will provide the values to be displayed, as shown in the following screenshot:



17. To display the details provided by the user, we will have to populate the **Expression** property. In the **Expression** property, we have to use the `${id}` syntax. As shown in the following screenshot, the **Expression** values are the IDs that were provided by us in the start node, and the **Writeable** and **Required** properties are set to `false` as these values cannot be edited:

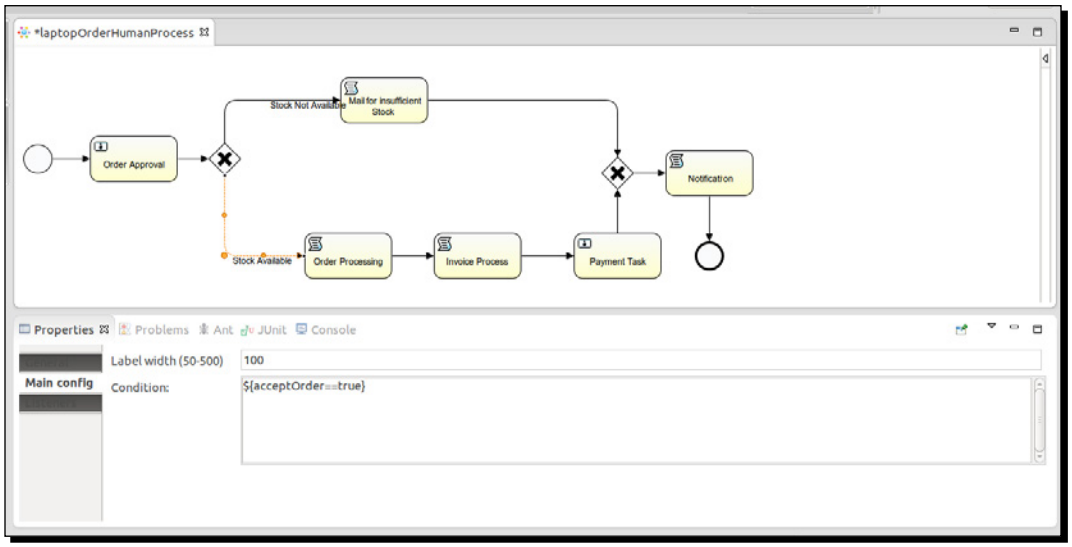


18. We have defined an exclusive gateway connected after the **Order Approval** task; it checks the output provided by the **Order Approval** task, that is, whether the order is accepted or rejected. There are two outgoing connections from the gateway: one for insufficient stock and another for accepting the order. To provide conditions, we have to make changes in the **Condition** property of the **Stock Not Available** flow with the `acceptOrder` value. This is the form property defined in the **Order Approval** task. So, here we are checking whether or not the output is set to `false`. If it is, follow the path shown in this screenshot:

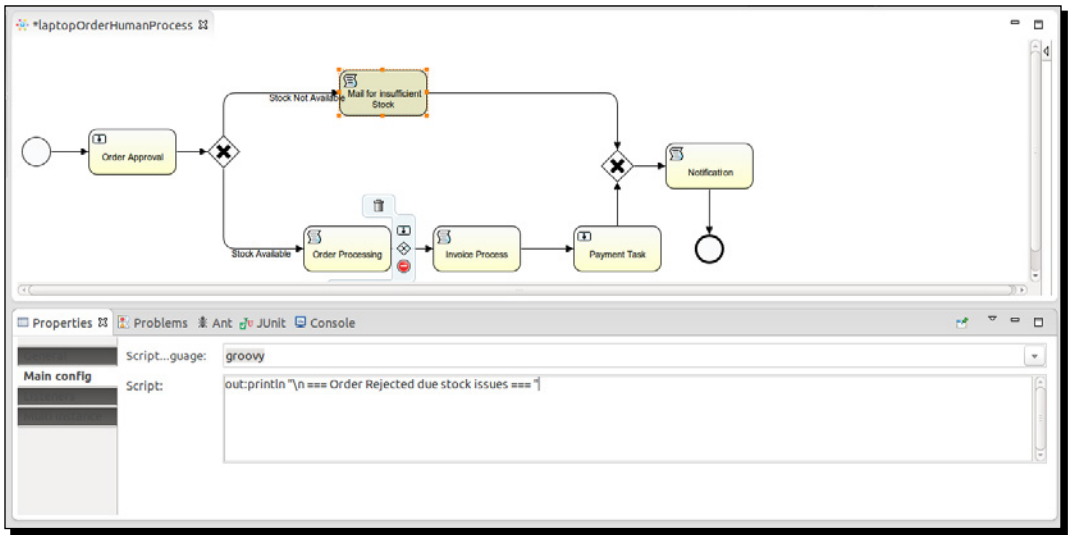


Conditions can be defined on the paths but not on the gateway.

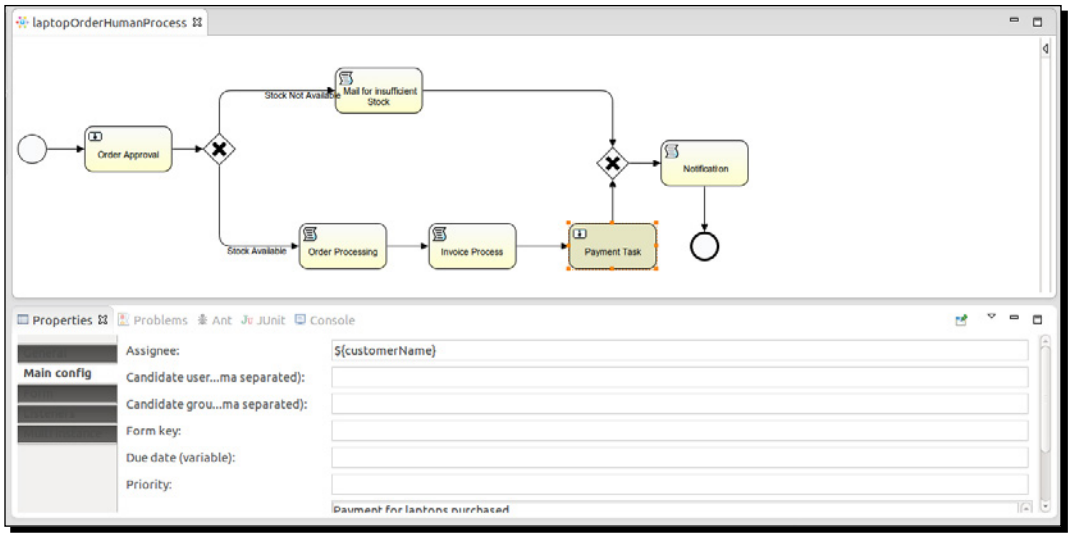
19. In the other flow conditions property, we are checking whether or not the output is set to `true`. If it is, follow the path shown in this screenshot:



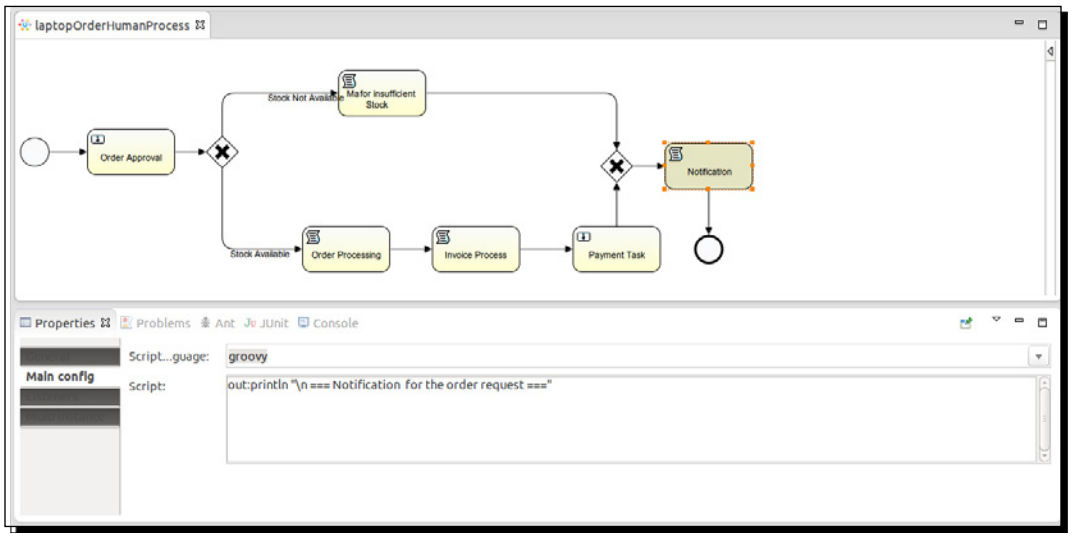
20. If the `acceptOrder` values are set to `false`, a script task named **Mail for insufficient stock** is executed. For this script task, we have set the **Script Language** property to `groovy`, and in the **Script** property, we have just printed a simple message to display on the console. Similarly, if the output is set to `true`, the **Order Processing** script task should be executed first and then the **Invoice Process** script task and the **Payment Task** user task, as shown in the following screenshot:



21. For the **Payment Task** user task, we are going to dynamically assign users. In the following screenshot, you can see that the **Assignee** property is populated with `${id}`, `customerName`, which is the form property field of the start event; this was the name provided at the start of the process. The **Payment Task** will be assigned to the particular user who has been assigned.



22. After the **Payment Task** and **Mail for insufficient Stock** tasks are completed, we have to add an exclusive gateway that has two incoming connections and one outgoing connection. This means that if any of the incoming flow is executed, the process will proceed to the **Notification** script task, the message defined in the **Script** property will be displayed, and the process will be terminated as there is an end node defined after the **Notification** task is executed, as shown in the following screenshot:



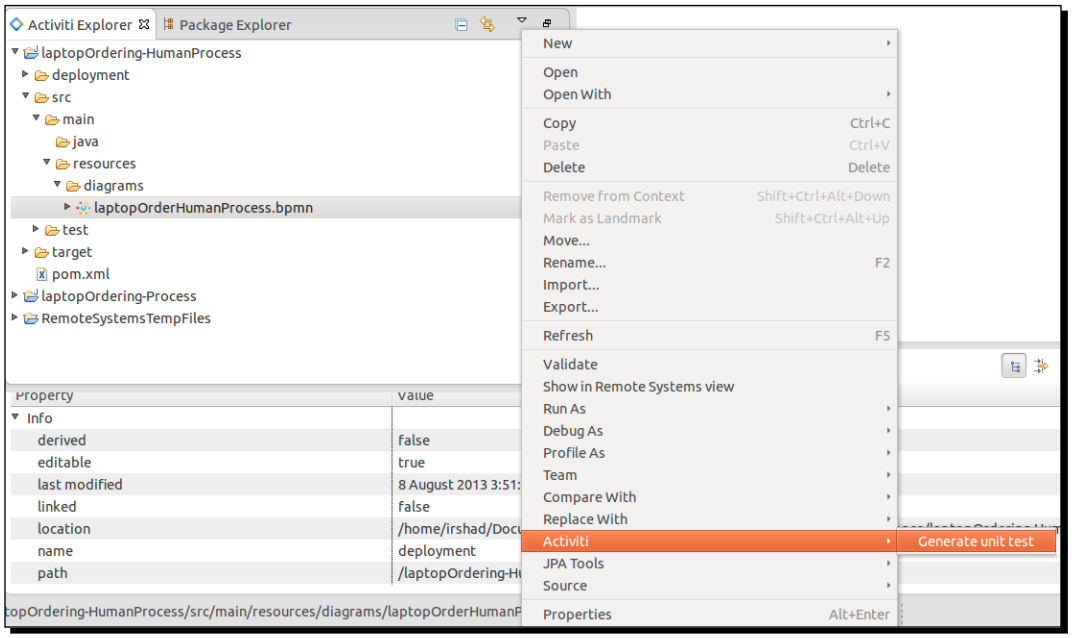
What just happened?

We have just finished discussing process creation using the Activiti Designer and also gained some knowledge regarding the various nodes and how to populate their properties. Now you should be able to design your own process using the Activiti Designer by adding technical details to your process.

Time for action – testing your process

Once the process is designed, you need to test it before deploying it to the Activiti Engine or production environment by performing the following steps:

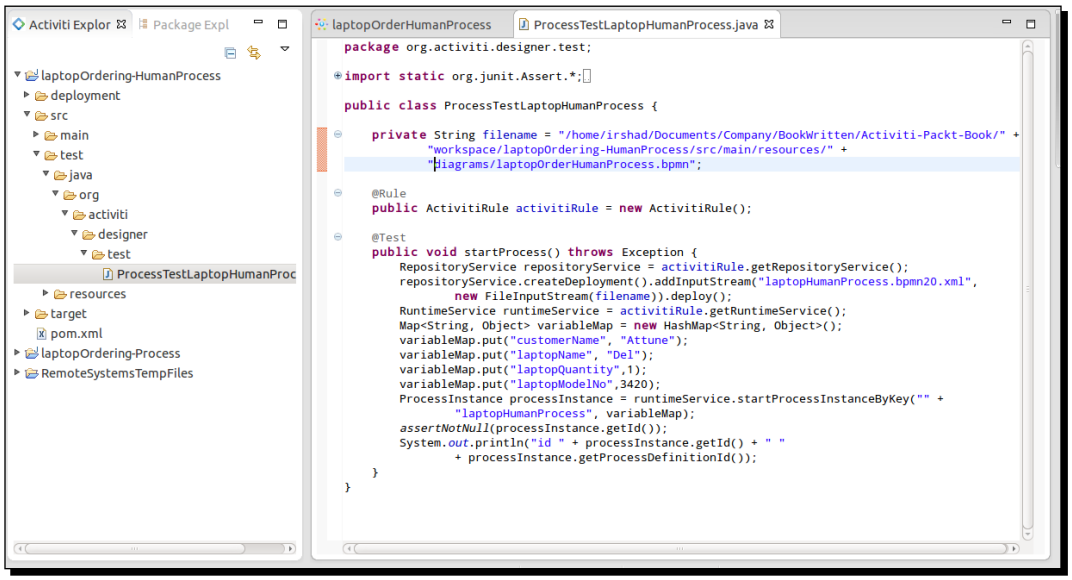
1. Now we will create a test file to test our Activiti process. To create a test file, we have to select the process file `.bpmn`, right-click on it, and select **Activiti** from the list. It will contain the **Generate Unit test** option; select it or just click on it.



2. Clicking on the option will generate a test file in the `test` folder as shown in the following screenshot, as the class file contains the path of the `.bpmn` file and the methods to execute the process. In the class file, there is also a map variable that is populated with `customerName`, `laptopName`, `laptopQuantity`, and `laptopModelNo`. These are the form properties defined in the start node of the process. These variables are passed to the process using the following code:

```
ProcessInstanceprocessInstance = runtimeService.startProcessInstanceByKey("laptopHumanProcess", variableMap);
```

This code snippet is shown in the following screenshot:



3. To test the business process, run the class file as a JUnit test. If your process is completed successfully, you will get the following screenshot; if not, an error message will be displayed on the screen:

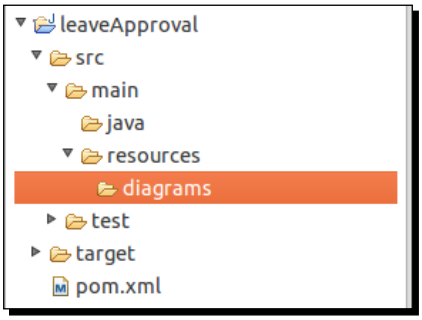
What just happened?

After successfully designing the process, we have gone through the testing phase of the process. In this section, we saw how to create a test file for our process and run it to check whether or not the process created is working properly.

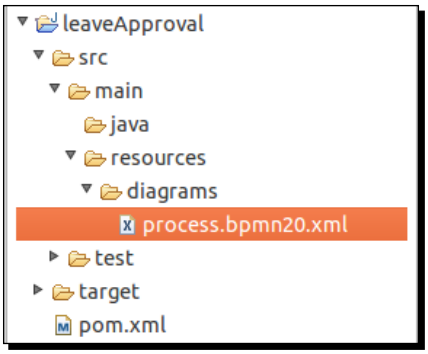
Time for action – importing a process from the Activiti Modeler to the Activiti Designer

Now, you are familiar with designing a process using the Activiti Designer and testing it using Eclipse. Now let's say a business process has already been designed by a business analyst and the developer has been asked to add technical details to the existing process. As in *Chapter 2*, *Modeling Using the Activiti Modeler*, we have designed a business process using the Activiti Modeler and exported and saved it. Now we will see how to import that process to the Activiti Designer. In this section, we will look at how to import a process from the Activiti Modeler to the Activiti Designer by performing the following steps:

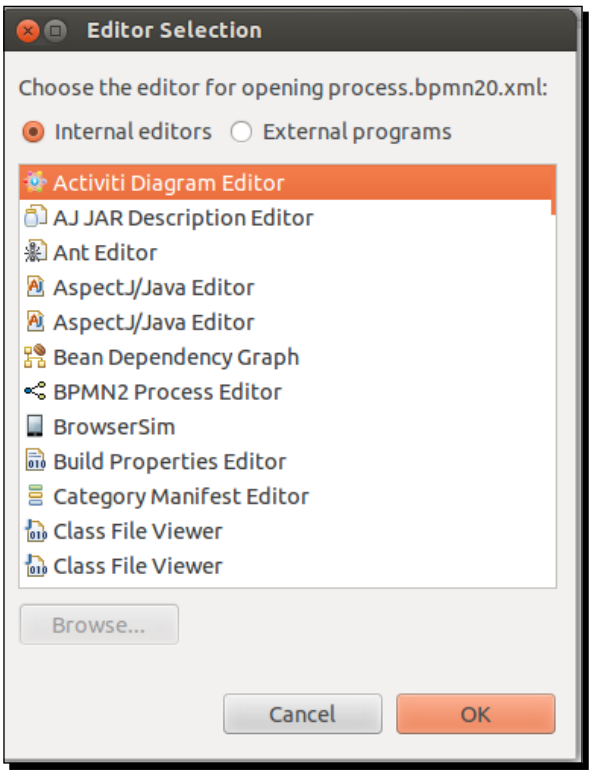
1. To import a process to the Activiti Designer, we have to copy the process created in *Chapter 2*, *Modeling Using the Activiti Modeler* to the Eclipse project. Create a new project in Eclipse as we did in the *Designing your first process* section. Just create a project; do not create an Activiti diagram in it.
2. After creating the project in Eclipse, paste the process' `bpmn20.xml` file in the `diagrams` folder of the project created, as shown in the following screenshot:



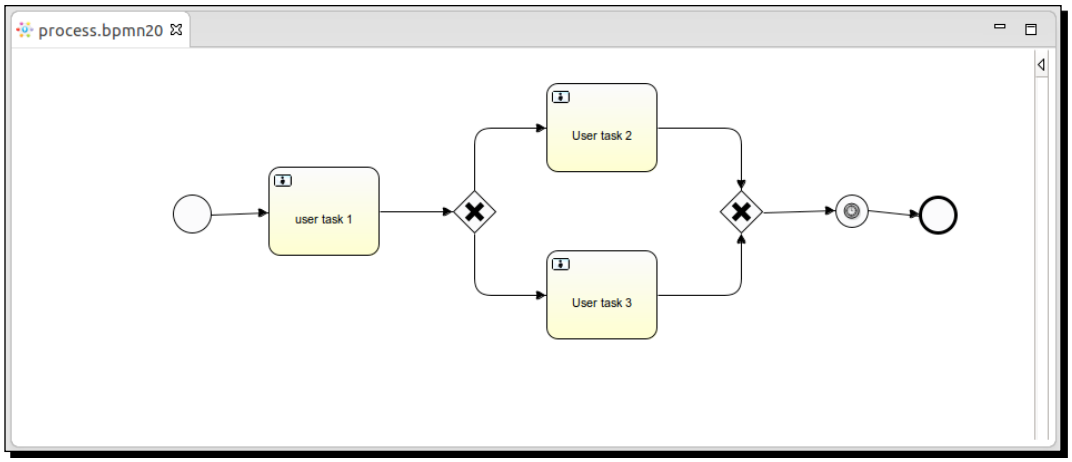
3. To open the process in Designer mode, right-click on the `process.bpmn20.xml` file, browse through it, and open it in another window.



4. This will open a pop-up window. Select **Activiti Diagram Editor** from the pop-up window, as shown in the following screenshot:



5. Opening the process using the Activiti Designer Editor will open it in graphical mode. Here, you can edit the process, add technical details to it, and again deploy it to the Modeler.



What just happened?

Now you're familiar with designing a process using the Activiti Designer and testing it in Eclipse. After testing the process, we need to import the process to the Modeler. In this section, we learned how to import our process to the Modeler.

Have a go hero

Create a driving license business process using the Activiti Designer. At the start of the event, use one form to get applicants' information, such as their name, age, and type of license. Then check the age condition in a script task; if all the conditions are satisfied, allocate the document approval task to the DMV officer. So for the approval task, take one human task. Send a confirmation message to the applicant about the status of their application, which can be either approved or rejected.

Pop quiz – the Activiti Designer

Q1. What is the name of the property that is used for executing Groovy script in a script task?

1. Flow condition
2. Script
3. Id
4. Form properties

Q2. How many different gateways are available?

1. 2
2. 3
3. 4
4. 1

Summary

This chapter was completely focused toward getting hands on with the Designer to get a clear understanding of business processes; how to design one, the forms for tasks, and so on.

By now, you should have gained some knowledge regarding the Activiti Designer and be able to install the Activiti Designer in Eclipse. Using the Activiti Designer, you can design a business process and add technical details to it, as we have also covered designing forms for human interaction. You should also be able to test your business process using the Activiti Designer. We have also covered the importing of a process from the Activiti Modeler to the Activiti Designer and how to add technical details to it.

In the next chapter, we will have a look at deploying the business process in the Activiti Engine.

4

Management and Monitoring Using the Activiti Explorer

In the previous chapter, we learned about the Activiti Designer and how we can use it to develop business workflows. In this chapter, we will learn about the Activiti Explorer. The Activiti Explorer is a web-based application used to deploy, start, and manage the process instance. The end user can easily start the process, manage, and monitor it. The Activiti Explorer uses the H2 database by default, but we can easily change our database for the production environment.

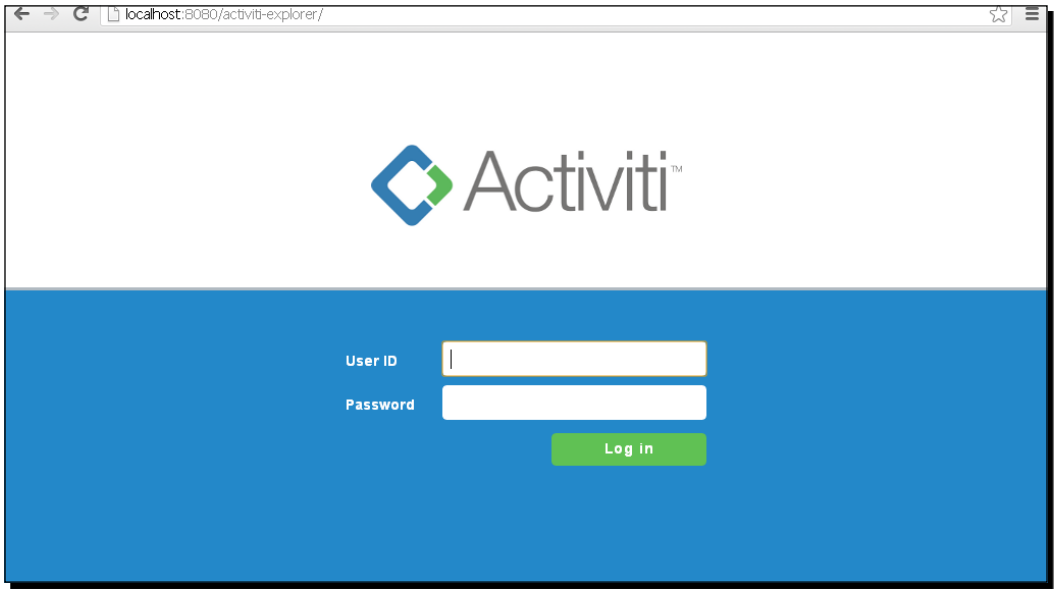
This chapter covers the following topics:

- Overview of the Activiti Explorer
- Deployment of the process
- Process management
- Reporting
- Administration using the Activiti Explorer

The Activiti Explorer provides an end user application for technical and non-technical people.

An overview of the Activiti Explorer

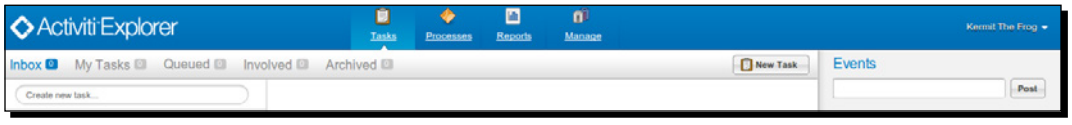
Here, we will just recall *Chapter 1*, *Installing Activiti*, in which we deployed the `activiti-explorer.war` file into Tomcat. We can access the Activiti Explorer by browsing to `localhost:8080/Activiti-Explorer`; upon doing this, we will get the following login screen:



You can log in using the following default user IDs and passwords:

- `kermit/kermit` is the admin with type `security-role`
- `gonzo/gonzo` is a simple user with type `assignment`
- `fozzie/fozzie` is a simple user with type `assignment`

After logging in successfully, we will get the following screen:



The features of the Activiti Explorer are as follows:

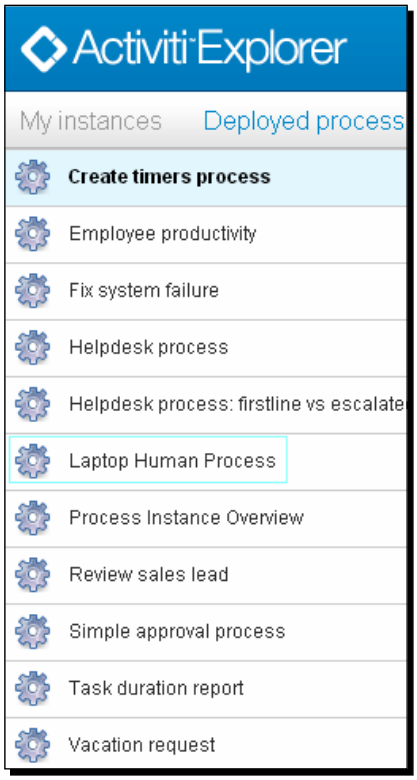
- **Tasks**: This tab manages all the task-related functionalities. The inbox will contain all the tasks assigned to the user. Tasks assigned to a group will be available in the **Queued** option. From this tab, we can start and complete the task.

- **Processes**: This tab manages the processes. We can view the processes that are in execution and those that are deployed into the engine as well as design and deploy business processes.
- **Reports**: This tab generates reports of the process that has been executed with information such as execution time, and saves those reports for future reference.
- **Manage**: This is a tab that offers complete administration functionalities and can be used to manage databases, process deployment, jobs, users, and so on.

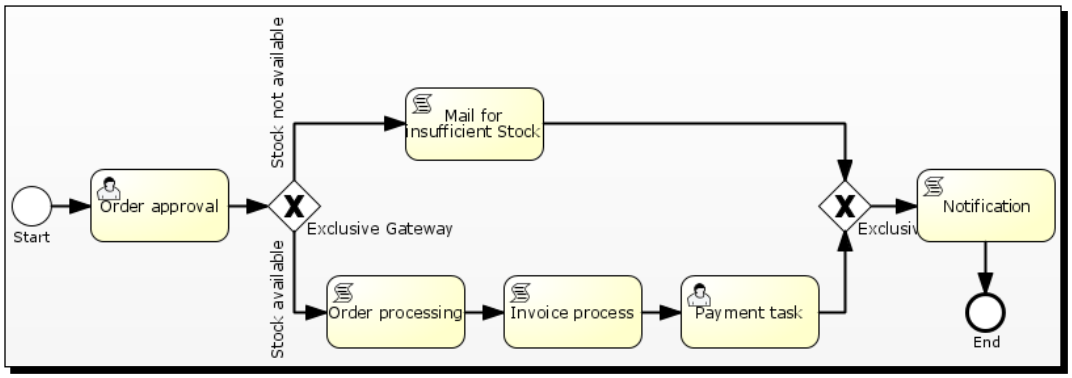
A process with the Activiti Explorer

The Explorer provides the functionality to deploy a process. So, the process we have developed using the Eclipse tool can be deployed into the Explorer. We can view a process graphically if a process image is available. We can deploy the process using a `.bar` file, which can be generated using the Eclipse tool.

We have already deployed various processes into the Activiti Explorer. You can see **Laptop Human Process** under the **Deployed process definitions** tab in the following screenshot:



When we click on **Laptop Human Process**, we get the process diagram shown in the following diagram:



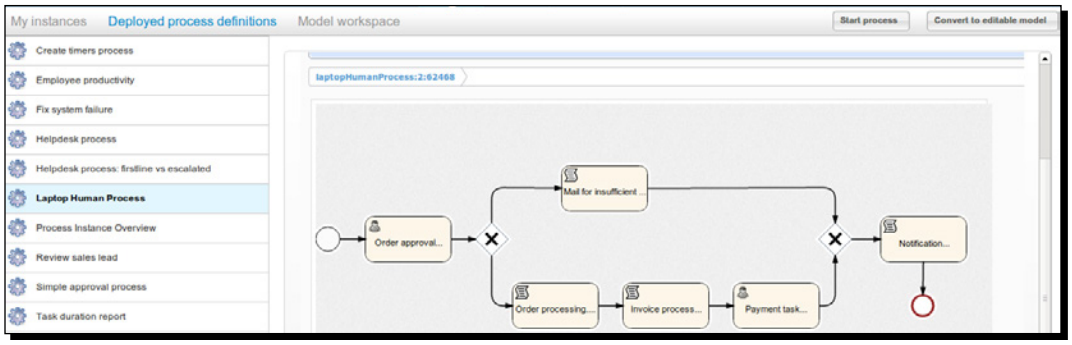
We can understand the process with the help of the preceding diagram. When a user places an order for a laptop, `kermit` will get an order-approval task. Then, if `kermit` rejects the order, the user will get a notification saying the product is out of stock. If the order is approved by `kermit`, the user will get a payment task, and after the payment is made, the user will get a notification.

Time for action – starting a process instance

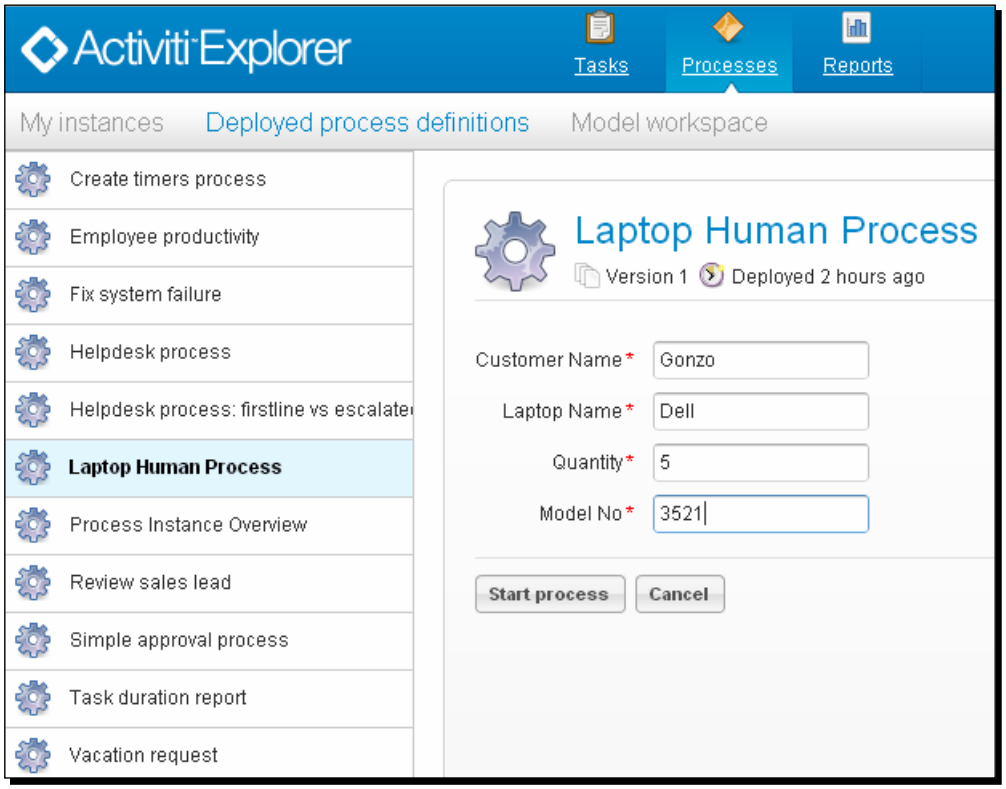
We will now start an instance of the laptop order process. To start an instance, we have to perform the following steps:

1. The user who placed the order for the laptop should be the `gonzo` user. So, log in with the user ID and password as `gonzo` as shown in the following screenshot:

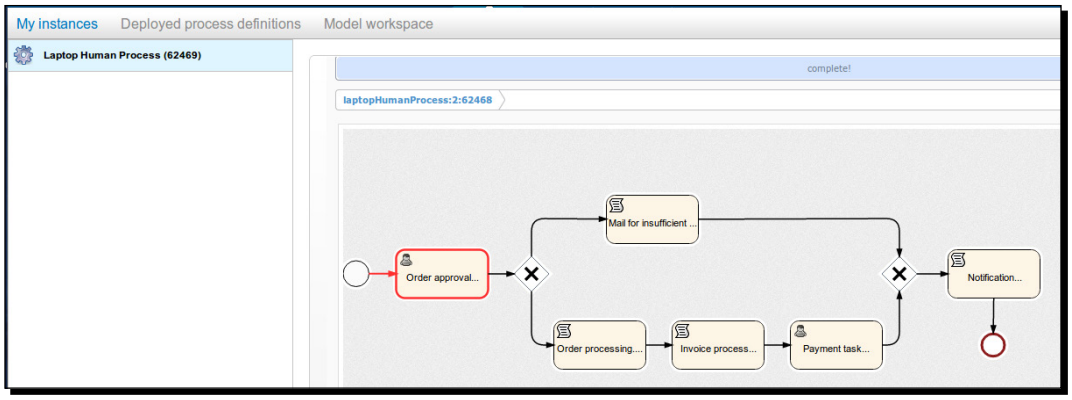
2. Now, start the process instance of the laptop order process. After logging in, browse to the **Process** tab, select the **Laptop Human Process** tab, and then click on **Start process**, as shown in the following screenshot:



3. We will now provide the details regarding the laptop to be ordered. So, at the start of the process, a form will be displayed with some fields. Fill in the required information and click on **Start process** as shown in the following screenshot:



4. When the process starts, you can view its status within the **My instances** tab, as shown in the following screenshot. This tab will provide information regarding the current task in execution.

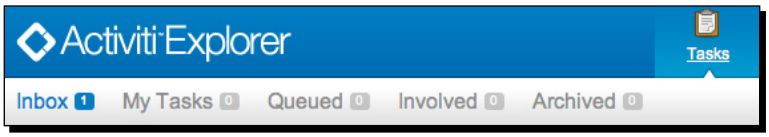


What just happened?

In this section, we started an instance of the laptop order process through the `gonzo` user. So, the approval task is assigned to `kermit`.

Managing tasks with the Activiti Explorer

In the previous section, we started a process. Now, in this section, we will have a look at how to manage a task using the Activiti Explorer.



In the **Tasks** tab, we have all the information related to a task. As you can see in the previous screenshot, there are various menus within the **Tasks** tab. The menus are explained as follows:

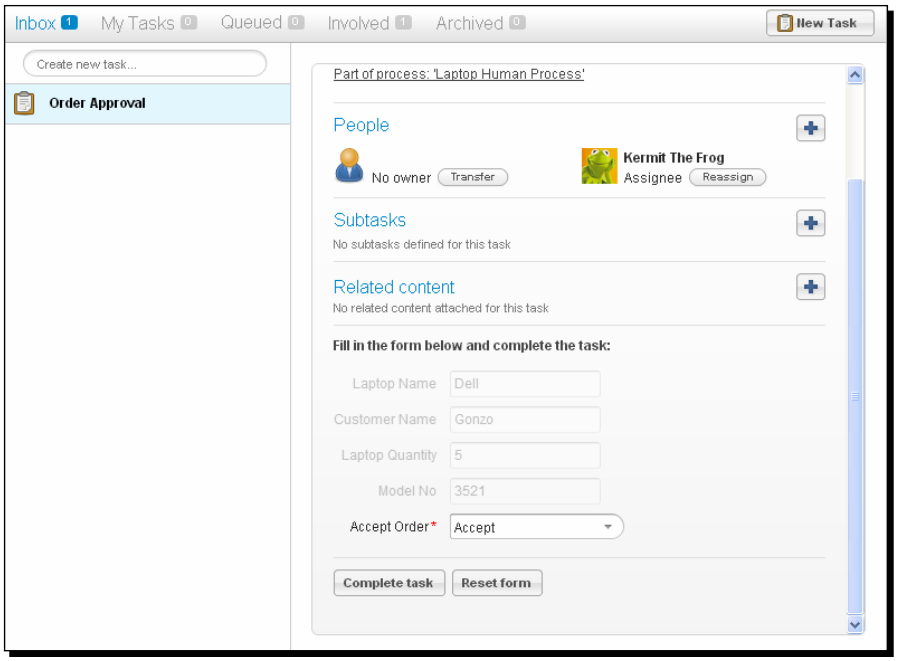
- **Inbox**: This menu represents the list of tasks assigned to the user.
- **MyTasks**: This menu shows the tasks created by the user.
- **Queued**: This menu displays the list of tasks assigned to a group that the user belongs to. In order to execute a task, the user will have to claim the task from the **Queued** list.
- **Involved**: This menu displays the list of available tasks in which the user is involved.
- **Archived**: This menu represents the list of tasks completed by the user.

Time for action – working with user tasks

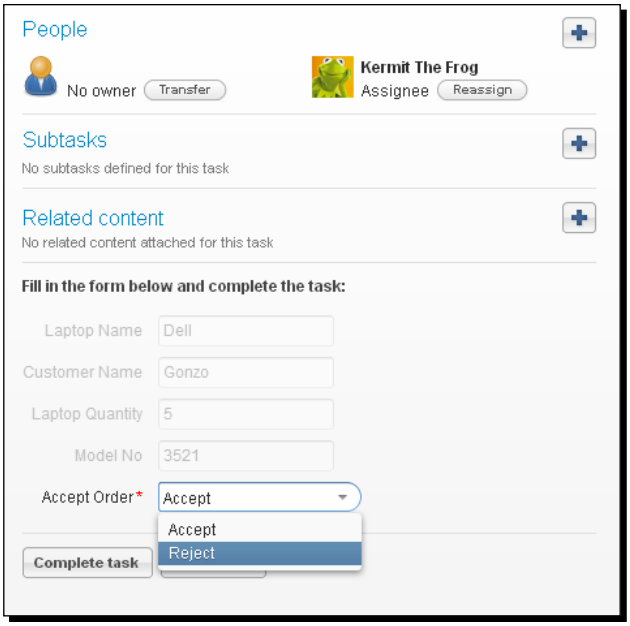
We have started the instance of the laptop order process as a `gonzo` user. Now, the approval task is assigned to the `kermit` user. So, let's complete that task by performing the following steps:

1. To complete the laptop order process, we will log in with the user ID and password as `kermit`, as shown in the following screenshot:

2. Once you log in as `kermit`, you will be redirected to the **Inbox** menu; if not, select the **Tasks** tab and then select the **Inbox** menu. The **Inbox** menu will have the number of tasks assigned to the user, as shown in the following screenshot:

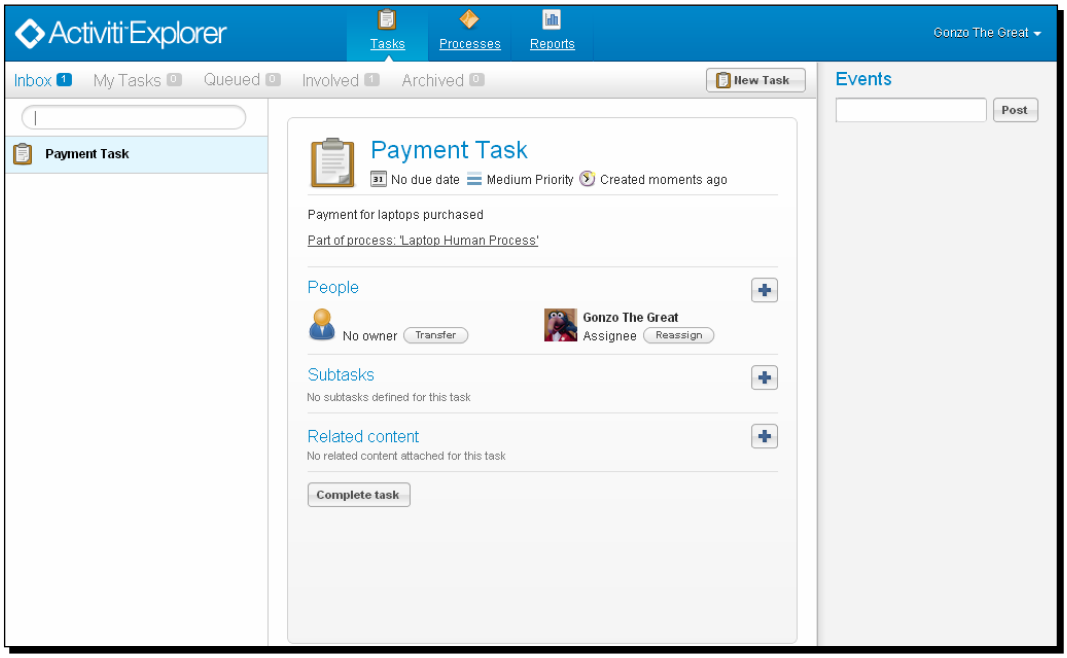


3. `kermit` can either accept or reject the order as per the information provided. If the order is accepted, `gonzo` will get the payment task in his inbox.

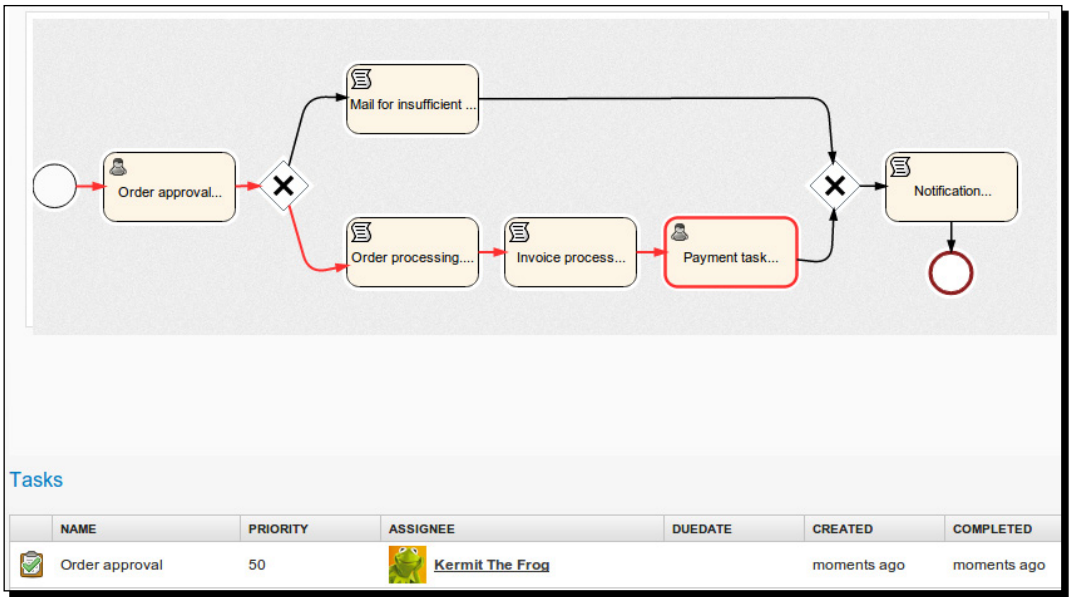


4. We will choose the **Accept** option for the order task. Therefore, `gonzo` will have his inbox populated with the payment task.

5. We will log in with `gonzo` to execute the payment task.



6. Before executing the payment task, we will view the state of our business process. Select the **Processes** tab and open the **My instances** view so that we can see the current status of the process highlighted with a red line.



7. Now, complete the payment task. The whole process is now completed.

What just happened?

We have completed the approval task that was assigned to `kermit`. We have also seen how we can see the current status of the process using the **My Instance** tab. At the end, we completed the payment task as a `gonzo` user.

Managing processes with the Activiti Explorer

We can easily manage processes using the Activiti Explorer. We can deploy a process that we have developed and we can also create new processes using the Activiti Modeler within the Activiti Explorer. It is also possible to change the business workflow after the deployment of a process using the Activiti Modeler. So, let's see how we can manage processes in the Activiti Explorer.

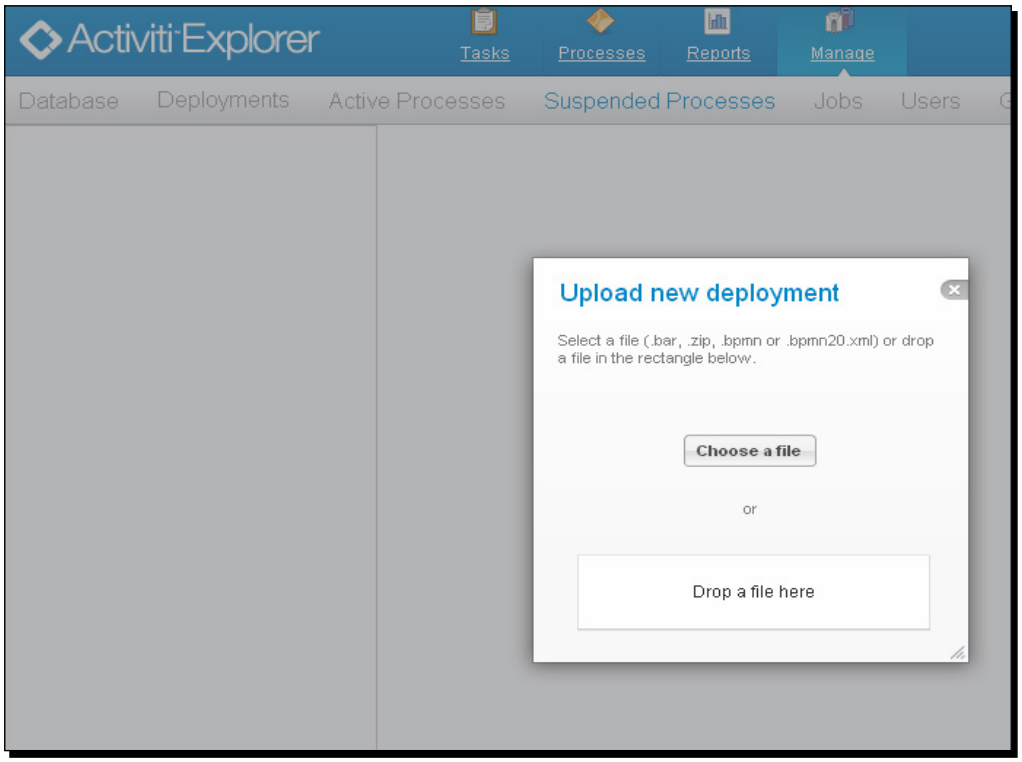
Time for action – process management

Here, we will show you the various different ways you can manage your process:

1. Log in as the admin user. Here, the admin user is `kermit`.
2. Click on the **Processes** tab. We will get a list of processes that are available by default and that we have deployed previously, as shown in the following screenshot:

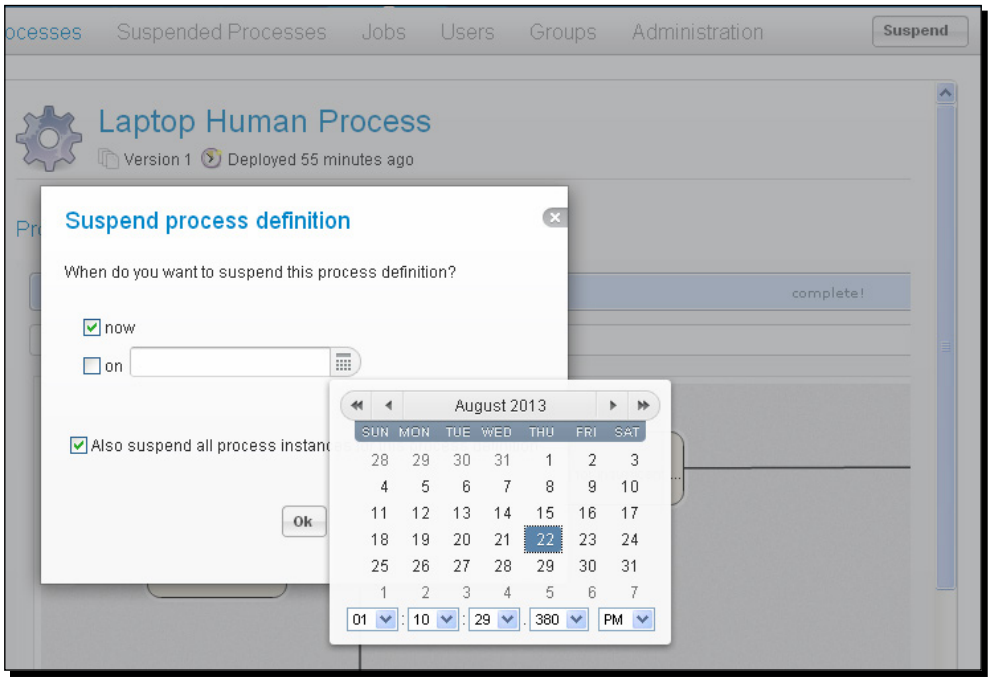
3. If we want to start any process, we have to select that particular process from the process list and click on the **Start process** button.

4. To deploy a process on the Explorer, we have to click on the **upload new** option by navigating to **Manage** | **Deployments**. It will ask for a `.bar`, `.zip`, `.bpmn`, or `bpmn20.xml` file as shown in the following screenshot:



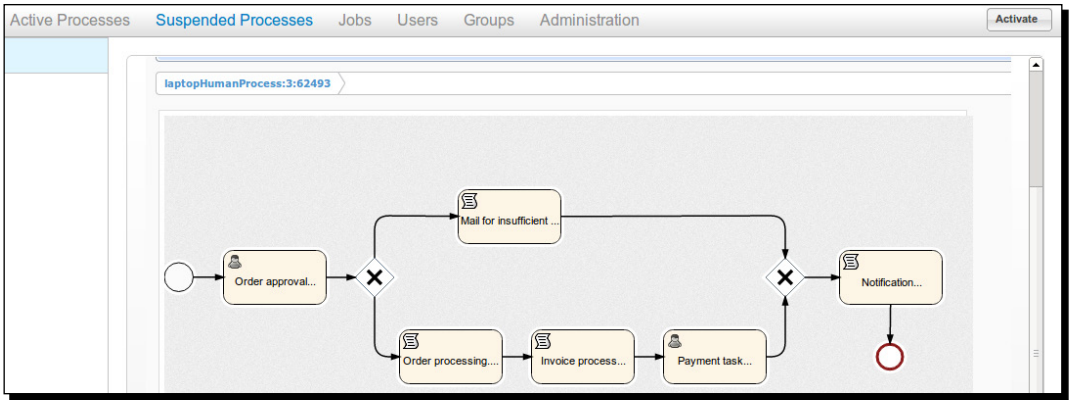
5. If we want to suspend a particular process at a particular time or date, we have to carry out the following steps:

1. First go to the **Active Processes** tab and, select the process that is to be suspended. Here, we will select the laptop order process.
2. Now click on the **Suspend** button. It will ask you when you wish to suspend the process, as shown in the following screenshot:



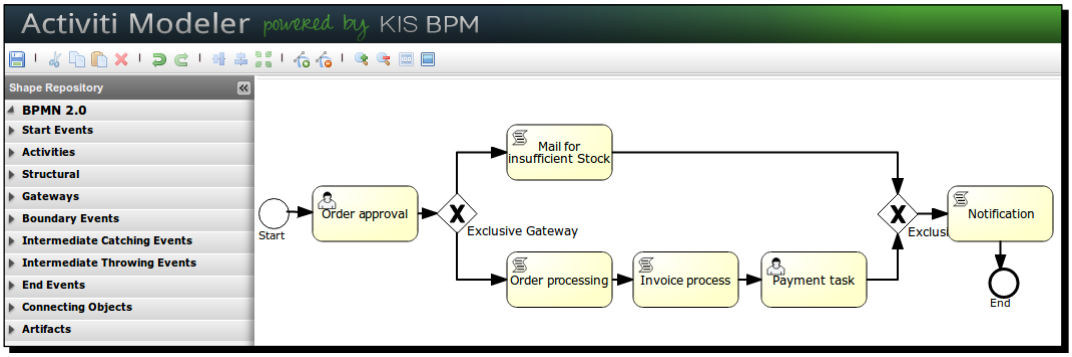
3. We can select a date from the calendar that is displayed. Choose **now** so no user can start the process for the laptop order approval.

4. Now, we can see the laptop approval process in **Suspended Processes**. You can also activate that process by clicking on the **Activate** button, as shown in the following screenshot:

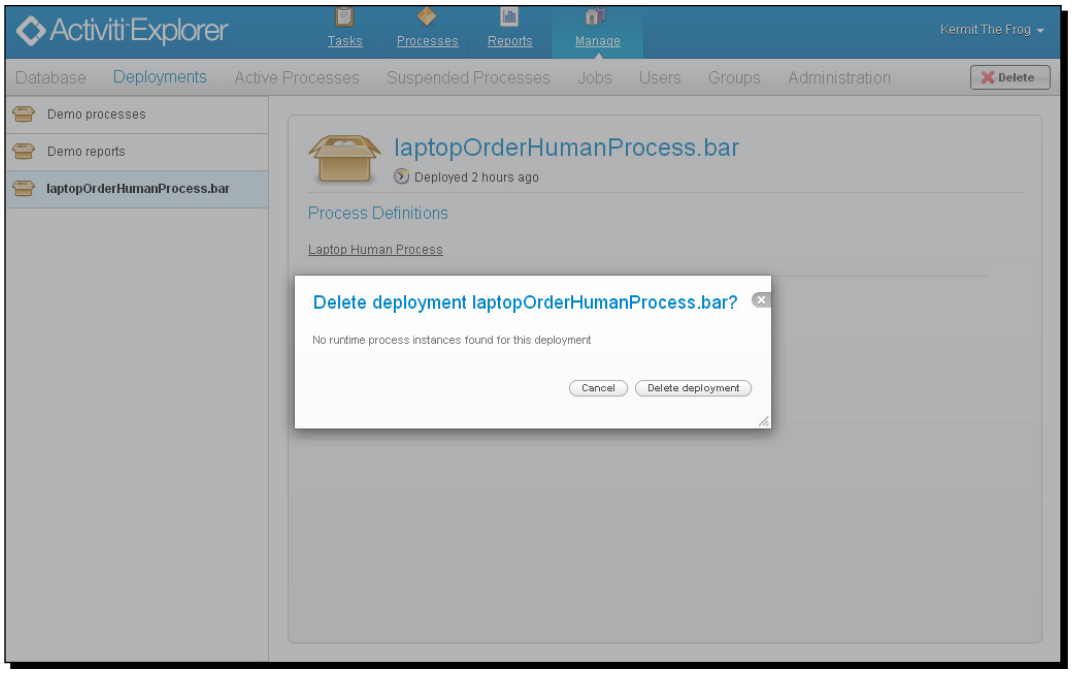


6. We can make changes to deployed processes using the Activiti Modeler. For that, we have to select the process from **Deployed Process Definitions** and click on the **Convert to editable model** button in the top-left corner.

7. We can use the Activiti Modeler to make changes to the current process, as you can see in the following screenshot:



8. If we want to delete the deployed process, we have to select the process from the **Deployments** tab. Then, click on the **Delete** button, as shown in the following screenshot:



What just happened?

In this section, we have seen how we can start a process. Also, we have seen how we can suspend a particular process using a calendar for a specific date. Finally, we saw how to delete a deployed process.

Reporting with the Activiti Explorer

The Activiti Explorer has the functionality of reporting. It has a separate tab for reporting and comes with some examples for the purposes of demonstration.

When we click on the **Reports** tab, we get the following two subtabs:

- **Generate reports**: This generates the reports known to the system
- **Saved reports**: This shows a list of previously saved reports

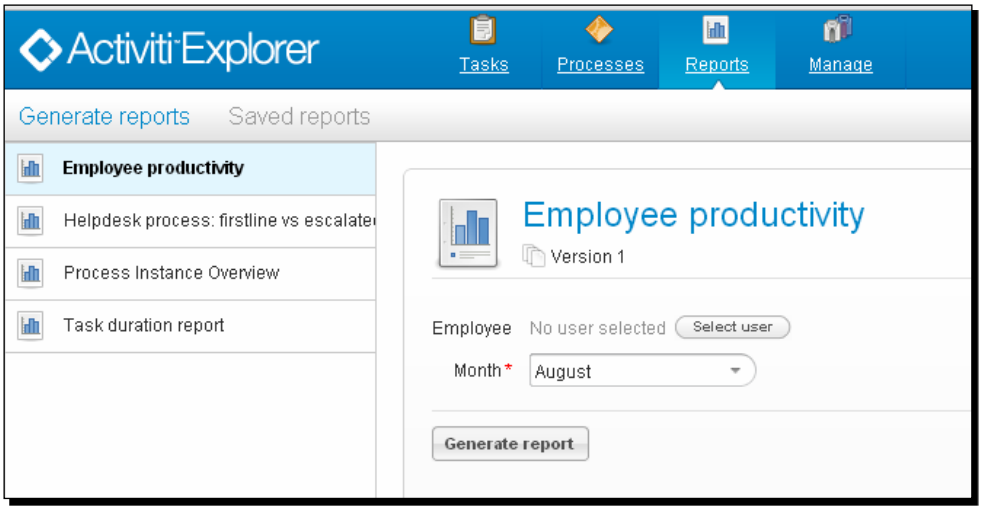
The Activiti Explorer also contains some example reports, which are as follows:

- **Task duration report**: This report uses a start form and variables to build a SQL query dynamically.
- **Employee productivity**: This demonstrates the use of a line chart and a start form. In this report, the data is fetched and interpreted by the script stored in the report data.
- **Process instance overview**: This is an example of a report where multiple datasets are used. The report contains a pie chart and a list view of the same data, thereby showing how multiple datasets can be used to generate a page with various charts.

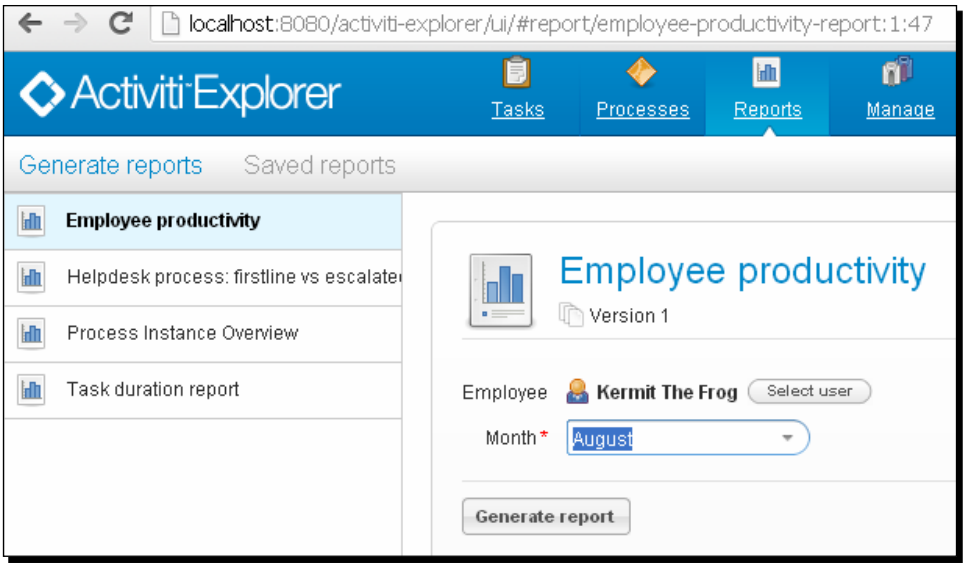
Time for action – reporting using the Activiti Explorer

Using the reporting functionality, we can generate reports for a particular process. We can also generate reports for employee productivity using a user ID. We will generate reports for the laptop order approval process by performing the following steps:

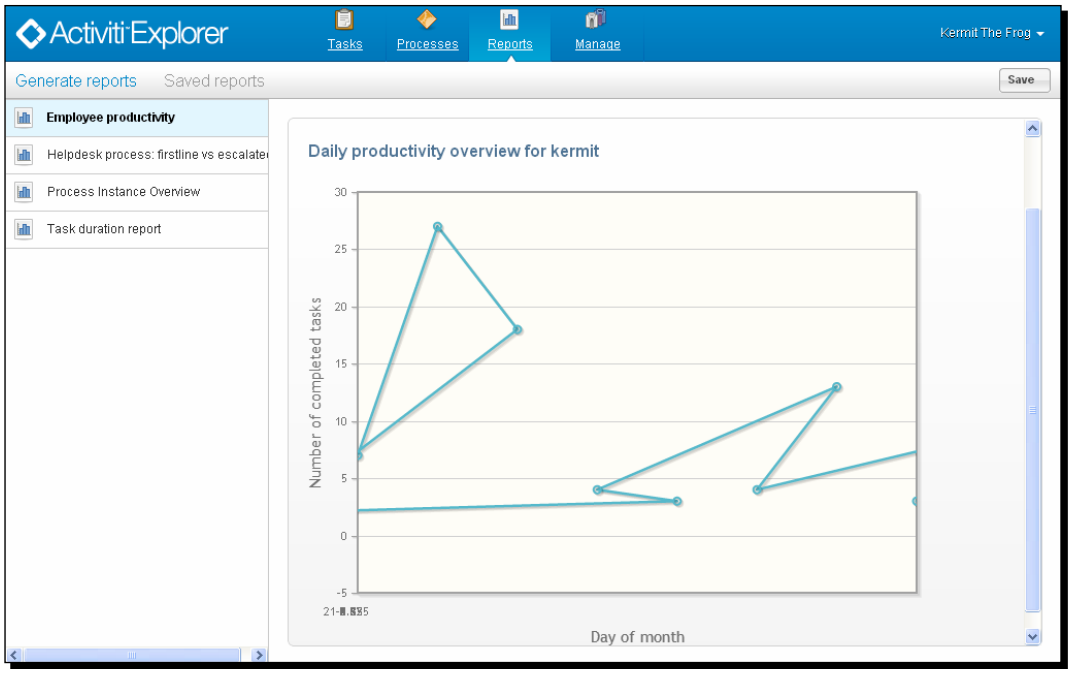
1. To have access to reports, we have to log in with the admin user. Here, we will log in as the `kermit` user.
2. There is a **Reports** tab available for generating reports. Within this tab, there are various categories of reports that can be generated, such as **Employee productivity** and **Process Instance Overview**, as you can see in the following screenshot:



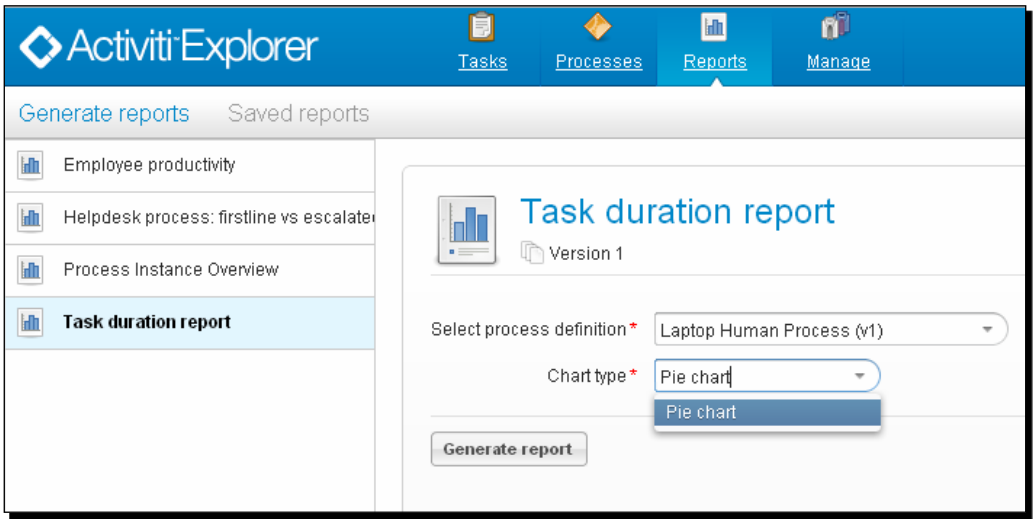
3. We will start with the **Employee productivity** report. We will generate a report for the `kermit` user. Select **Kermit The Frog** as **Employee** and the month for which the report is to be generated as shown in the following screenshot:



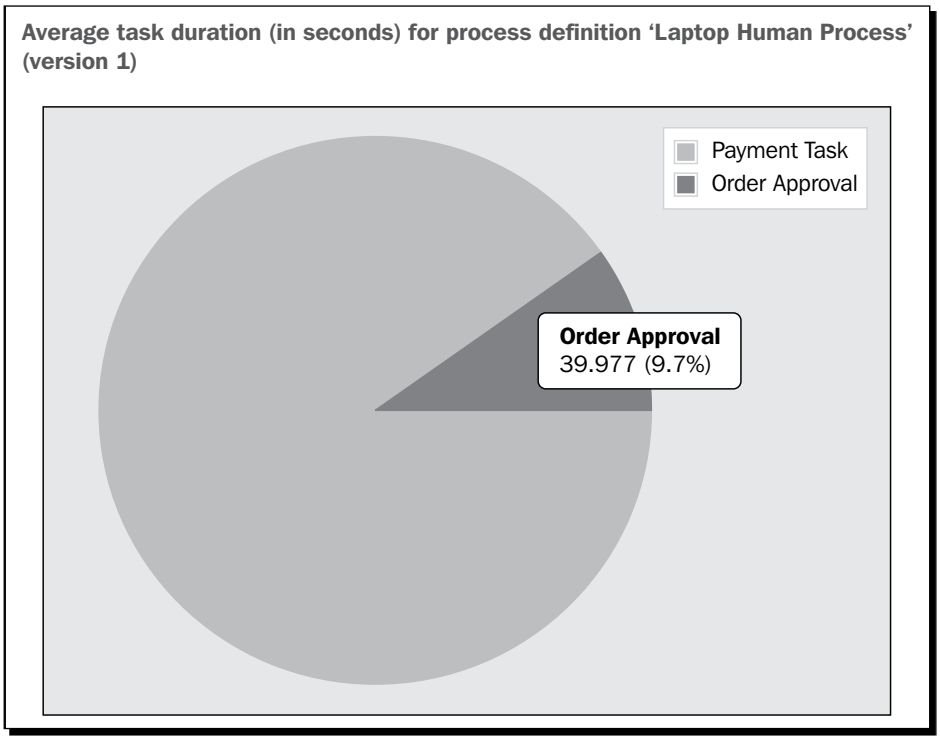
4. Now, when we click on **Generate report**, we will get a graph for the entire completed task, as shown in the following screenshot:



5. To generate a task-related report, select **Task duration report** and then the name of the process used to generate the task report.
6. First, we will start the instance of the laptop order process.
7. Select **Task duration report** and then the **Laptop Human Process(v1)** definition. From the **chart type** drop-down list, select **Pie chart** or **Bar chart,** as seen in the following screenshot:



8. This will generate a chart for the business process based on the execution of the process.



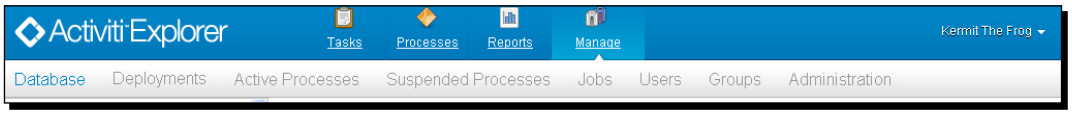
What just happened?

We have now covered the functionality of reporting. We have generated a report for employee productivity based on the user and the month. We have also seen how we can generate a task duration report for a particular process.

Administration using the Activiti Explorer

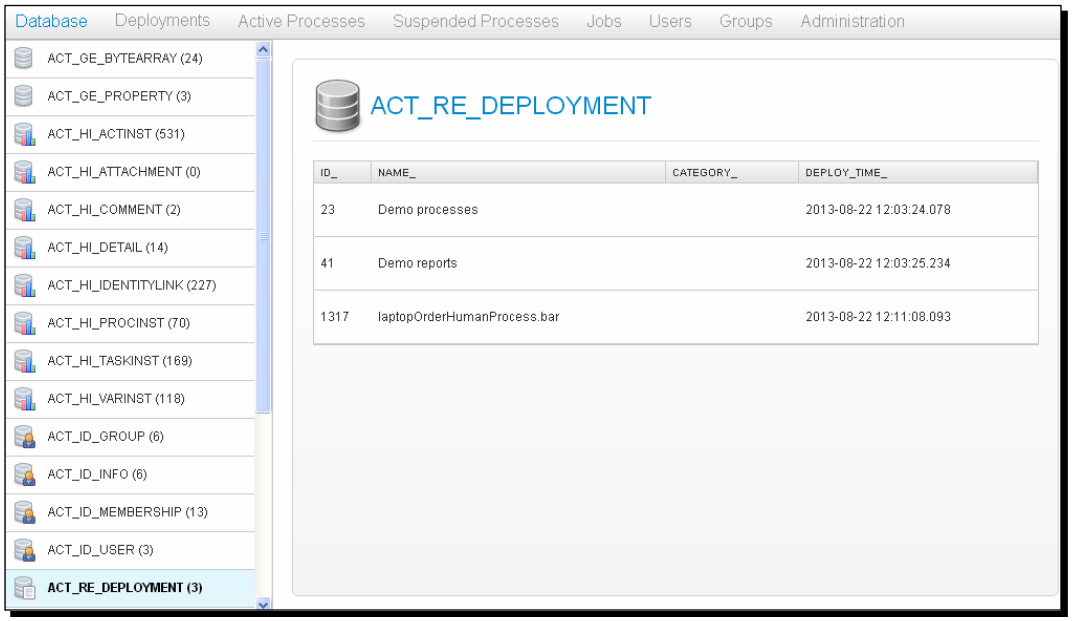
The Activiti Explorer also provides the functionality of administration. `kermit` is part of the security group admin. So, he can access **Administration** from the Explorer.

For the admin, there is a **Manage** tab that contains submenus, as shown in the following screenshot:

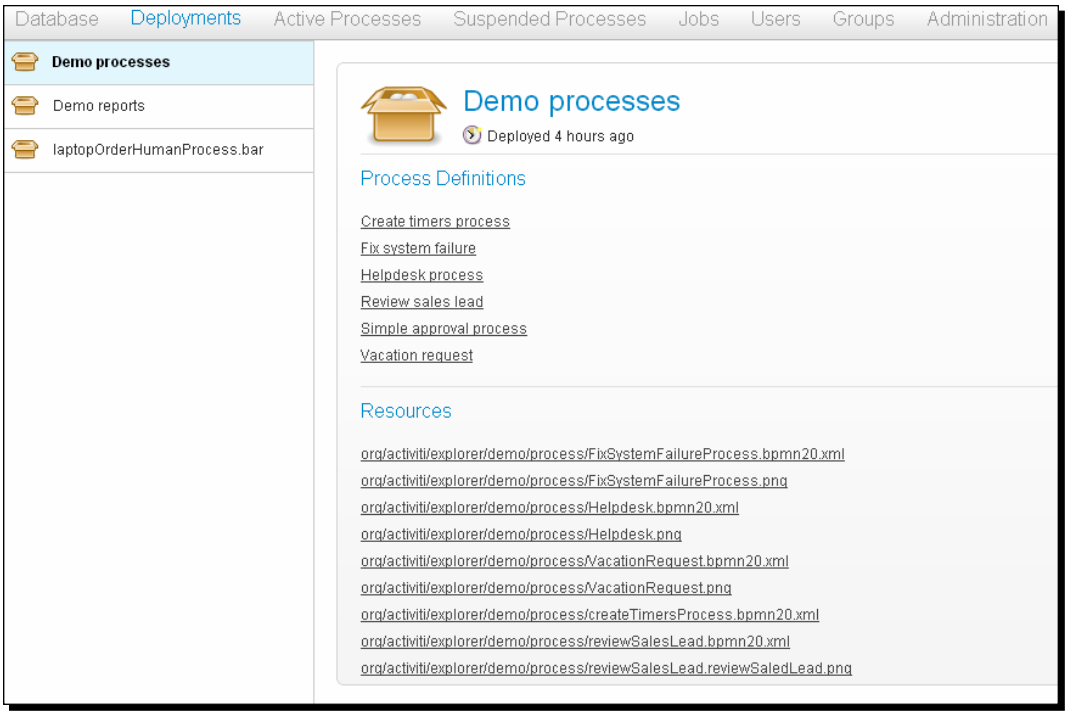


The following are the different submenus in the **Manage** tab:

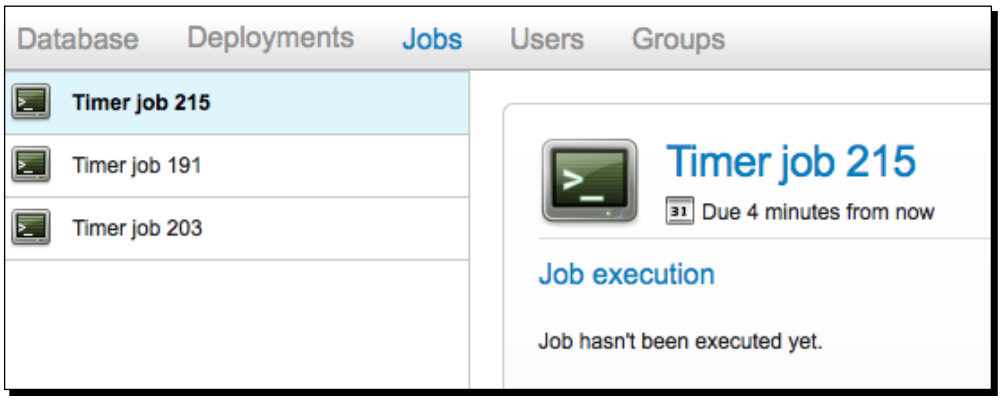
- **Database**: This tab is used to display the database tables and the content of the database. It is very useful for troubleshooting problems.



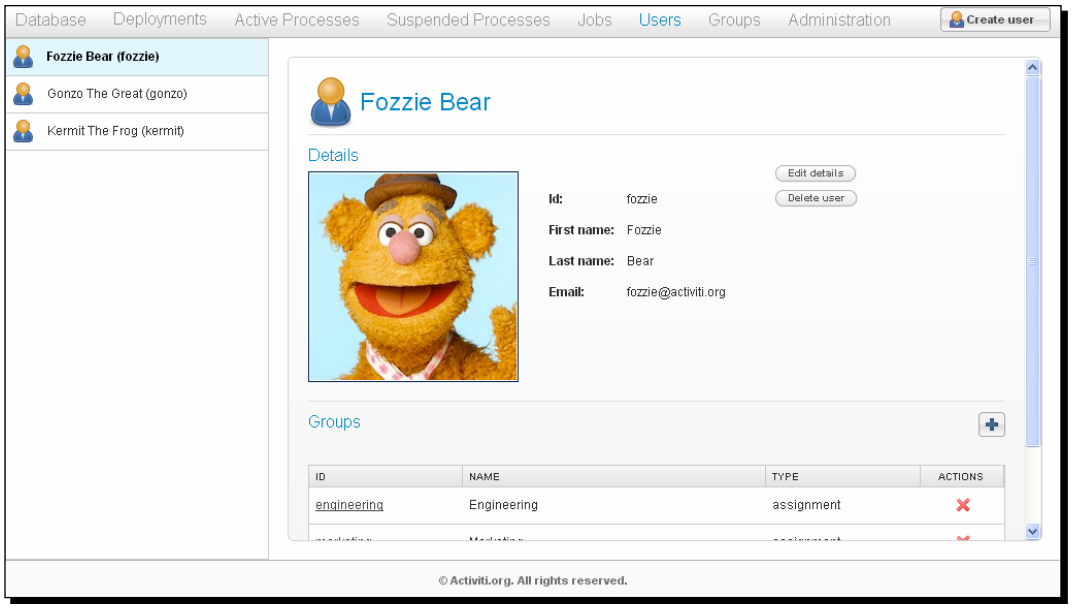
- **Deployments**: This tab is used to display the deployments of the engine. We can also see the process definitions and the resources. Additionally, we can upload a new process definition by uploading a `.bar` file of the process.



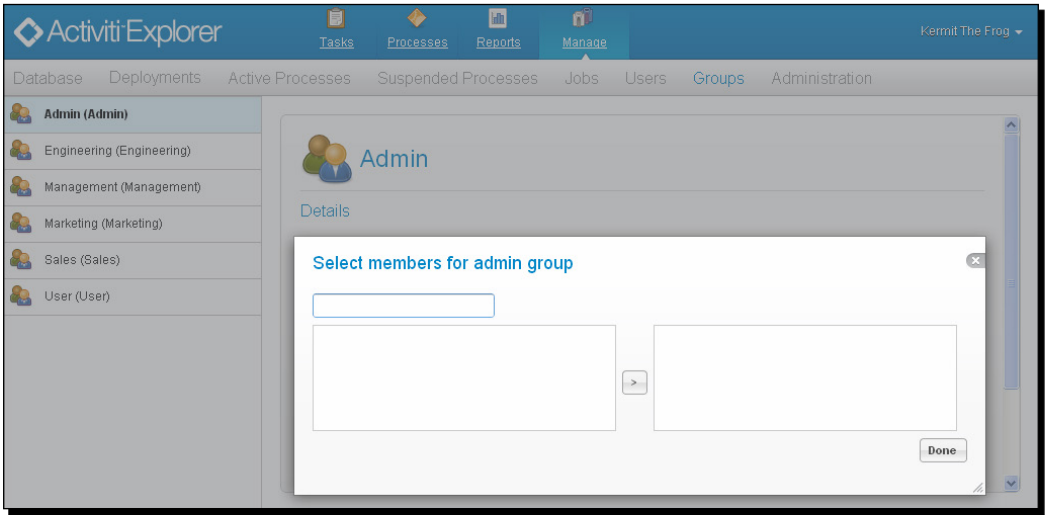
- **Jobs**: This tab is used to display the asynchronous tasks and allows us to manually execute them.



- **Users**: This tab is used for the user management functionality. We can create a new user, edit the existing details of a user, and also delete users. Furthermore, we can also assign a particular group to a user.



- **Groups**: This tab manages the functionalities of a group; that is, we can add, edit, and delete groups and assign members to them as well.

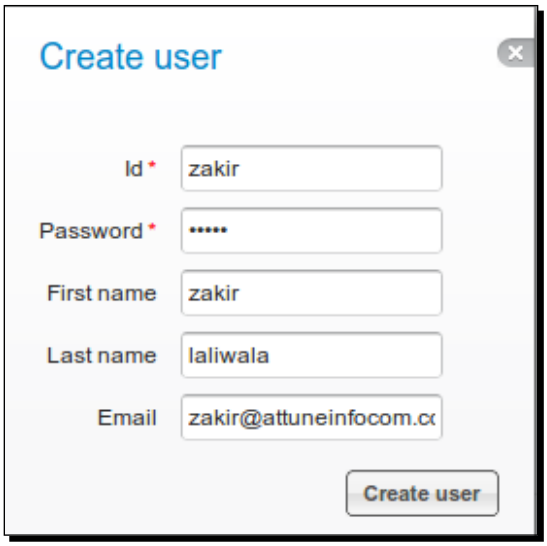


Time for action – administration using the Activiti Explorer

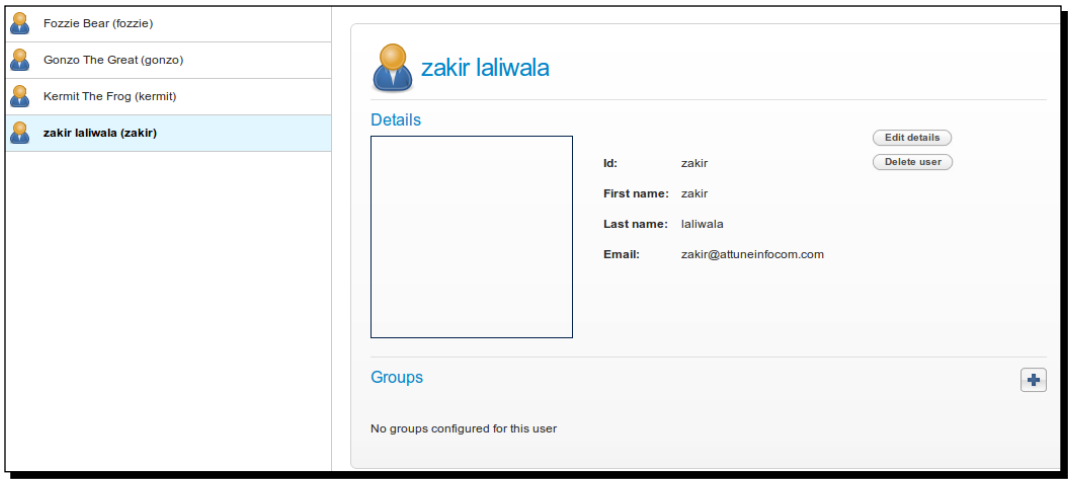
User management is an easy task using the Activiti Explorer. We can create a new user from the **Users** tab. Let's create a new user and assign it to the admin group using the following steps:

1. We will now create some new users in the Activiti Engine. In order to do this, click on the **Users** tab and then on the **Create user** button as shown in the following screenshot:

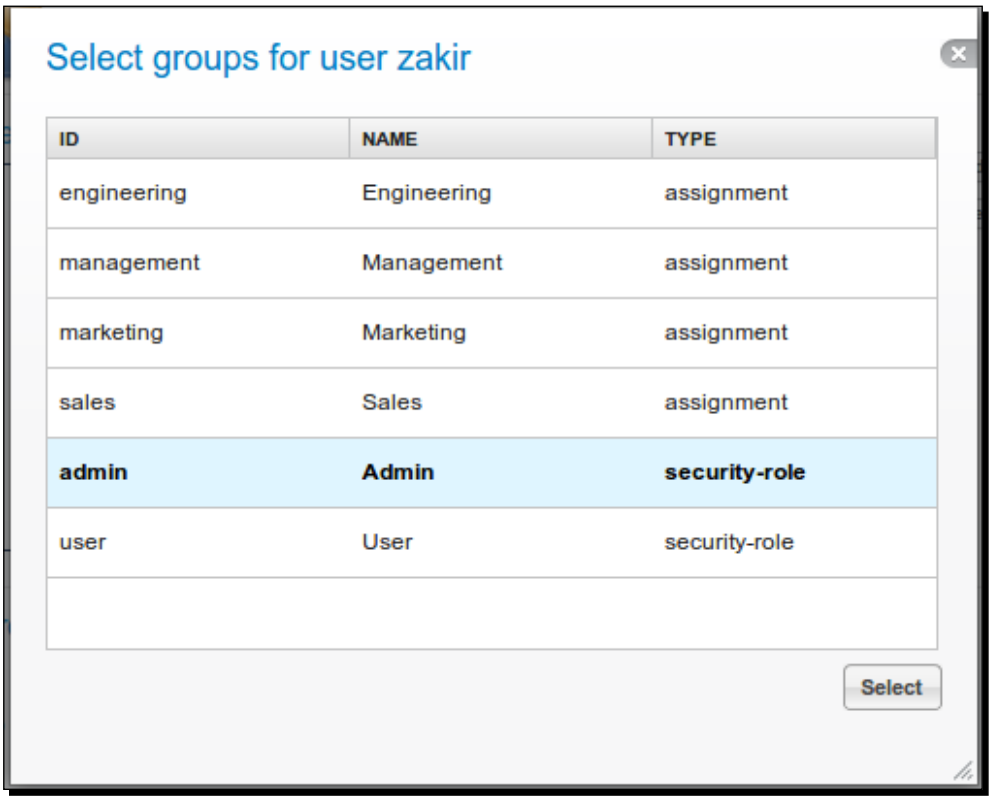
2. This button will pop up a form to be filled to create a new user; provide the information as mentioned in the following screenshot:



3. After filling in the information, click on the **Create user** button. Now, the new user will be available in the users menu as shown in the following screenshot:



4. As you can see in the preceding screenshot, the user is not part of any of the groups. So, we will assign this user to a group. Click on the **+** sign under the **Groups** section and a list of groups will be available, as you can see in the following screenshot:



5. We can select any of the groups from this list for the user; here, we will select the **Admin** group. The user is now part of the **Admin** group.

What just happened?

We have created a new user and also seen how we can assign the user to a particular group.

Time for action – changing to a production-ready database

Activiti comes with a default database, `H2`. This database is only used for the purposes of demonstration. For production, we have to change the database. So, let's see how we can configure the database:

1. To configure Activiti with a database, we have to create a schema named `MySQL`.
2. We will also create another schema named `activitiin mysql`.
3. Now, we have to configure Activiti with MySQL. In order to do this, we have to browse to `apache-tomcat-7.0.27/webapps/activiti-Explorer/WEB-INF/classes/db.properties`.
4. Make the following changes in `db.properties`:

```
db=activiti

jdbc.driver=com.mysql.jdbc.Driver

jdbc.url=jdbc:mysql://localhost:3306/activiti

jdbc.username=root

jdbc.password=root
```

5. Also, we have to place `mysql-connector-java-5.1.18-bin.jar` into `apache-tomcat-7.0.27/webapps/activiti-Explorer/WEB-INF/lib`.
6. After performing these steps, we have to restart the Tomcat server if it is already running. If not, just start the Tomcat server.
7. Once the Tomcat server has started successfully, the Activiti schema will be populated with the default tables of Activiti.

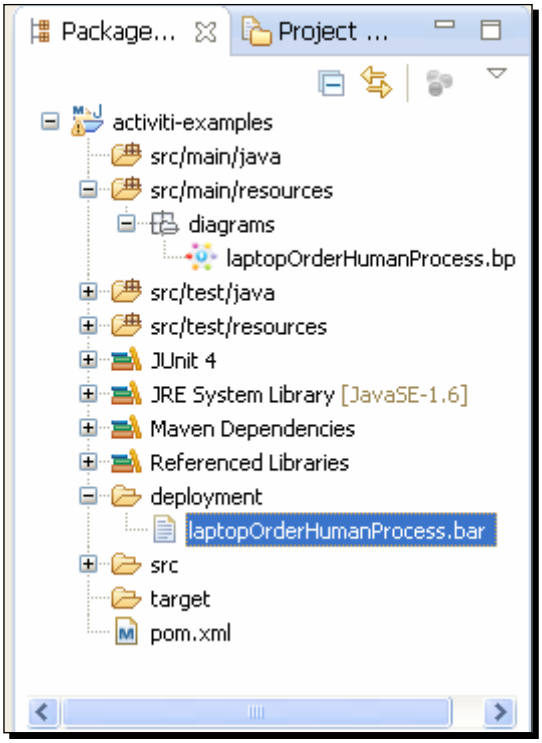
What just happened?

In this section, we saw how to set up a production database in Activiti. We have also seen how to make changes in `db.properties` and learned how to populate all the tables into the MySQL database.

Time for action – deploying a process using the Activiti Explorer

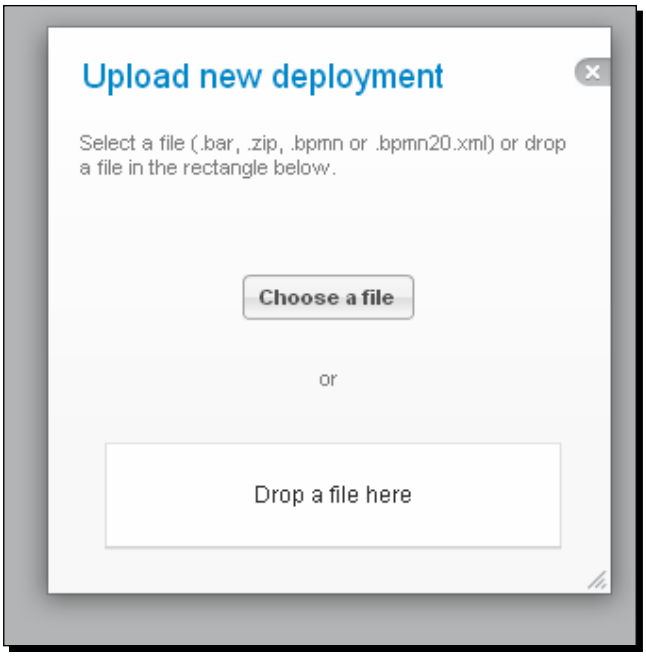
The Activiti Explorer provides a tab for deploying processes. To deploy a process into the Activiti Explorer, we have to generate a `.bar` file for the process from Eclipse. In order to do this, perform the following steps:

1. To generate a `.bar` file, open the laptop human process example in Eclipse.
2. Your project should be opened in the **Package Explorer** view; only then will you be able to create a `.bar` file.
3. In **Package Explorer**, right-click on the project and select the **Create deployment artifacts** option.
4. There should now be a `.bar` file in the `deployment` folder (it is named `laptopOrderHumanProcess.bar` in the following example screenshot):



5. Now, to upload the `.bar` file in the Activiti Explorer, log in as the admin user, browse to the **Manage** tab, and select **upload new** under the **Deployments** menu.

6. The **upload new** option will pop up a window in which we can upload the `.bar` file by using either the **Choose a file** or **Drop a file here** option, as shown in the following screenshot:



7. Once the process file is successfully uploaded, it will be deployed into the Activiti Explorer and we can start the process from the **Deployed process** menu in the **Processes** tab.

What just happened?

In this section, we saw how to generate a `.bar` file from Eclipse and have also deployed a `.bar` file into the Activiti Explorer. So, now we can deploy the process into the Explorer.

Have a go hero

Having gone through the chapter, feel free to attempt the following tasks:

- Configure Activiti with any other database that you are using
- Assign a task to a particular group instead of only one user
- Execute a process and check each table in the Activiti database

Pop quiz – the Activiti Explorer

Q1. Which user can access the **Admin** tab?

1. A guest user
2. A normal user
3. Whoever is part of the admin user group

Q2. How can you monitor the current process graphically?

1. Using the **Model workspace** tab
2. Using the **My instances** tab
3. Using the **Active processes** tab

Q3. What is the extension of the file that is used for deployment?

1. `.jar`
2. `.bar`
3. `.zip`

Q4. Which file is used to configure the database in Activiti?

1. `Web.xml`
2. `db.properties`
3. `activiti-standalone-context.xml`

Q5. Where is the Historical task available?

1. The **Inbox** tab
2. The **My Tasks** tab
3. The **Archived** tab

Summary

We have played with the Activiti Explorer in this chapter. We have seen the various functionalities of the Activiti Explorer and know how we can deploy a process and start an instance. We have seen the functionalities that Activiti Explorer provides for the management of users and groups. We have also seen how we can set up a production database with Activiti. In the next chapter, we will learn how to set up a development environment for Activiti. We will also learn how to configure the mail server settings for the mail tasks in Activiti.

5
Development Using the Process Engine

In the previous chapter, we learned about the different functionality of the Activiti Explorer. We learned how to deploy and start a process and monitor the current process. The Process Engine is the heart of Activiti. The Process Engine of Activiti contains different layers. Each layer has a different functionality. In this chapter, we will take a look at how to configure the Process Engine for development and using a mail task to send mails. The rest of the topics that will be covered in this chapter are mentioned below.

This chapter covers the following topics:

- The development environment
- The Process Engine's configuration
- The logging configuration
- The mail server configuration

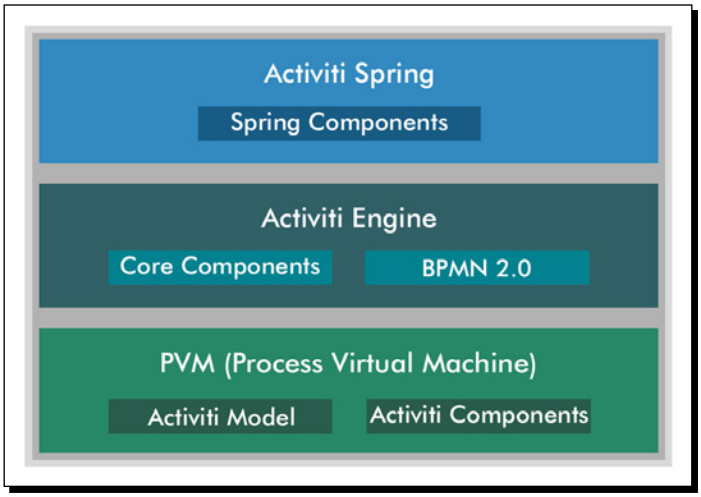
Each layer of the Process Engine plays a major role in Activiti. Before looking at the Process Engine, let's take a look at the Activiti development environment.

Understanding the Activiti development environment

The Activiti Engine contains various libraries. Each library has a unique functionality. The following is a detailed explanation of each layer of the Activiti Engine:

- The Activiti Engine has a Spring layer at the top. So, we can take advantage of the Spring framework within Activiti itself.
- The second layer is the Activiti Engine that contains components for the core functionality of Activiti and the specification for BPMN 2.0.
- The third layer is the **PVM** (**Process Virtual Machine**), which is capable of converting a graphical representation to the state model.

All these layers are shown in the following diagram:



The Activiti Spring layer

Activiti supports the Spring functionality, which is an optional layer. We can take advantage of Spring in Activiti. The Process Engine can be configured as the Spring bean. We can start the Spring integration using `org.activiti.spring.ProcessEngineFactoryBean`. For the configuration of `activiti-spring-version`, the JAR file available in the Activiti lib is useful. We can configure Persistency using the Spring framework in Activiti. Activiti libraries contain `spring-jdbc-version.jar`, `spring-orm-version.jar` and `spring-web-version.jar` for various functionality using Spring.

The Activiti Engine layer

Activiti Engine is a core layer of Activiti. It provides engine interfaces to achieve the core functionality of Activiti. Major functionality is provided by the Engine layer. This layer contains core interfaces and the BPMN 2.0 specification. We have to add the required libraries to the build path to use the Activiti Engine in the development environment.

PVM

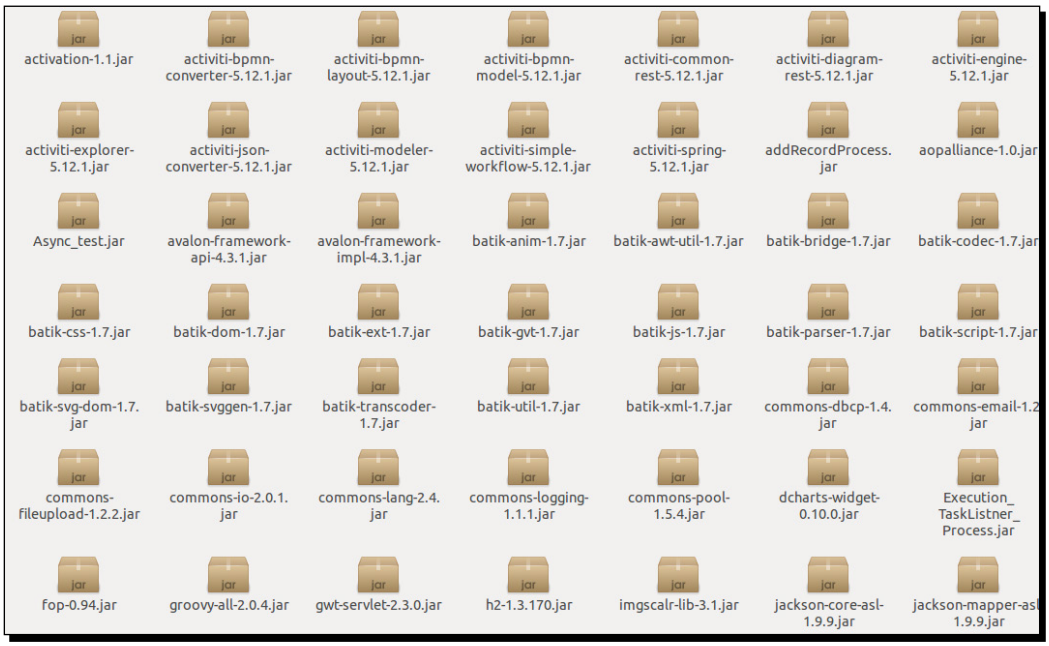
Process Virtual Machine is embedded with the Activiti Engine. It is responsible for converting the BPMN 2.0 engine stuff to the state model of the Activiti Engine.

Setting up the Activiti development environment

In *Chapter 1*, *Installing Activiti*, we saw how we can install the Activiti Explorer on a web server. In *Chapter 2*, *Modeling Using the Activiti Modeler*, we went through using the Activiti Modeler for designing workflows and in *Chapter 3*, *Designing Your Process Using the Activiti Designer*, we went through using the Activiti Designer to design the workflows. However, to understand how it actually works, we need to understand the workings of the development environment behind the scenes.

As mentioned earlier, the Activiti Engine is a separate layer that can be used independently. It consists of the engine interface, which we will be covering in *Chapter 6*, *The Activiti ProcessEngine API*, and also implements the BPMN 2.0 specification. The next layer is the process virtual machine layer, which supports the other processes' languages. The entire engine component is implemented using the `activiti-engine-version.jar` file.

Hence, to deal with the development, we have to use `activiti-engine.version.jar`. In *Chapter 1*, *Installing Activiti*, we have already deployed Activiti in a Tomcat web server. So, this JAR file will be available within the path `apache-tomcat-7.0.37/webapps/activiti-explorer/WEB-INF/lib`, as shown in the following screenshot:



All the libraries required for development will be available within this folder. If you don't want to use the JAR file, you can use Maven. Activiti, by default, creates a Maven project; it makes dependency management easy. You have only configured your `pom.xml` file with the required dependencies as in the following code:

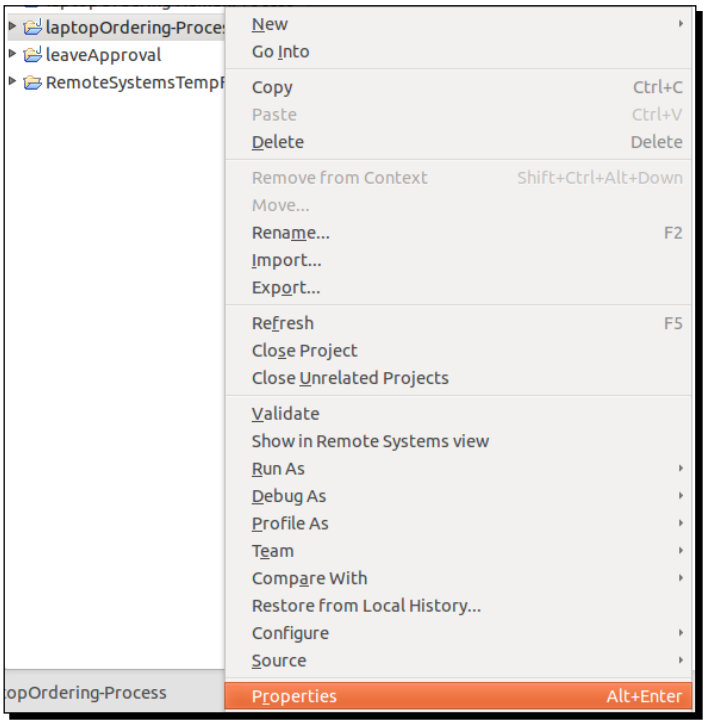
```
<project xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://
www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.
apache.org/maven-v4_0_0.xsd">
<modelVersion>4.0.0</modelVersion>
<groupId>org.activiti.examples</groupId>
<artifactId>activiti-examples</artifactId>
<version>1.0-SNAPSHOT</version>
<packaging>jar</packaging>
<name>BPMN 2.0 with Activiti - Examples</name>
<properties>
<activiti-version>5.9</activiti-version>
</properties>
<dependencies>
<dependency>
```

```
<groupId>org.activiti</groupId>
<artifactId>activiti-engine</artifactId>
<version>${activiti-version}</version>
</dependency>
<dependency>
<groupId>com.h2database</groupId>
<artifactId>h2</artifactId>
<version>1.2.132</version>
</dependency>
</project>
```

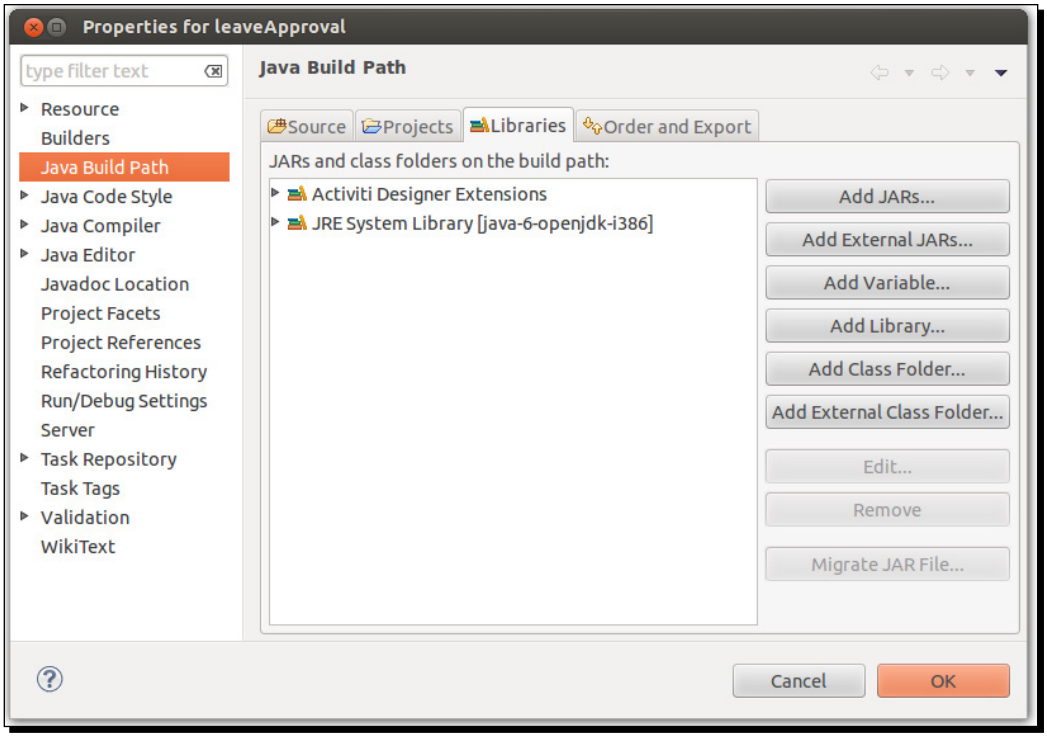
In the preceding `pom.xml` file, you can see that there is an `activiti-engine` dependency for the Activiti Engine and an `h2` dependency for configuring the process with an in-memory `H2` database. In this case, if you want to perform unit testing for your development, you need to add the unit testing dependency as well.

If you are not comfortable with Maven, don't worry; all the JAR files are available in the `lib` folder, so you can import them to your Eclipse project. For that, you will have to perform the following steps:

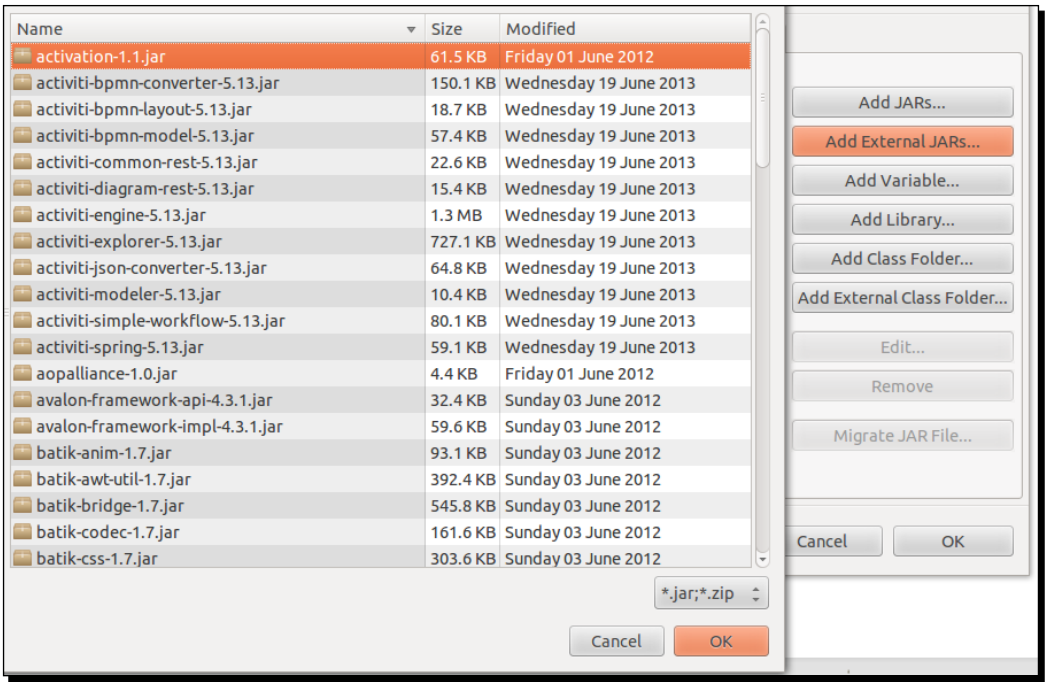
1. Select the project for importing libraries, right-click on it, and browse through the **Properties** option. We can select the **Properties** option as shown in the following screenshot:



2. On selecting the **Properties** option, a window will pop up. From that window, select **Java Build Path**; this displays all the libraries associated with the project. By default, it will not have any Activiti Engine libraries associated with it.



3. To add the JAR files, select **Add External JARs** in the window. On selecting this option, another window will pop up in which you need to browse to the path where your JAR files are located, for example, `/apache-tomcat-7.0.37/webapps/activity-explorer/WEB-INF/lib`. By going to the given path, you can select the JAR file required for your development.



Configuring the Activiti Process Engine

We can configure the Activiti Process Engine using an XML file known as `activiti.cfg.xml`. We have to configure a bean in `activiti.cfg.xml` for the Process Engine's configuration.

We have to define the following bean as follows:

```
<bean id="processEngineConfiguration" class="org.activiti.engine.impl.
cfg.StandaloneProcessEngineConfiguration">
```

This bean is used to generate the Process Engine. Different types of classes are available to configure the Process Engine. We can choose which class to use depending on the environment. Choose the appropriate class for your environment so that the number of properties used for configuring the engine can be minimized.

We can use the following classes:

- `org.activiti.spring.SpringProcessEngineConfiguration`: This can be used when we have configured the Process Engine in Spring.

- `org.activiti.engine.impl.cfg.StandaloneInMemProcessEngineConfiguration`: This is the appropriate class for unit testing. The `H2` in-memory database is used by default. A database will be automatically created when the engine starts and dropped when it shuts down. We don't have to put extra effort into the configuration.
- `org.activiti.engine.impl.cfg.JtaProcessEngineConfiguration`: This is used with the `Jta` transaction when the engine runs in the standalone mode.
- `org.activiti.engine.impl.cfg.StandaloneProcessEngineConfiguration`: The Process Engine is used in a standalone way. Transactions will be conducted by Activiti itself. By default, the database will only be checked when the engine boots.

We can point the Activiti Engine to the production database using the Spring environment. Let's see how we can configure the production database.

Time for action – configuring a database

In the previous chapter, we saw how to set up a production database using the `db.properties` file. Here, we will see how to set up a database using Spring.

To configure a database, we have to define the following properties in the `datasource` bean:

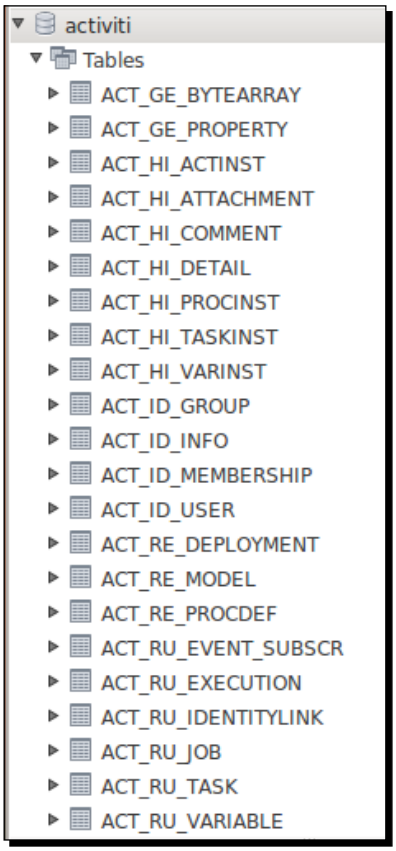
- `jdbcDriver`: This property defines a driver name for a particular database
- `jdbcURL`: This property mentions the JDBC URL of the database
- `username`: This property defines the username to connect to the database
- `password`: This property defines the password to connect to the database

Perform the following steps to configure a database:

1. Create a database named `activiti` in MySql.
2. Open the `activiti-standalone-context.xml` file from `apache-tomcat-7.0.34\webapps\activiti-explorer\WEB-INF`.
3. Make the following changes to the `datasource` bean:

```
<bean id="dataSource" class="org.springframework.jdbc.datasource.
DriverManagerDataSource">
<property name="driverClassName" value="com.mysql.jdbc.Driver" />
<property name="url" value="jdbc:mysql://localhost:3306/activiti"
/>
<property name="username" value="root" />
<property name="password" value="root" />
</bean>
```

4. Add the `Mysql-coonector-version.jar` file to the library of the Activiti Explorer.
5. If your Tomcat server is running, restart it.
6. To check whether or not the DB is populated, you can open the `activiti` database in MySql; you will find all the default tables populated, as shown in the following screenshot:



What just happened?

We have learned to configure a database using the Spring bean. In the development environment, debugging is very useful. Logging shows errors in detail so that you can solve them easily.

Time for action – the logging configuration

The logger is responsible for capturing the message to be logged along with certain metadata and passing it to the logging framework. After receiving the message, the framework calls the formatter with the message. The formatter formats it for output. The framework then hands the formatted message to the appropriate appender for disposition.

Activiti uses the SLF4J logging framework. To use the SLF4J logging framework in Activiti, we need to add `SFL4J-binding.jar` to our project. By default, Activiti uses NOP-logger; it will not log anything other than a warning.

To add the `log4j12` dependency to Maven, open your Activiti project and add the following code to the `pom.xml` file:

```
<dependency>
<groupId>org.slf4j</groupId>
<artifactId>slf4j-log4j12</artifactId>
<version>1.7.5</version>
</dependency>
```

The Log4j binding is configured with the Activiti Explorer and Activiti REST web apps. To use Log4j in Activiti, create the `log4j-config.xml` file using the following code:

```
< ?xml version="1.0" encoding="UTF-8" ?>

<!--<span class="hiddenSpellError" pre=""-->DOCTYPE
log4j:configuration SYSTEM "log4j.dtd">
 xmlns:log4j="http://jakarta.apache.org/log4j/">
<appender name="console" class="org.apache.log4j.ConsoleAppender">
<param name="Target" value="System.out"/>
<layout class="org.apache.log4j.PatternLayout">
-5p %c %x - %m%n"/>
</layout>
</appender>
<appender name="fileAppender" class="org.apache.log4j.
RollingFileAppender">
<param name="File" value="demoApplication.log"/>
<layout class="org.apache.log4j.PatternLayout">
-5p %c %x - %m%n"/>
</layout>
</appender>
<root>
<priority value ="debug"></priority>
<appender -ref ref="console"></appender>
<appender -ref ref="fileAppender"></appender>
</root>
```

The `Log4j-config.xml` file has all the runtime configurations used by Log4j. This file will contain the `appender` information, log-level information, and output filenames for file appenders.

What just happened?

We have seen the logging configuration. Now we know which dependency is required to use the log framework in Activiti.

Time for action – configuring the mail server

We can send an e-mail using Activiti. Activiti provides an e-mail task to send a number of e-mails. It can be possible using an external mail server with SMTP. The e-mail task is implemented as a service task in Activiti. To send an e-mail, we have to configure some properties of e-mails for the Activiti Engine.

To send real e-mails, we have to configure the following properties in `activiti-standalone-context.xml`:

- `mailServerHost`: This property is used to define the hostname of your mail server
- `mailServerPort`: This property is used to define a port for the SMTP server
- `mailServerDefaultFrom`: This property is used to set a default e-mail address
- `mailServerUsername`: This property is used to specify a username for an authentication
- `mailServerPassword`: This property is used to specify a password for your mail
- `mailServerUseSSL`: This property is required for SSL communication

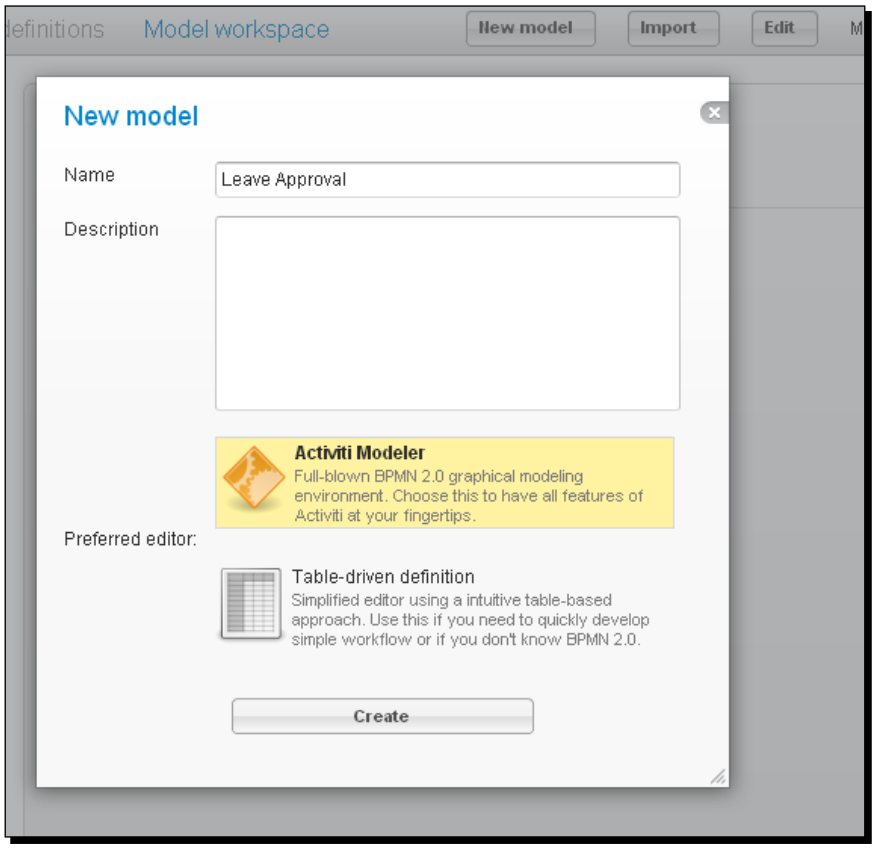
Let's see an example using an e-mail task by performing the following steps:

1. Open `activiti-standalone-context.xml` from `apache-tomcat-7.0.33\webapps\activiti-explorer\WEB-INF\` and add the following properties:

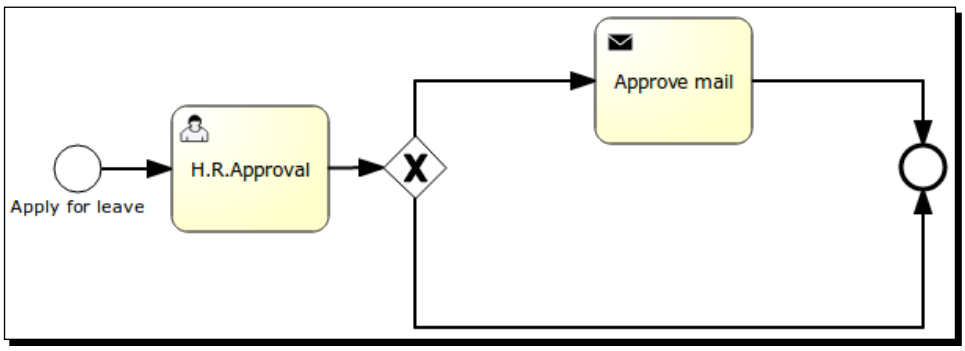
```
<bean id="processEngineConfiguration" class="org.activiti.spring.
SpringProcessEngineConfiguration">
.....
<property name="mailServerHost" value="smtp.gmail.com" />
<property name="mailServerPort" value="465" />
<property name="mailServerUsername" value="giveyouremailidhere" />
<property name="mailServerPassword" value="*******"/>
<property name="mailServerUseSSL" value="true"/>
```

2. As we are configuring the properties file, we have to restart the server.

3. Design a process for `Leave Approval` using the Activiti Modeler, as shown in the following screenshot:

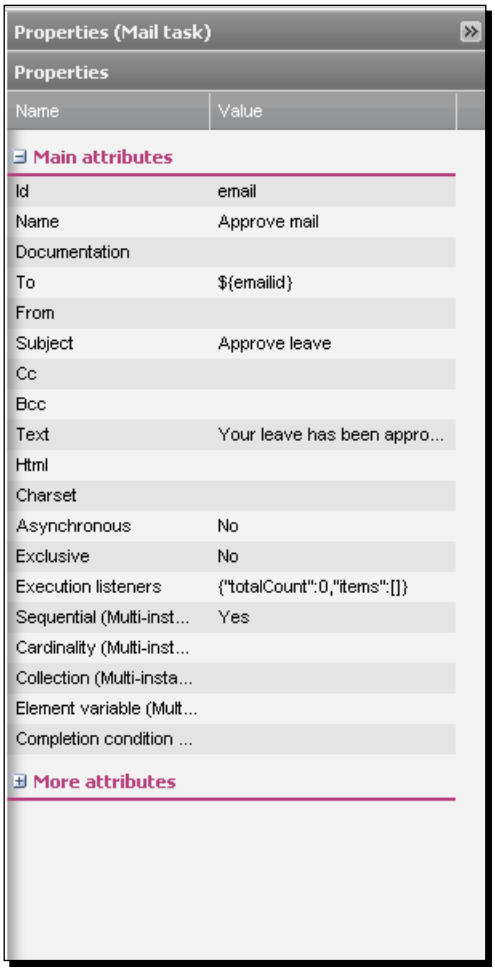


4. We will design the process as shown in the following flowchart:



5. The user starts the process and fills in some information. Then the **H.R.Approval** task is assigned to the `kermit` user. When `kermit` approves, an approval mail will be sent to the user who started the process.

6. For mail configuration, we have to populate some properties to send an e-mail, as shown in the following screenshot:



The properties of an e-mail task contain `To`, `From`, `Cc`, `Bcc`, `Text`, and `Html`. We have defined `To` for sending a mail to the person who started the process. We can include other recipients in the `Cc` and `Bcc`. If we want to send a simple message, we can use the `Text` property. If we want to send a hyperlink or a formatted message, we can use the HTML code within the `Html` property.

7. There is a `process.bpmn20` file in the code example of this book. You can import this file using the import model option and deploy the process as follows:

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/
MODEL" xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xmlns:activiti="http://activiti.org/bpmn" xmlns:bpmndi="http://
www.omg.org/spec/BPMN/20100524/DI" xmlns:omgdc="http://www.
omg.org/spec/DD/20100524/DC" xmlns:omgdi="http://www.omg.org/
spec/DD/20100524/DI" typeLanguage="http://www.w3.org/2001/
XMLSchema" expressionLanguage="http://www.w3.org/1999/XPath"
targetNamespace="http://www.activiti.org/test">
<process id="process" isExecutable="true">
<startEvent id="start" name="Apply for leave">
<extensionElements>
<activiti:formProperty id="name" name="name" type="string"></
activiti:formProperty>
<activiti:formProperty id="nodays" name="Total Days"
type="long"></activiti:formProperty>
<activiti:formProperty id="emailid" name="Email id"
type="string"></activiti:formProperty>
</extensionElements>
</startEvent>
```

The preceding code is for the start event node in which we are defining a form to fetch the values from the user for the number of days required for leave. So, whenever an employee starts this process, they will be asked to provide the preceding details. The code for creating a user task is as follows:

```
<userTask id="hrApproval" name="H.R.Approval"
activiti:assignee="kermit">
<documentation>Leave requested by ${name}</documentation>
<extensionElements>
<activiti:formProperty id="accept" name="Do you Approved ?"
type="boolean"></activiti:formProperty>
</extensionElements>
</userTask>
```

The preceding code snippet is used for creating a user task. This user task will be assigned to a specific user; let us say `kermit`, who is part of the HR team. He will display the name of the person who has applied for leave and has asked that their leave application be either approved or rejected.

8. The following line of code is used for the implementation of the mail task within the business process. The mail task itself is one of the service tasks provided by Activiti. With the help of this task, the mail-sending functionality is implemented. The e-mail ID provided by the user will be taken from the start node.

```
<sequenceFlow id="sid-6993F3A0-8898-4776-A71A-024300A11DBA"
sourceRef="start" targetRef="hrApproval"></sequenceFlow>
<exclusiveGateway id="sid-790E9ECF-B12B-4BFC-B1BA-9332F48DC346"></
exclusiveGateway>
<sequenceFlow id="sid-66FCA025-2413-469A-8FD3-61AECE3C13BA"
sourceRef="hrApproval" targetRef="sid-790E9ECF-B12B-4BFC-B1BA-
9332F48DC346"></sequenceFlow>

<serviceTask id="email" name="Approve mail" activiti:type="mail">
<extensionElements>
<activiti:field name="to">
<activiti:expression>${emailid}</activiti:expression>
</activiti:field>
<activiti:field name="subject">
<activiti:string>Approve leave</activiti:string>
</activiti:field>
<activiti:field name="text">
<activiti:string>Your leave has been approved.</activiti:string>
</activiti:field>
</extensionElements>
</serviceTask>
```

9. The following sequence flows will be executed based on the input provided by the `kermit` user in the user task; if the `accept` property is set to `true`, the mail task flow will be executed and if it is set to `false`, the end node flow will be executed:

```
<endEvent id="sid-AEED2815-DCEC-4EDC-83C7-9ABCF57EA2B8"></
endEvent>

<sequenceFlow id="aprov" sourceRef="sid-790E9ECF-B12B-4BFC-
B1BA-9332F48DC346" targetRef="email"><conditionExpression xs
i:type="tFormalExpression"><![CDATA[${accept=='true'}]]></
conditionExpression>
</sequenceFlow>
```

10. The following code is used for defining your entire process:

```
<sequenceFlow id="sid-3342C68C-A334-4743-B0CF-E94F2AD1C37E"
sourceRef="email" targetRef="sid-AEED2815-DCEC-4EDC-83C7-
9ABCF57EA2B8"></sequenceFlow>

<sequenceFlow id="sid-6E2C6C1D-0493-4A09-8521-A6616823BCF6"
sourceRef="sid-790E9ECF-B12B-4BFC-B1BA-9332F48DC346"
targetRef="sid-AEED2815-DCEC-4EDC-83C7-9ABCF57EA2B8"></
sequenceFlow>

</process>
```

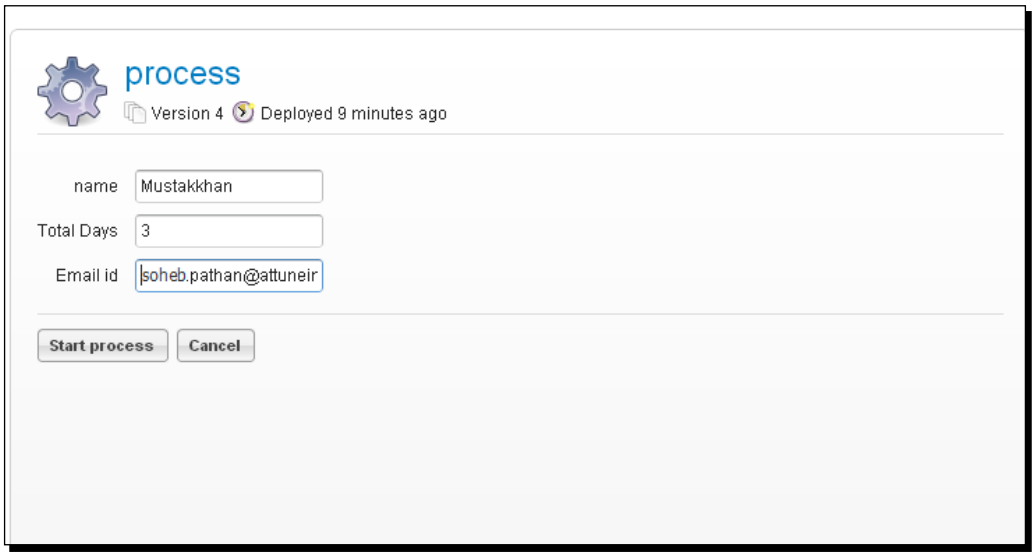
11. The following code is used to represent your entire process in a graphical format; the `bpmndi:BPMNDiagram` tag is used to convert your process to the graphical format:

```
<bpmndi:BPMNDiagram id="BPMNDiagram_process">
<bpmndi:BPMNPlanebpmnElement="process" id="BPMNPlane_process">
<bpmndi:BPMNShapebpmnElement="start" id="BPMNShape_start">
<omgdc:Bounds height="30.0" width="30.0" x="60.0" y="126.0"></
omgdc:Bounds>
</bpmndi:BPMNShape>
<bpmndi:BPMNShapebpmnElement="hrApproval" id="BPMNShape_
hrApproval">
<omgdc:Bounds height="80.0" width="100.0" x="135.0" y="101.0"></
omgdc:Bounds>
</bpmndi:BPMNShape>
<bpmndi:BPMNShapebpmnElement="sid-790E9ECF-B12B-4BFC-B1BA-
9332F48DC346" id="BPMNShape_sid-790E9ECF-B12B-4BFC-B1BA-
9332F48DC346">
<omgdc:Bounds height="40.0" width="40.0" x="270.0" y="120.0"></
omgdc:Bounds>
</bpmndi:BPMNShape>
<bpmndi:BPMNShapebpmnElement="email" id="BPMNShape_email">
<omgdc:Bounds height="80.0" width="100.0" x="375.0" y="45.0"></
omgdc:Bounds>
</bpmndi:BPMNShape>
<bpmndi:BPMNShapebpmnElement="sid-AEED2815-DCEC-4EDC-83C7-
9ABCF57EA2B8" id="BPMNShape_sid-AEED2815-DCEC-4EDC-83C7-
9ABCF57EA2B8">
<omgdc:Bounds height="28.0" width="28.0" x="600.0" y="126.0"></
omgdc:Bounds>
</bpmndi:BPMNShape>
<bpmndi:BPMNEdgebpmnElement="aprov" id="BPMNEdge_aprov">
<omgdi:waypoint x="287.1014492753623" y="122.89855072463769"></
omgdi:waypoint>
<omgdi:waypoint x="280.0" y="81.0"></omgdi:waypoint>
<omgdi:waypoint x="375.0" y="83.62068965517241"></omgdi:waypoint>
</bpmndi:BPMNEdge>
<bpmndi:BPMNEdgebpmnElement="sid-6993F3A0-8898-4776-A71A-
024300A11DBA" id="BPMNEdge_sid-6993F3A0-8898-4776-A71A-
024300A11DBA">
<omgdi:waypoint x="90.0" y="141.0"></omgdi:waypoint>
<omgdi:waypoint x="135.0" y="141.0"></omgdi:waypoint>
</bpmndi:BPMNEdge>
<bpmndi:BPMNEdgebpmnElement="sid-6E2C6C1D-0493-4A09-8521-
A6616823BCF6" id="BPMNEdge_sid-6E2C6C1D-0493-4A09-8521-
A6616823BCF6">
<omgdi:waypoint x="310.0" y="140.0"></omgdi:waypoint>
```

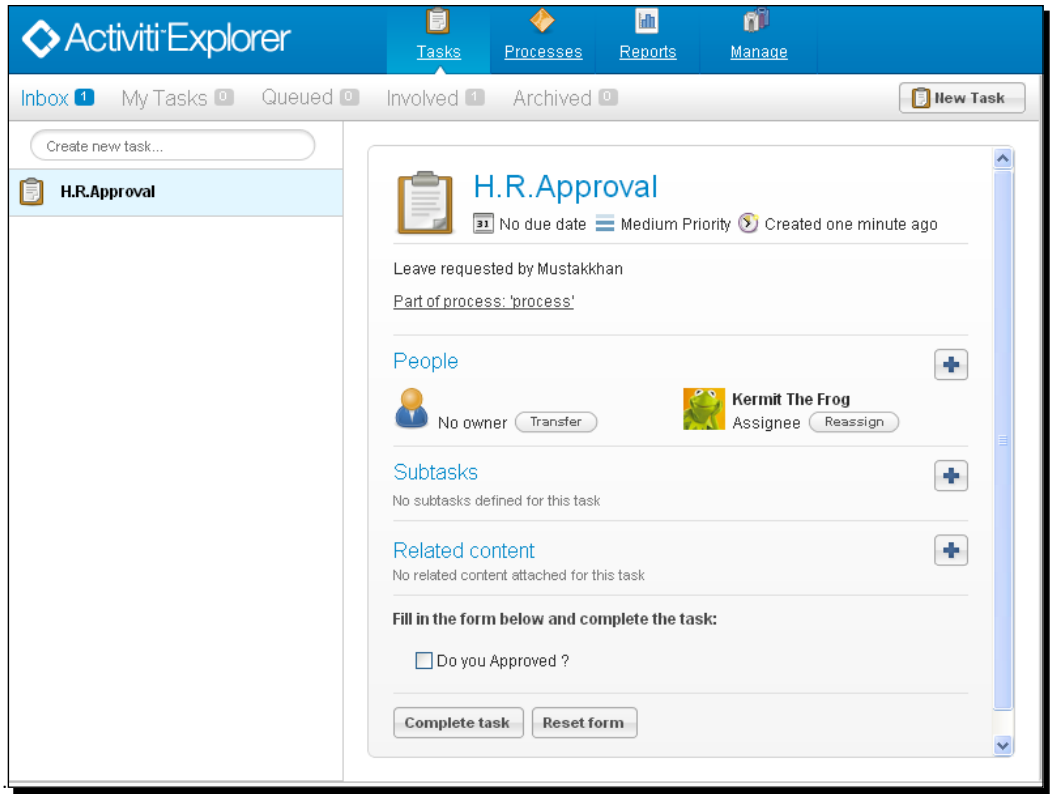
```
<omgdi:waypoint x="440.0" y="140.0"></omgdi:waypoint>
<omgdi:waypoint x="454.0" y="242.0"></omgdi:waypoint>
<omgdi:waypoint x="602.1948192271975" y="147.52580274266163"></
omgdi:waypoint>
</bpmndi:BPMNEdge>
<bpmndi:BPMNEdgebpmnElement="sid-3342C68C-A334-4743-B0CF-
E94F2AD1C37E" id="BPMNEdge_sid-3342C68C-A334-4743-B0CF-
E94F2AD1C37E">
<omgdi:waypoint x="475.0" y="85.0"></omgdi:waypoint>
<omgdi:waypoint x="545.0" y="85.0"></omgdi:waypoint>
<omgdi:waypoint x="545.0" y="140.0"></omgdi:waypoint>
<omgdi:waypoint x="600.0" y="140.0"></omgdi:waypoint>
</bpmndi:BPMNEdge>
<bpmndi:BPMNEdgebpmnElement="sid-66FCA025-2413-469A-8FD3-
61AECE3C13BA" id="BPMNEdge_sid-66FCA025-2413-469A-8FD3-
61AECE3C13BA">
<omgdi:waypoint x="235.0" y="140.52380952380952"></omgdi:waypoint>
<omgdi:waypoint x="270.188679245283" y="140.18867924528303"></
omgdi:waypoint>
</bpmndi:BPMNEdge>
</bpmndi:BPMNPlane>
</bpmndi:BPMNDiagram>
</definitions>
```

12. Once your process is deployed successfully, you can test it by logging in to the Activiti Engine as a gonzo user and starting the leave process.

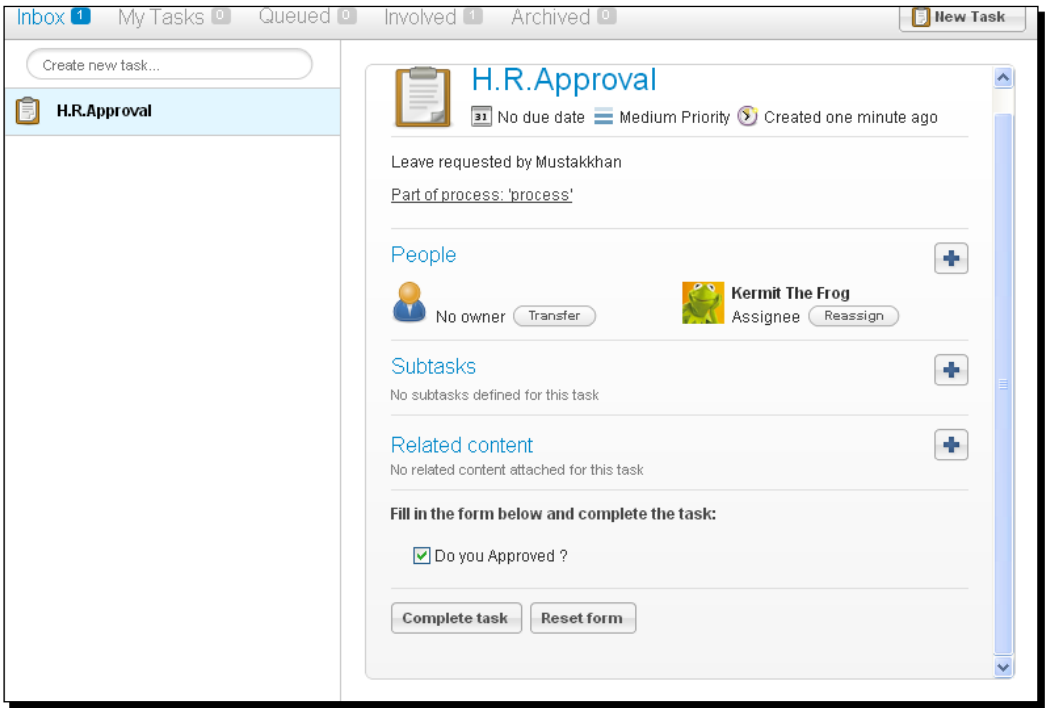
13. At the start of the process, a form will pop up that asks for the leave information of the respective employee, as shown in the following screenshot:



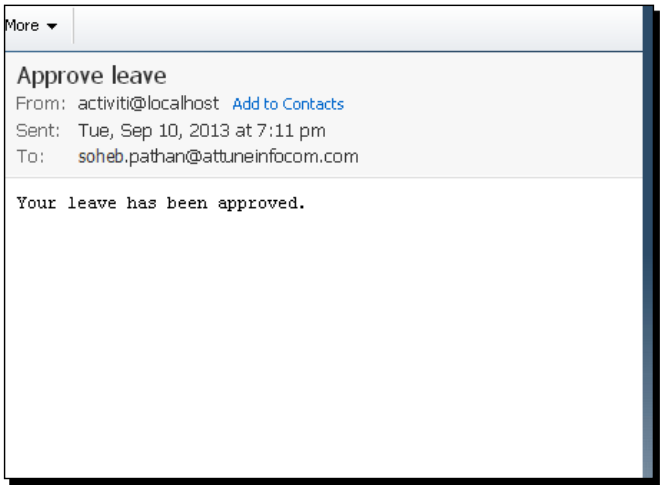
14. As there is a user task after the start node, log in as a `kermit` user. In **Inbox** folder of the `kermit` user, there will be an **H.R.Approval** task, as shown in the following screenshot:



15. Now the leave will be approve. So, a leave approval e-mail will be sent to user.



16. To check whether or not the mail has been received, the user needs to log in to the mail account entered at the start of the process. They would have received a mail for leave approval, as shown in the following screenshot:



What just happened?

We have seen how to use the e-mail task in Activiti. Now we know which configuration is required in `activiti-standalone-context.xml` to send mails using Activiti. We configured the e-mail task in Activiti to send an e-mail from the leave approval process workflow.

Time for action – running the Activiti Engine

So far, we have created a process using the Designer and Modeler and deployed it using the Activiti Explorer. We have seen that configuration can be done using `activiti-standalone-context.xml`. Similar configurations can also be done using the `ProcessEngine` class. In this section, we will take a look at this and in *Chapter 6*, *The Activiti ProcessEngine API,* we will see more details regarding various APIs.

We can create the Activiti Process Engine using the `org.activiti.engine.ProcessEngines` class. To configure the Process Engine, we have to create the `activiti.cfg.xml` file.

Let's see how we can run the Activiti Engine.

Firstly, we can create an instance of `ProcessEngine` as follows:

```
ProcessEngineprocessEngine = ProcessEngines.getDefaultProcessEngine()
```

This instance will search for the `activiti.cfg.xml` file and create a Process Engine based on the configuration defined in that file.

We can configure the file using the following configuration properties:

```
<beans xmlns="http://www.springframework.org/schema/beans"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.springframework.org/schema/beans
http://www.springframework.org/schema/beans/spring-beans.xsd">

<bean id="processEngineConfiguration" class="org.activiti.engine.impl.
cfg.StandaloneProcessEngineConfiguration">

<property name="jdbcUrl" value="jdbc:h2:mem:activiti;DB_CLOSE_
DELAY=1000" />
<property name="jdbcDriver" value="org.h2.Driver" />
<property name="jdbcUsername" value="sa" />
<property name="jdbcPassword" value="" />

<property name="databaseSchemaUpdate" value="true" />
```

```
<property name="jobExecutorActivate" value="false" />

<property name="mailServerHost" value="mail.demo.com" />
<property name="mailServerPort" value="25" />
</bean>

</beans>
```

This configuration is done for the Spring environment.

It is not mandatory to create the configuration of `ProcessEngine` in the Spring environment. We can also create an object of `ProcessEngineConfiguration` programmatically in the configuration file. We can also use a different bean ID.

Using the following code, we can create `ProcessEngine`:

```
ProcessEngineConfiguration.
createProcessEngineConfigurationFromResourceDefault();
ProcessEngineConfiguration.createProcessEngineConfigurationFromResourc
e(String resource);
ProcessEngineConfiguration.createProcessEngineConfigurationFromResourc
e(String resource, String beanName);
ProcessEngineConfiguration.createProcessEngineConfigurationFromInputSt
ream(InputStream inputStream);
ProcessEngineConfiguration.createProcessEngineConfigurationFromInputSt
ream(InputStream inputStream, String beanName);
```

All the `ProcessEngineConfiguration.createXXX()` methods return a `ProcessEngineConfiguration` instance that can be used further.

`ProcessEngine` is created after calling the `buildProcessEngine()` method as follows:

```
ProcessEngineprocessEngine = ProcessEngineConfiguration.
createStandaloneInMemProcessEngineConfiguration()
  .setDatabaseSchemaUpdate(ProcessEngineConfiguration.DB_SCHEMA_
UPDATE_FALSE)
  .setJdbcUrl("jdbc:h2:mem:my-own-db;DB_CLOSE_DELAY=1000")
  .setJobExecutorActivate(true)
  .buildProcessEngine();
```

What just happened?

We learned about the class that is used to create the Process Engine. Now we know which properties are required in the `activiti.cfg.xml` file for the Process Engine. We learned to create the Activiti Process Engine without using the Spring environment.

Have a go hero

Now that you've gone through the chapter, feel free to attempt the following activities:

- Configuring Activiti with the Oracle database
- Adding the e-mail task for order confirmation in the laptop order process that we saw in *Chapter 2*, *Modeling Using the Activiti Modeler*

Pop quiz – the Activiti Process Engine

Q1. Which bean ID is used for configuring the mail server's properties?

1. `transactionManager`
2. `datasource`
3. `ProcessEngineConfiguration`
4. `processEngine`

Q2. Which file is used to configure the production database in the Spring environment?

1. `db.properties`
2. `activiti-standalone-context.xml`
3. `web.xml`
4. `rebel.xm`

Q3. Which layer is optional in Activiti?

1. The Activiti Spring layer
2. The Activiti Engine layer
3. The PVM layer
4. The Activiti components

Summary

We have learned how to set up a production database in the Spring environment. Now we can send an actual e-mail using the e-mail task in Activiti. We have also learned about the Activiti Engine and logging framework. In the next chapter, we will see an overview of APIs and different services. We will also learn about the different types of APIs for Activiti.

6
The Activiti ProcessEngine API

In the previous chapter, we came across various functionalities and configured a mail engine in Activiti. In this chapter, we will be interacting with the process engine using various APIs of Activiti, such as RuntimeService, FormService, ManagementService, and HistoryService.

In this chapter, we will cover the following topics:

- An overview of APIs and services
- API for the start process instance and deploying the process
- API for working with user tasks
- API for querying historic activities

We can interact with the Activiti process engine using various Activiti process APIs and services. To look into the various APIs, let's see an overview of the APIs and their services.

Overview of the APIs and services

Activiti Engine provides different types of functionalities. To implement these functionalities, Activiti provides various services within the **APIs** (**Application Programming Interfaces**) to interact with Activiti. To achieve the functionalities of Activiti, we need to use the ProcessEngine API, which is an entry point of the process engine. We can use the various services of Activiti using the ProcessEngine API. The ProcessEngine API and the services objects are thread-safe.

Using the ProcessEngine API, we can use various services, such as RuntimeService, RepositoryService, TaskService, ManagementService, IdentityService, HistoryService, and FormService. Each service API offers a unique service. Let's see how these will help us:

- **RepositoryService**: This is mainly used to deploy process definitions. It provides operations for the management and deployment of process definitions. We can delete, query, deploy, and retrieve process definitions.
- **IdentityService**: This provides an interface for the management of groups and users. It is also useful for authentication in Activiti.
- **FormService**: This provides service in the form of startform and taskform. We can use this service for getting inputs from the user.
- **TaskService**: This provides functionality related to various user tasks. We can start a task, assign a task, or create a new standalone task.
- **ManagementService**: This is a service for administrative and maintenance operations on the process engine. These operations will typically not be used in a workflow-driven application, but are used in, for example, the operational console.
- **RuntimeService**: This is used to retrieve and store process variables. It is also used for querying on process instances and executions.
- **HistoryService**: We can use this service to get historical data.

Let's see each service in detail.

Delving into the process engine

The process engine is the heart of Activiti and is used to take advantage of the various services in Activiti. Using the ProcessEngine API, we can interact with the Activiti Engine and perform various functionalities. The ProcessEngine objects are thread-safe, so we can keep a reference for the whole server. It provides access to all the services that expose the workflow operations. An end-user application will require only one central `Process Engine` instance. A process engine is built through a `ProcessEngineConfiguration` instance and is a costly operation that should be avoided. For that purpose, it is advised that you store it in a static field or at a `JNDI` location (or something similar). This is a thread-safe object, so no special precautions need to be taken. The `ProcessEngine` class will scan for two files: one is `activiti.cfg.xml` and the other is `activiti-context.xml`.

The process engine uses Spring, so first a Spring application context is created and then the process engine is obtained from that application context. In Activiti, all services are stateless, so we can easily run Activiti on multiple nodes in a cluster environment, with each pointing to the same database. For configuring a process engine, you have to use the following API:

```
ProcessEngine processEngine = ProcessEngines.getDefaultProcessEngine()
```

RuntimeService

RuntimeService deals with starting new process instances of process definitions. This service is used to retrieve and store process variables. The data available in the process variables is specific to the given process instance. In a given process instance, various types of gateways can be used. These process variables determine the path to continue the process. We can also query on process instances and executions using RuntimeService. Executions are a representation of the *token* concept introduced by BPMN 2.0. RuntimeService points to the process that is being executed. Lastly, RuntimeService is used whenever a process instance is waiting for an external trigger and the process needs to be continued. A process instance can have various wait states, and RuntimeService contains various operations to signal to the instance that the external trigger is received and the process instance can be continued. The API used for RuntimeService is as follows:

```
RuntimeService runtimeService = processEngine.getRuntimeService();
```

RepositoryService

RepositoryService is considered the main service for interacting with the Activiti Engine. With the help of this service, we can manage and manipulate the business process. Using this service, we can perform the following actions:

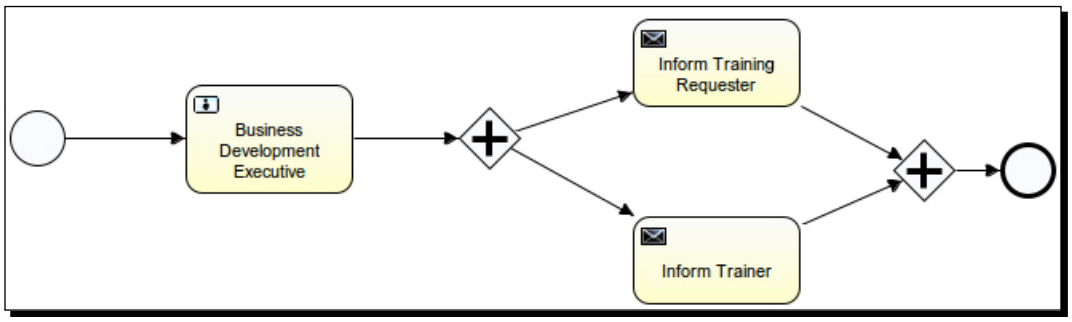
- Deployment of a process into the repository
- Querying the deployed process from the process engine
- Suspending and activating the deployed process

To access RepositoryService, use the following API:

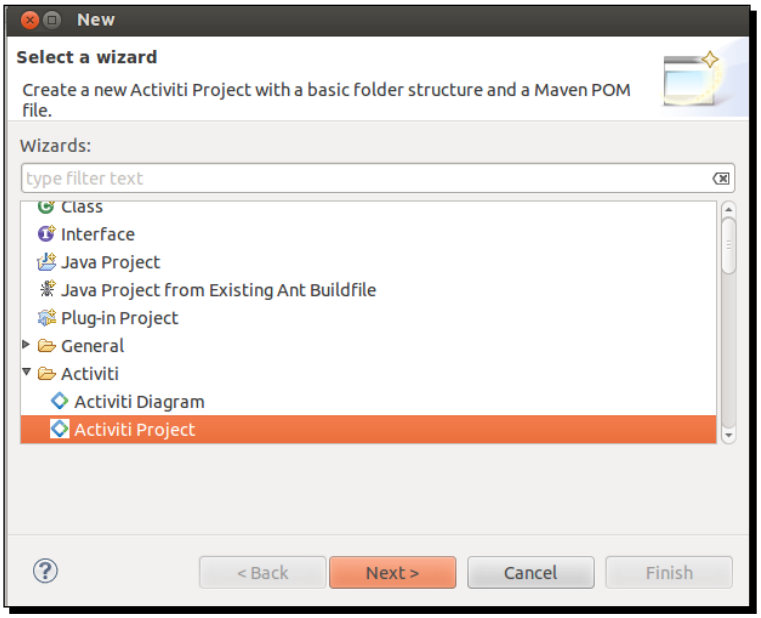
```
RepositoryService repositoryService = processEngine
        .getRepositoryService();
```

Time for action – deploying the process

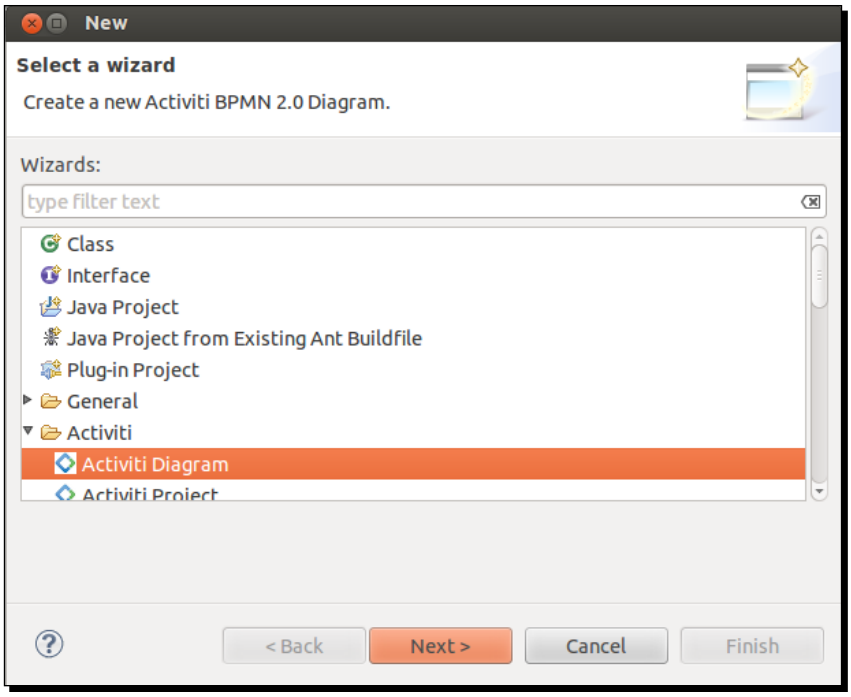
As we have discussed certain repositories, we will now use them to deploy processes. To start with the deployment, we will take an example of the Attune University where a customer registers for a training. So, in this process, the customer will provide their name, e-mail ID, date, and the topic for which they want training. Once they submit their request, it will be sent to the BDE, who will check whether or not the trainer is available on the specified date. Then, he will schedule the training with the trainer and a mail will be sent to both the trainer and trainee regarding the training schedule. The process will look as shown in the following screenshot:



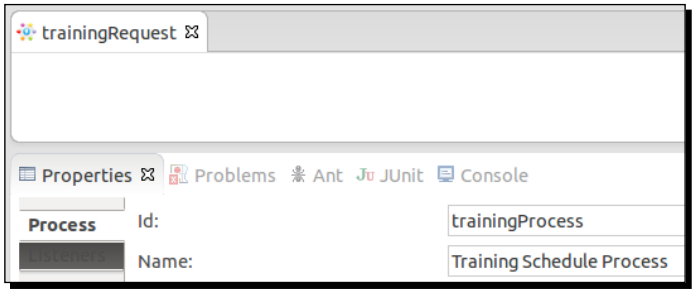
1. To start the implementation, we will create a new Activiti project named `AttuneUniversity-Process` in Eclipse by navigating to **File** | **New** | **Other** | **Activiti Project**:



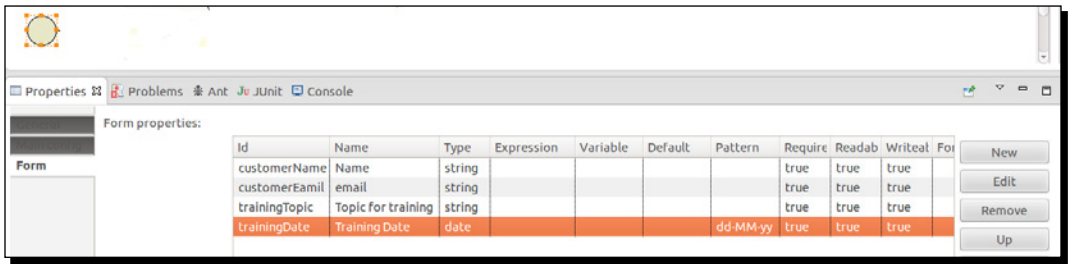
2. On successful creation of the project, we will create an Actviti diagram in it with the name `trainingRequest`. We can do that by navigating to **File** | **New** | **Other** | **Activiti Diagram**.



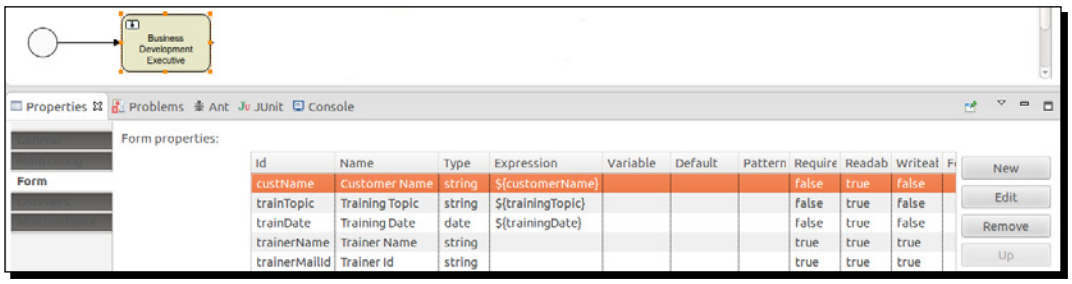
3. First, we will provide an ID and name for the business process. To do that, open the **Properties** window as shown in the following screenshot:



4. Now drag-and-drop the `Start` node into the canvas and populate the **Properties Form** tab to accept values from the user as in the following screenshot:

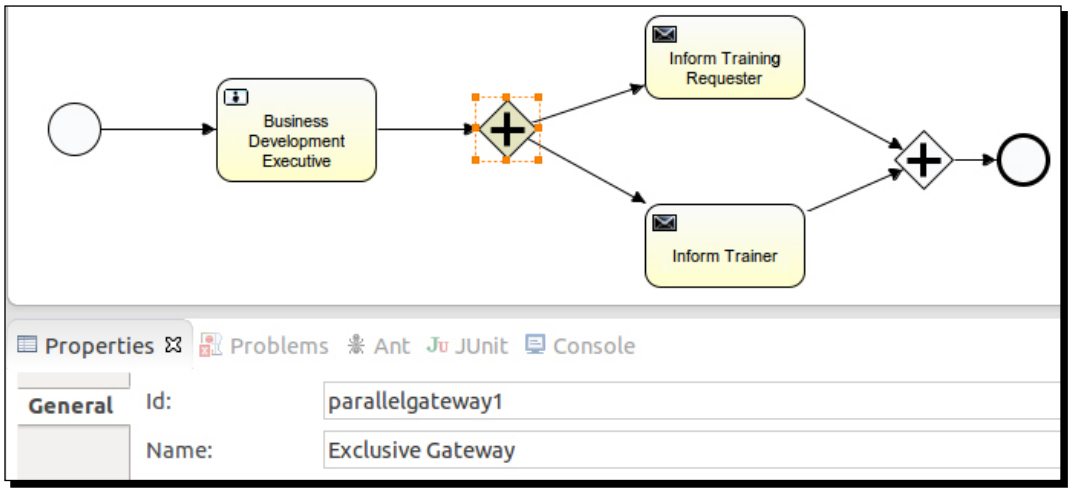


5. Create a user task called **Business Development Executive** where the business developer will be able to view the customer's request for training and accordingly assign a trainer for them:

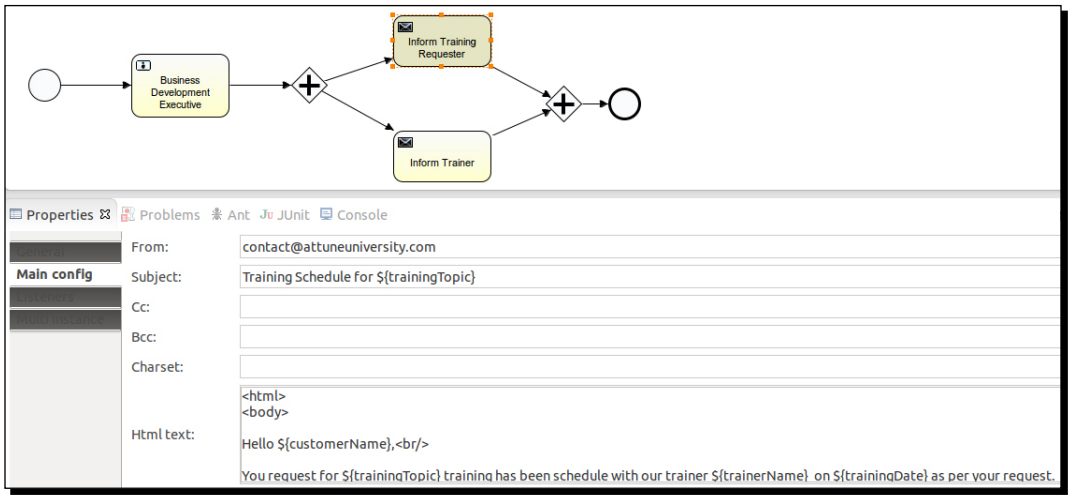


6. We will assign the user task to `gonzo` so that he can perform the task and execute the process:

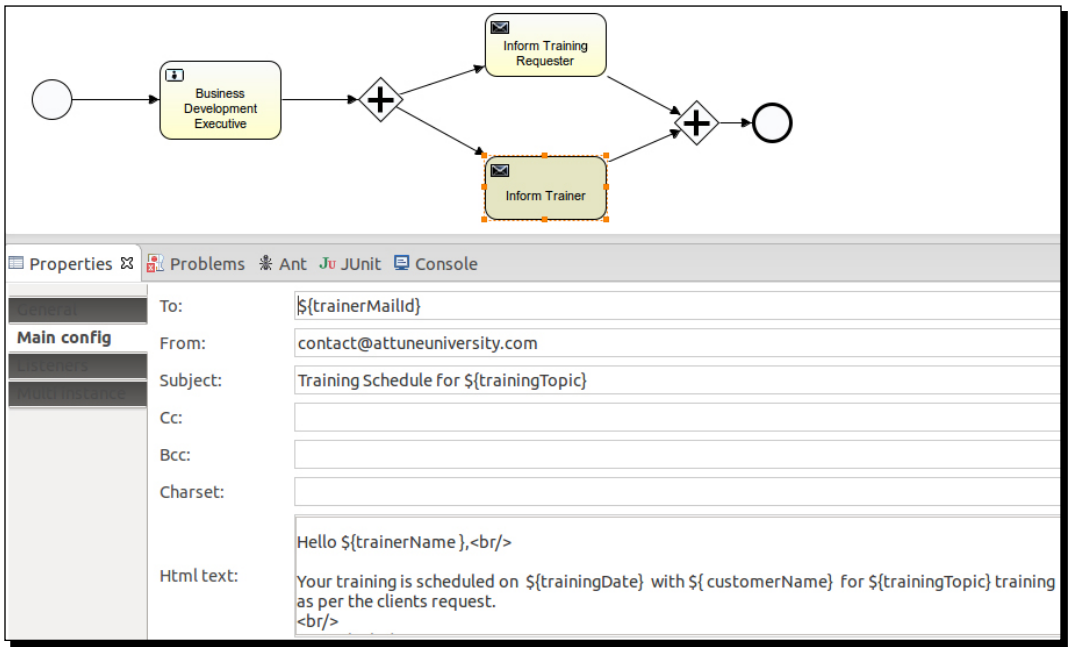
7. Now, as we want an e-mail to be sent to both the trainer and the trainee regarding the schedule, we will insert a parallel gateway and two mail tasks:



8. As we are sending e-mails to both the trainer and the trainee, we have to populate the properties of the `Mail` task, as shown in the following screenshot; the mail will be sent to the trainee and will contain `trainingTopic` with the name of the trainer, `trainerName`, and the date of training, `trainingDate`:



9. A similar e-mail will also be sent to the trainer, containing `customerName`, `trainingDate`, and `trainingTopic`:



10. We have created the business process using the Eclipse designer and now we want to deploy it into the process engine, but this time using the ProcessEngine API.

11. We will create a class `ExecuteProcess` having a main method configuring standalone process engine with database, as in the following code:

```
ProcessEngine processEngine = ProcessEngineConfiguration
.createStandaloneProcessEngineConfiguration()
.setJdbcDriver("com.mysql.jdbc.Driver")
.setJdbcUrl("jdbc:mysql://localhost:3306/activiti_book")
.setJdbcPassword("root").setJdbcUsername("root")
.buildProcessEngine();
```

12. Now, to deploy the process into the repository, we have to invoke RepositoryService; so, implement the following code within the class file for the deploying process:

```
RepositoryService repositoryService = processEngine
  .getRepositoryService();

// Deploying The process into the repository
repositoryService.createDeployment().addInputStream(
  "trainingRequest.bpmn20.xml", ReflectUtil
```

```
    .getResourceAsStream("diagrams/trainingRequest.bpmn"))
      .deploy();
```

13. Now, to deploy the process into the repository, we need to fetch the RepositoryService instance of the process engine that we have created. For the deployment, we need to create a deployment file in which to place our process. The `createDeployment` method will create the deployment file for us. As we have a `.bpmn` file, we need to generate a `.bpmn20.xml` file for the deployment.

What just happened?

So far, we have seen how to create and configure a process engine using API and using the process engine, we deployed the business process into the process engine through API.

Time for action – starting a new process instance

As we have deployed the process into the Activiti Engine, now we have to start the process instance. For that, we have to perform the following steps:

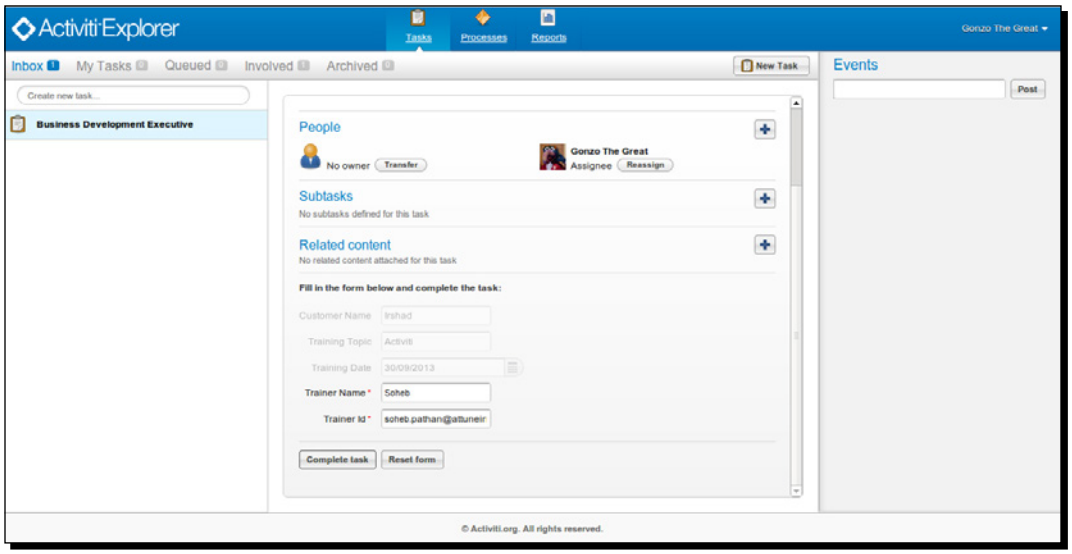
1. To start the process using API, we have to implement the given code in the class file:

```
// Starting the Deployed Process

Date date = new Date();
Map<String, Object> variables = new HashMap<String, Object>();

variables.put("customerName", "Irshad");
variables.put("customerEamil", "irshad.mansuri@attuneinfocom.
com");
variables.put("trainingTopic", "Activiti");
variables.put("trainingDate", date);

RuntimeService runtimeService = processEngine.getRuntimeService();
ProcessInstance processInstance = runtimeService
    .startProcessInstanceByKey("trainingProcess", variables);
```

2. Here, we are creating a `Map` variable to pass values from the class files to the business process. We have declared the form variables in our process, so we are populating them to execute the business process.

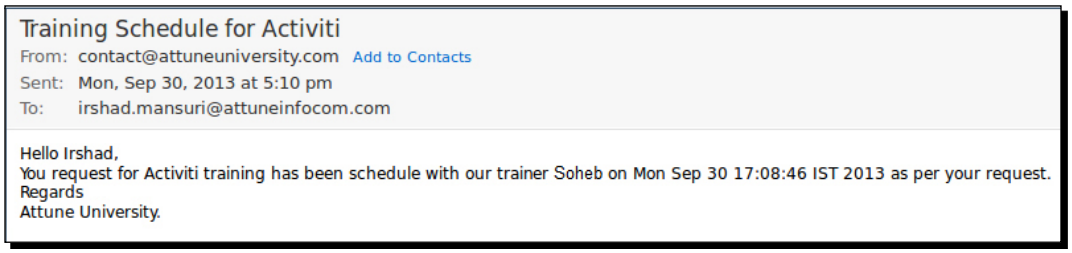
3. To start the business process, we are creating a runtime instance for our process. We need to provide the `processId` and pass the variables for execution.

4. After executing the preceding code, you can log in with a `gonzo` user who is assigned a BDE task. On logging in with the `gonzo` user, you will find a form with the details of the trainee. Provide the name and e-mail ID of the trainer to whom you want to assign the training, as shown in the following screenshot:

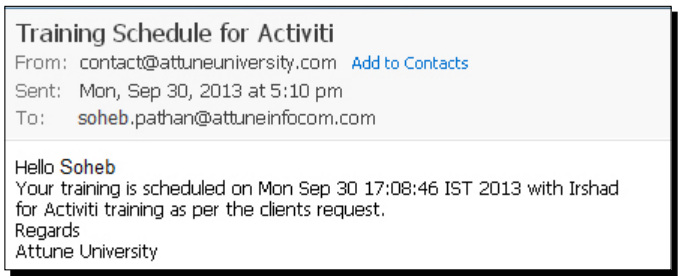


5. On completion of the preceding task, an e-mail will be delivered to both the trainee as well as the trainer with the training details.

6. The following screenshot shows the e-mail received by the trainer:



7. The following screenshot shows the e-mail received by the trainee:



What just happened?

We learned how we can use RuntimeService with the process instance. We were using start instances using Activiti Explorer, but now we know how to start process instances through API.

TaskService

TaskService is used to access all kinds of operations related to `Task` and `Form`. Using TaskService, we can claim a task, create attachments, assign a user to a task, create a standalone task that is not related to process instances, create task-related queries, and change the ownership of a task to another user. It is possible to delete a task through the TaskService API.

Time for action – querying for user tasks

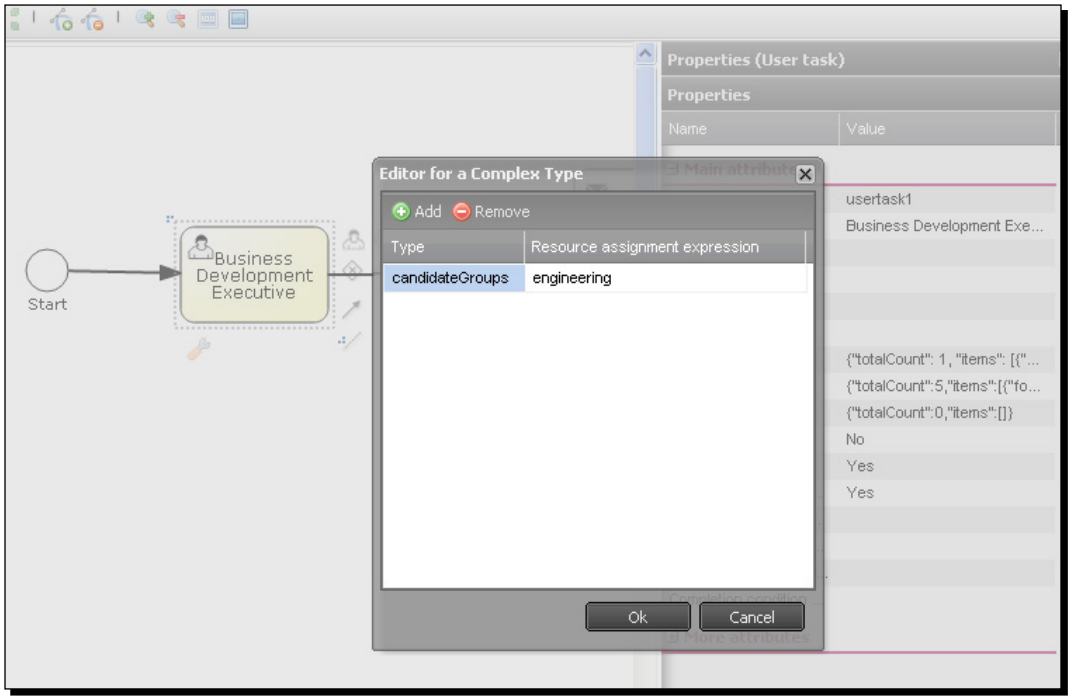
Now, let's see how we can query user tasks. Let's say we want to find the tasks available for users of the engineering group; so, we will perform the following steps:

1. To fetch the list of tasks, we have to create a class file, `ListTaskDemo`, with the following code:

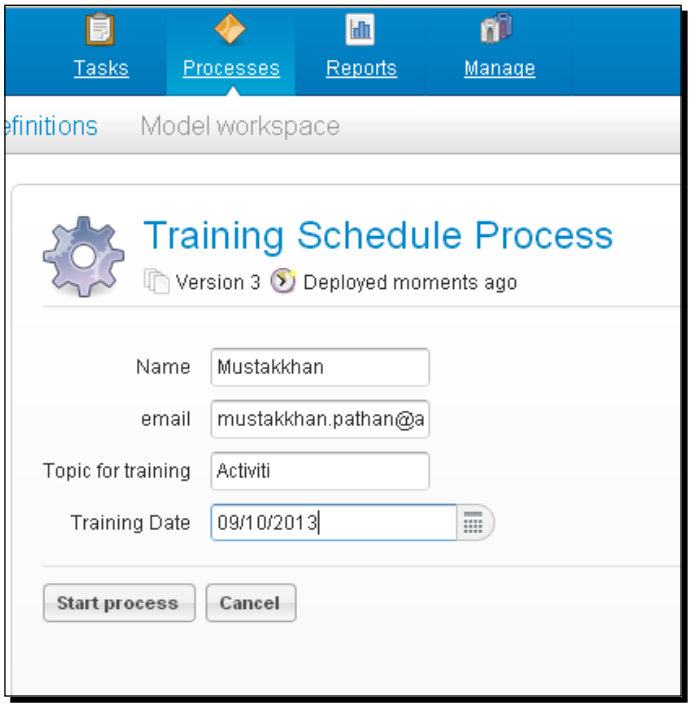
```
public class ListTaskDemo {
public static void main(String ar[])
  {
  ProcessEngine processEngine = ProcessEngineConfiguration
      .createStandaloneProcessEngineConfiguration()
      .setJdbcDriver("com.mysql.jdbc.Driver")
      .setJdbcUrl("jdbc:mysql://localhost:3306/activiti_book")
      .setJdbcPassword("root").setJdbcUsername("root")
      .buildProcessEngine();
```

```
    TaskService taskService = processEngine.getTaskService();

List<Task> tasks = taskService.createTaskQuery().taskCandidateGrou
p("engineering").list();

    for(Task task : tasks)
{
    System.out.println("Task available for Engineering group: " +
    task.getName());
}
    }
}
```

2. Deploy the previous training process into the Activiti Explorer.
3. Edit the **Business Development Executive** task. Assign it to the **engineering** group as shown in the following screenshot:



4. After making the changes, we have to redeploy the process to put them in effect.
5. Once the changes are deployed, we have to restart the process instance. We can start the process instance either from the Activiti Explorer or re-execute the class in which the API for starting the process is implemented:



6. Execute the `ListTaskDemo` class.
7. We will get the following output:

```
Task available for Engineering group: Business Development
Executive
```

What just happened?

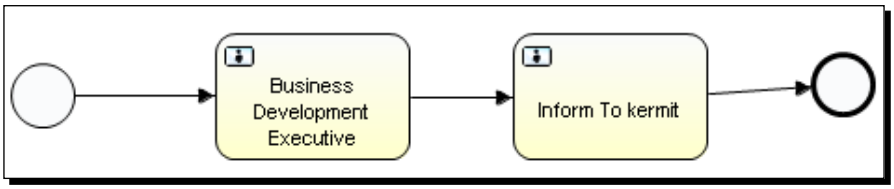
We saw how to use TaskService for different purposes. We have learned how to query a user task using API. We also learned to list out the available tasks for a particular group.

Time for action – creating and completing user tasks

We can create and complete user tasks using the various service APIs of Activiti. In the previous example, we saw how to find a task allocated to a particular group or user. Now, let's see how to complete a user task using the API. We will create a new process for this task.

In this example, we will create a process in which at the start event, we will have one form from which we get information about training requirements and then we connect the flow of the process with the BDE task as in the previous example. Then, we will find the task allocated to gonzo and complete the gonzo task using the API. Then, the trainer information will be available to the Kermit user. We will have to perform the following steps to complete the user task using the API:

- We will design a training request process as we discussed previously.
- The process should look like the one shown in the following screenshot:



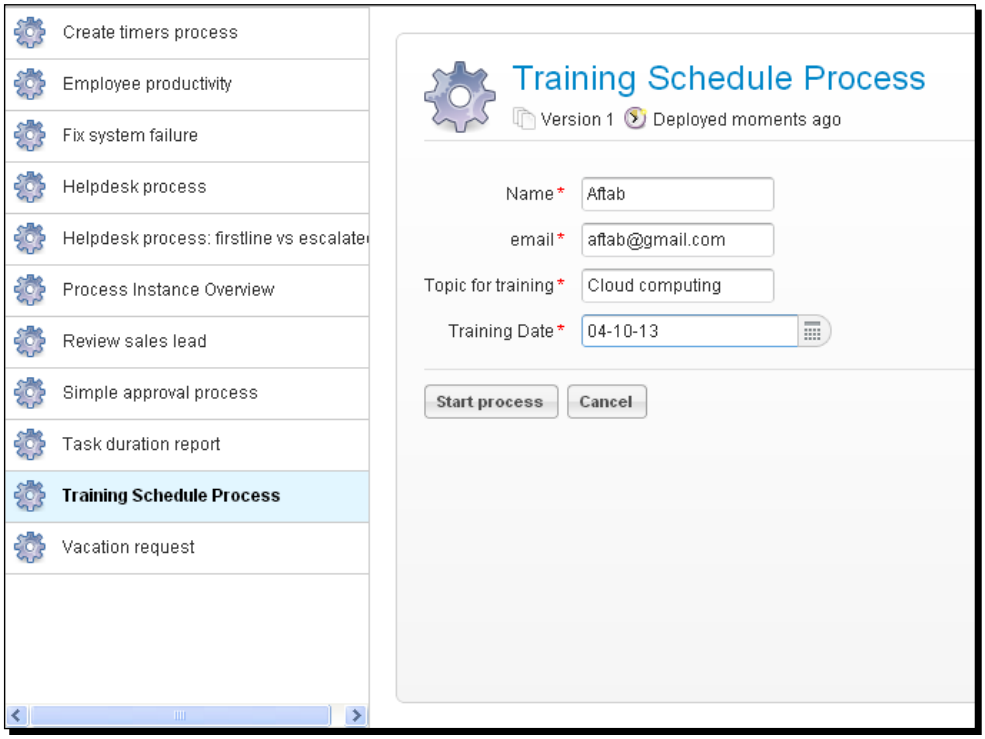
1. Now deploy this process into the Activiti Explorer.
2. Create a Java class and write the following code:

```
public static void main(String ar[])
  {
  ProcessEngine processEngine = ProcessEngineConfiguration
      .createStandaloneProcessEngineConfiguration()
      .setJdbcDriver("com.mysql.jdbc.Driver")
      .setJdbcUrl("jdbc:mysql://localhost:3306/activiti_book")
      .setJdbcPassword("root").setJdbcUsername("root")
      .buildProcessEngine();

  TaskService taskService = processEngine.getTaskService();

  List<Task> tasks =
  taskService.createTaskQuery().taskAssignee("gonzo").list();

  for(Task task : tasks)
{
  System.out.println("Task available for Gonzo: " + task.
getName());
```

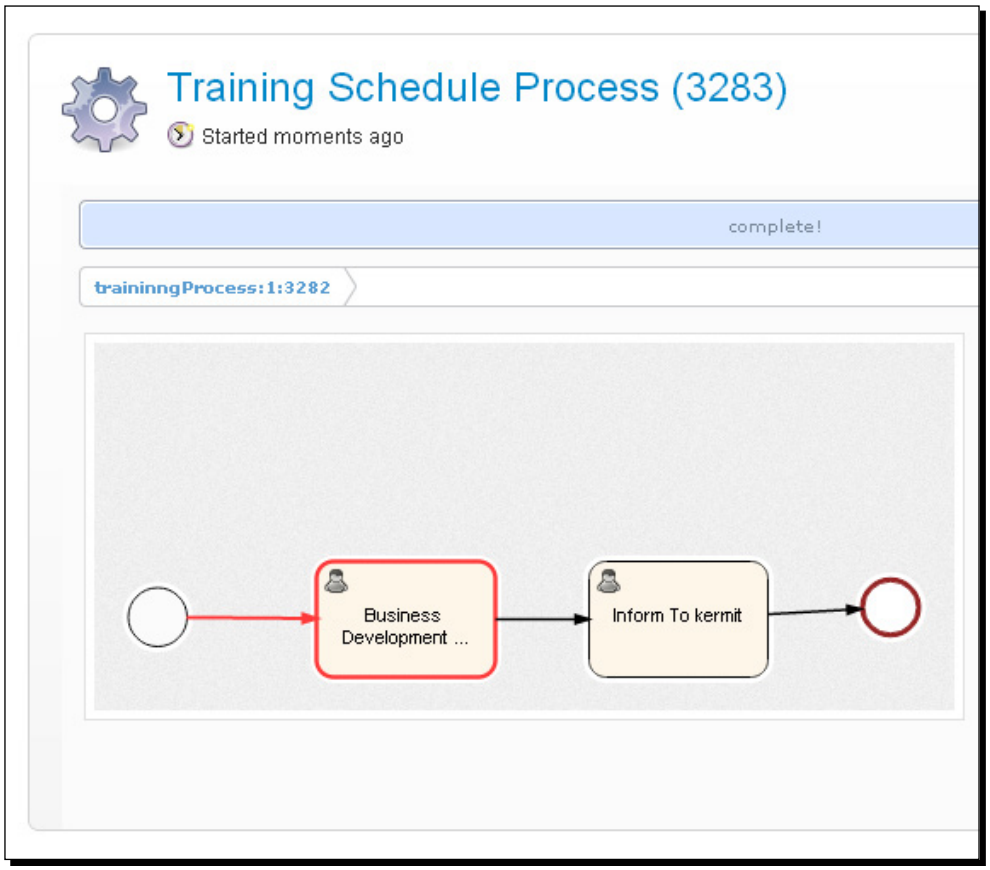
```
}

  Task task = tasks.get(0);
  Map<String, Object> taskVariables = new HashMap<String,
Object>();
  taskVariables.put("trainerName", "MustakKhan");
  taskVariables.put("trainerMailId", "Mustakkhan@attuneinfocom.
com");
  taskService.complete(task.getId(), taskVariables);
  }
```

As we can see in the preceding code, first it is getting the process engine and then using `TaskService` to fetch the currently executing task list. So we can get the available tasks for the `gonzo` user and then, using the `taskservice.complete` method, complete the task for the `gonzo` user in five lines

3. Now, let's start the **Training Schedule Process** from the Activiti Explorer as shown in the following screenshot:

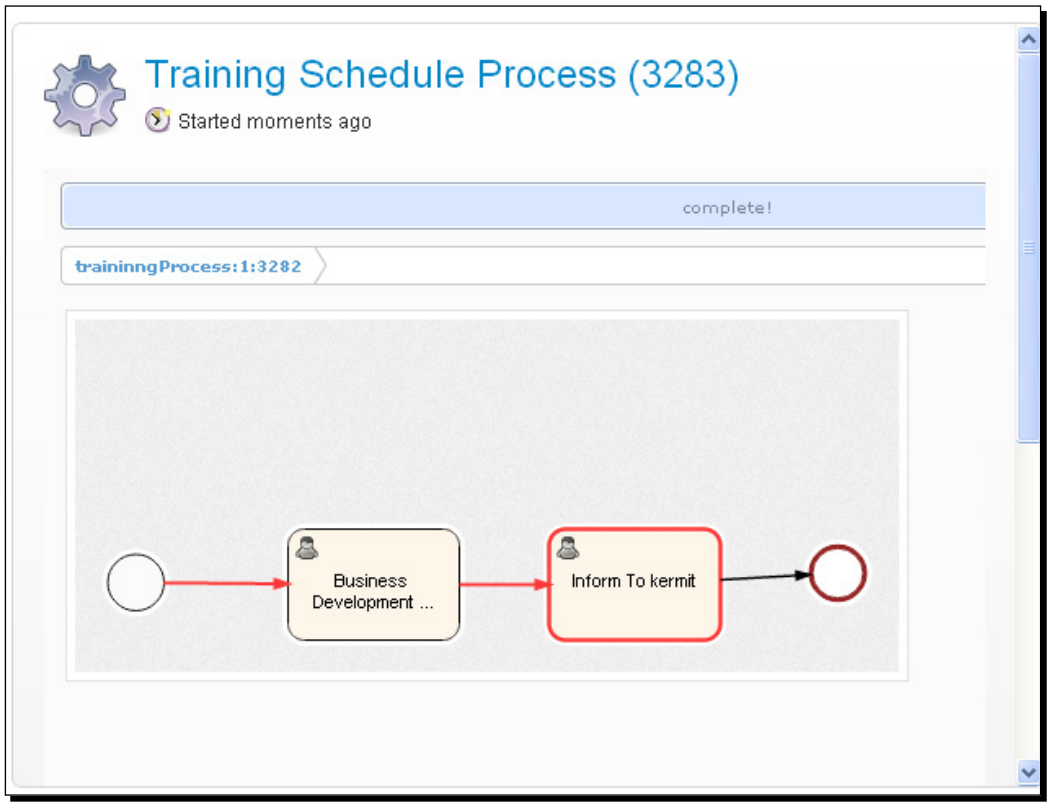


4. As shown in the following screenshot, a task that is assigned to the gonzo and also available to the BDE:



5. We want to complete this gonzo BDE task using an API; so, let's execute the Java class.

6. Now, the gonzo task has been completed. Refresh the page. You can see in the following screenshot that a task is available for Kermit:



7. Now, we have to complete the Kermit task for the successful completion of the process.

What just happened?

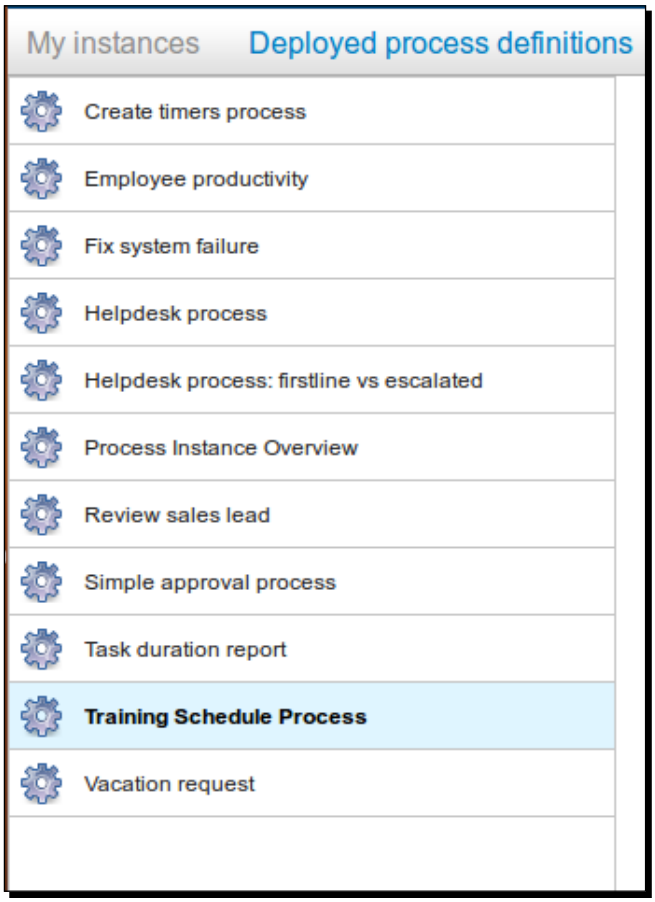
We saw how to use the user TaskService and TaskService using the ProcessEngine API. We learned how to complete a task using TaskService.

Time for action – suspending a process

So far, we have seen how to configure a process engine, deploy a process into the process engine, start a process, and query for a user task using an API.

Now, if there is a requirement to suspend a process, you can implement it using an API. We have already created a training process and now we want to suspend the process for some time. You can follow the ensuing steps to do so:

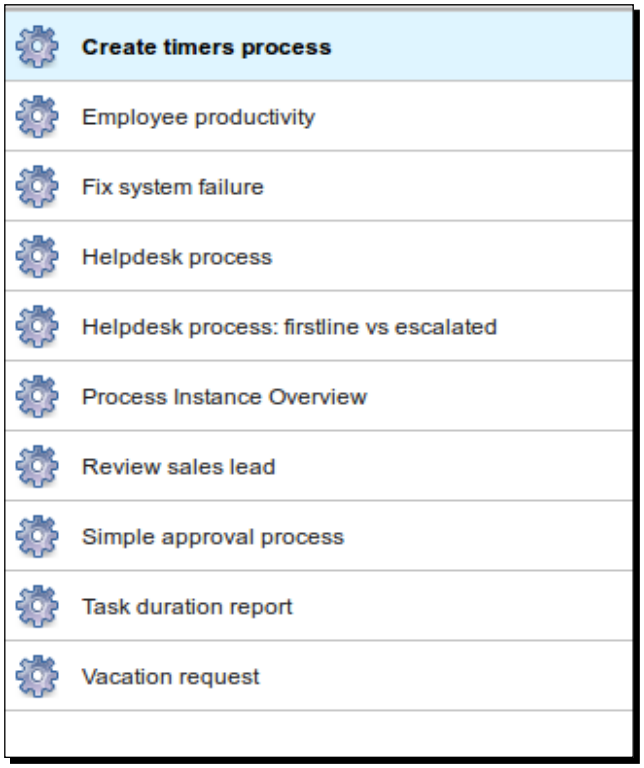
1. Before suspending a process, just make sure that **Training Schedule Process** is available in the **Deployed process definitions** tab:



2. Now create a `SuspensionActivation` class file to suspend the process with the following code in the class file:

```
ProcessEngine processEngine = ProcessEngineConfiguration

  .createStandaloneProcessEngineConfiguration()
  .setJdbcDriver("com.mysql.jdbc.Driver")
  .setJdbcUrl("jdbc:mysql://localhost:3306/activiti_book")
  .setJdbcPassword("root").setJdbcUsername("root")
  .buildProcessEngine();

RepositoryService repositoryService = processEngine
        .getRepositoryService();

repositoryService.suspendProcessDefinitionByKey("traininngProce
ss");
```

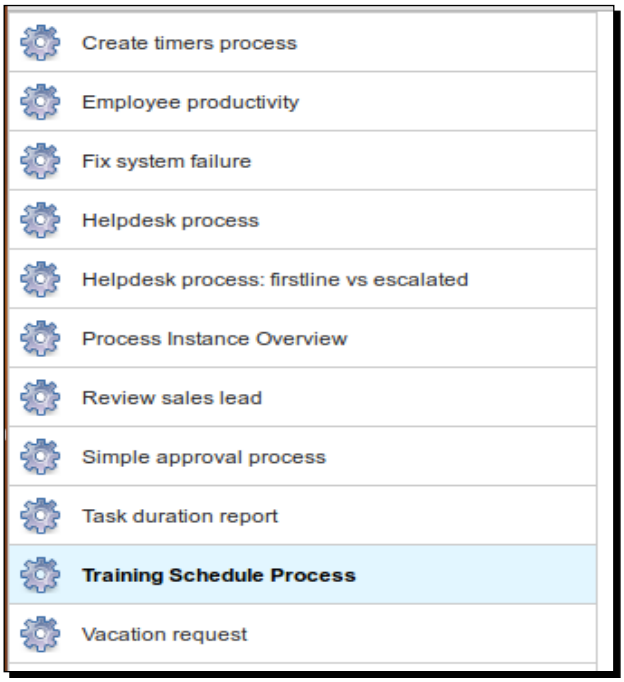
3. Once we finish with the coding part, just execute the class file and then browse to the Activiti Explorer and check for **Training Schedule Process** in the **Deployed process definitions** tab:



4. If you don't find your process within the list, it means you have suspended the process successfully.
5. So, once you suspend the process, it cannot be restarted for execution.
6. To reactivate the process, you can add the following code within your `SuspensionActivation` class file:

```
ProcessEngine processEngine = ProcessEngineConfiguration

  .createStandaloneProcessEngineConfiguration()
  .setJdbcDriver("com.mysql.jdbc.Driver")
  .setJdbcUrl("jdbc:mysql://localhost:3306/activiti_book")
  .setJdbcPassword("root").setJdbcUsername("root")
  .buildProcessEngine();

RepositoryService repositoryService = processEngine
  .getRepositoryService();

repositoryService.activateProcessDefinitionByKey("traininngProce
ss");
```

7. On execution of the following code, you will again be able to see **Training Schedule Process** in the process list, as shown in the following screenshot:



8. The preceding code will activate the process, and you can continue further executing the business process.

What just happened?

Recently, we learned the use of the `suspendProcessDefinition` and `activateProcessDefinition` methods of `repositoryService`. So, with their help, we can suspend and reactivate our processes.

ManagementService

ManagementService is used to maintain operations performed on the process engine. With the help of this service, we can upgrade the schema, fetch the executed commands and properties of the process engine, and execute the table page query for fetching table row data. ManagementService cannot be used in the workflow application. This service is just for managing your business process and not for executing any operations on the business process.

IdentityService

IdentityService is used for managing users and groups. We can create, update, delete, and update group and user information. Activiti doesn't perform checks on users at runtime, because in production you may have used a user management protocol such as LDAP. Using IdentityService, we can programmatically query for users and groups. We can delete a particular user or group. We can also create a new user and group.

Time for action – working with users

IdentityService is used to manage users and groups in Activiti. Now, let's see a demo for working with users. To implement IdentityService, we have to follow the ensuing steps:

1. We will create a class file where we will use IdentityService. Write the following code in it:

```
ProcessEngine processEngine = ProcessEngineConfiguration
      .createStandaloneProcessEngineConfiguration()
      .setJdbcDriver("com.mysql.jdbc.Driver")
      .setJdbcUrl("jdbc:mysql://localhost:3306/activiti_book")
      .setJdbcPassword("root").setJdbcUsername("root")
      .buildProcessEngine();

  IdentityService identityService=processEngine.
getIdentityService();

  List<Group> partofuser =identityService.createGroupQuery().
groupMember("gonzo").list();

System.out.println("Gonzo is a part of following group");

for(Group partofuse:partofuser)
```

```
{
  System.out.println("Group name:"+partofuse.getName());
}
```

2. Once you are done implementing the code, we will have to execute it to view the results.
3. The result of the preceding code will list all the groups to which the specified user belongs, as follows:

```
Gonzo is a part of following group
Group name:Management
Group name:Marketing
Group name:Sales
Group name:User
```

What just happened?

We learned about IdentityService. Now, we are able to manage users and groups using IdentityService. We saw how to find a user's group using the `group` query, which is a part of IdentityService.

HistoryService

HistoryService is used to get historical data from the Activiti Engine, such as the following:

- When a particular process was started
- Which task has been completed by whom
- What the execution time was for a process or task

So, with the help of HistoryService, we can generate reports for the business process and identify the efficiency of the business process execution.

Time for action – querying historical activities

Using HistoryService, we can get historical data such as names of the finished process instances and when a process started and was completed.

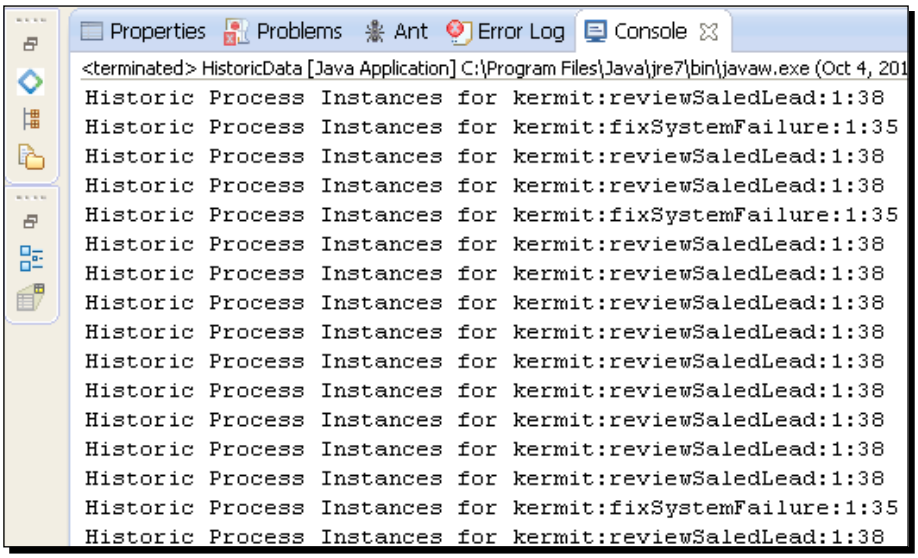
We will follow the ensuing steps to fetch some historical data from the database for a specific task:

1. We will create a new Activiti project within the Eclipse IDE.
2. Within the Activiti project, create a class file `HistoricData`.

3. Write the following code in `HistoricData`:

```
package com.activiti;

import java.security.Identity;
import java.util.HashMap;
import java.util.List;
import java.util.Map;

import org.activiti.engine.HistoryService;
import org.activiti.engine.IdentityService;
import org.activiti.engine.ProcessEngine;
import org.activiti.engine.ProcessEngineConfiguration;
import org.activiti.engine.ProcessEngines;
import org.activiti.engine.TaskService;
import org.activiti.engine.history.HistoricActivityInstance;
import org.activiti.engine.identity.Group;
import org.activiti.engine.identity.GroupQuery;
import org.activiti.engine.identity.User;
import org.activiti.engine.runtime.ProcessInstance;
import org.activiti.engine.task.Task;



public class HistoricData {

  public static void main(String ar[])
  {
  ProcessEngine processEngine = ProcessEngineConfiguration
      .createStandaloneProcessEngineConfiguration()
      .setJdbcDriver("com.mysql.jdbc.Driver")
      .setJdbcUrl("jdbc:mysql://localhost:3306/activiti_book")
      .setJdbcPassword("root").setJdbcUsername("root")
      .buildProcessEngine();

  HistoryService history=processEngine.getHistoryService();

  List<HistoricActivityInstance>   processinstance=history.
createHistoricActivityInstanceQuery().taskAssignee("kermit").
list();

  for(HistoricActivityInstance pi:processinstance)
  {
    System.out.println("Historic Process Instances for
kermit:"+pi.getProcessDefinitionId());

  }

  }
}
```

4. Execute the `HistoricData.java` class so it will display all the process instances that were assigned to the Kermit user, as shown in the following screenshot:



5. If we changed the username from `kermit` to `gonzo`, the output would be as follows:

```
Historic Process Instances for gonzo: traininngProcess:1:3282
```

What just happened?

We learned the types of historical data that are maintained by the Activiti Engine. Now we know how to use HistoryService for different purposes. We executed a task to get a list of the process instances that were assigned to a specific user.

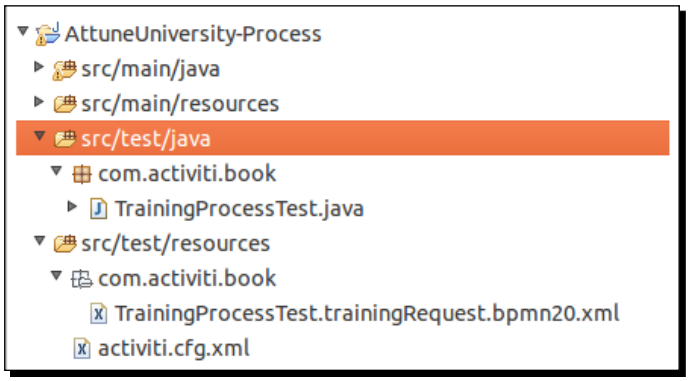
FormService

FormService is a part of the Activiti Engine. It is used to access various form data. It is also used to render forms for starting new process instances and completing tasks. Using `getStartFormData`, we can retrieve all the data necessary for rendering a form to start a new process instance.

Time for action – unit testing and debugging processes

Before deploying any piece of code into production, every organization performs some testing on it. Similarly, Activiti performs tests on the process—even the debugging of the process is possible in Activiti. Now, we will perform some testing on our **Training Schedule Process**. We will perform the following steps to do this:

1. As we are performing a test on the process, we need to create the following files within the `sr/test/java and src/test/resources` folder structure of `AttuneUniversity-Process`:

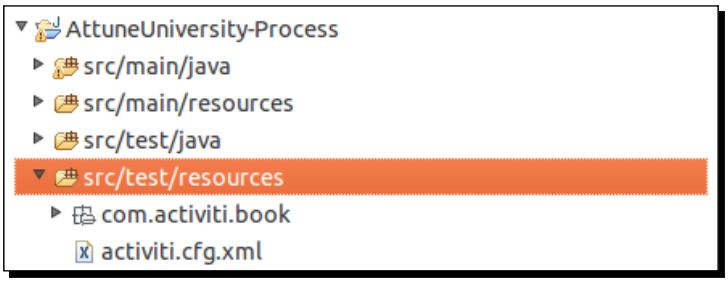


2. Create a test class file in which to perform the testing on your business process with the given code. Within the test file, we are creating a rule engine, populating the variables with values to check the output, and passing those values to the business process, as follows:

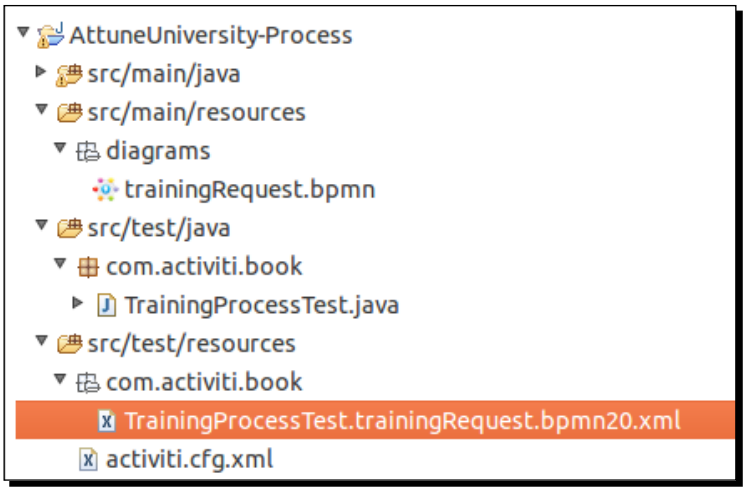
```
public class TrainingProcessTest {
  @Rule
  public ActivitiRule activitiRule = new ActivitiRule();

  @Test
  @Deployment
  public void trainingRequest() {
    Date date = new Date();
    Map<String, Object> variables = new HashMap<String, Object>();
    variables.put("customerName", "Irshad");
    variables.put("customerEamil",
    "irshad.mansuri@attuneinfocom.com");
    variables.put("trainingTopic", "Activiti");
    variables.put("trainingDate", date);
    variables.put("trainerName", "Irshad);
```

```
        variables.put("trainerMailId","Soheb.pathan@attuneinfocom.
com");

        RuntimeService runtimeService = activitiRule.
getRuntimeService();
        runtimeService.startProcessInstanceByKey("traininngProcess",
        variables);

        TaskService taskService = activitiRule.getTaskService();
        Task task = taskService.createTaskQuery().singleResult();

        assertEquals("Business Development Executive", task.
getName());

        taskService.complete(task.getId());
        assertEquals(0, runtimeService.createProcessInstanceQuery()
        .count());
    }
}
```

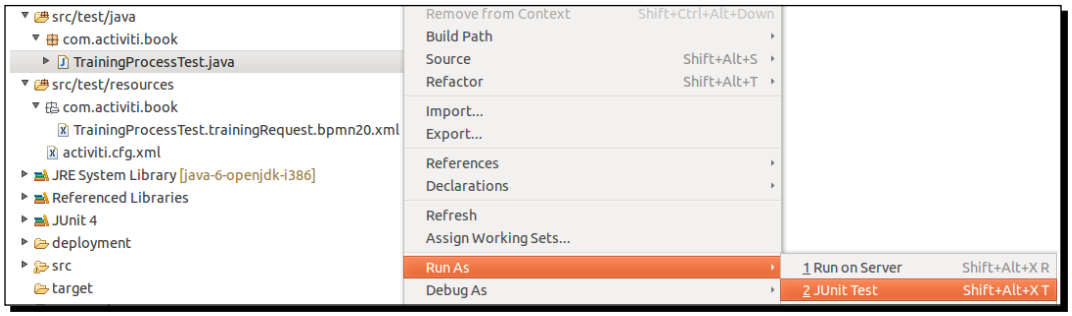
3. After creating the test file, we need to create a configuration file, `activiti.cfg.xml`, to configure the Activiti Engine. You can copy the `activiti-standalone-context.xml` file from your Activiti installation path `/Activiti-Engine/apache-tomcat-7.0.37/webapps/activiti-explorer/WEB-INF/ activiti-standalone-context.xml` and place it within `src/test/resources`, as shown in the following screenshot:



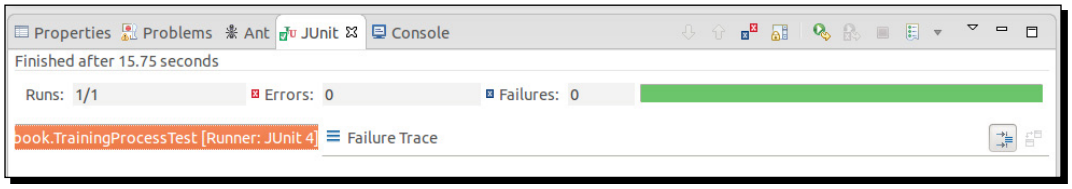
4. Now, copy your `trainingrequest.bpmn` file from the diagrams folder, paste it in the `src/test/resources` folder structure, and rename your test class file with the `bpmn20.xml` extension:



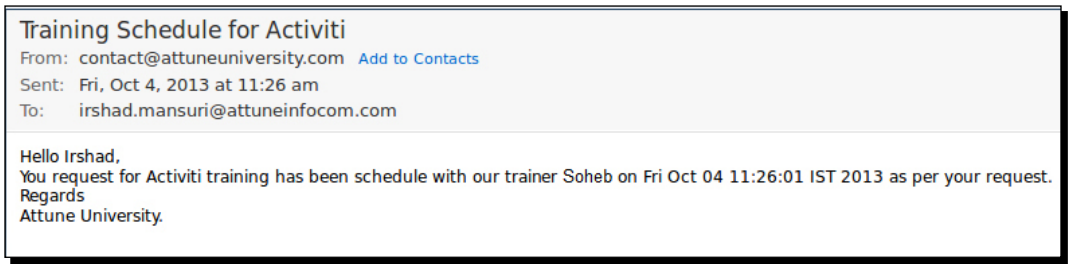
5. After performing all the preceding steps, it's now time to execute the test file. So right-click on your test class file and navigate to **Run As** | **Junit Test** as shown in the following screenshot; this will start the testing of the application:



6. To check whether or not the test was successful, you can open the Junit console in Eclipse and check it out as shown in the following screenshot; if there's any error, there would be an error stack trace. So, the following screenshot shows that the testing was successful:



7. To check the output, you can check out the e-mail you will have received about it, as shown in the following screenshot:



Training Schedule for Activiti

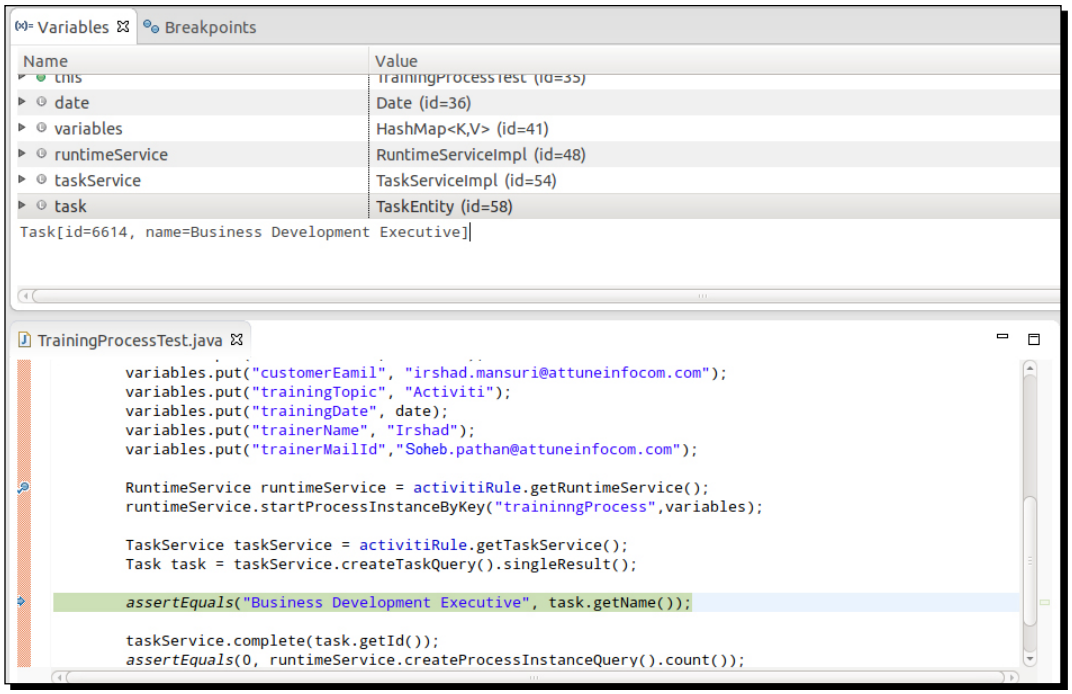
From: contact@attuneuniversity.com Add to Contacts
Sent: Fri, Oct 4, 2013 at 11:26 am
To: irshad.mansuri@attuneinfocom.com

Hello Irshad,
You request for Activiti training has been schedule with our trainer Soheb on Fri Oct 04 11:26:01 IST 2013 as per your request.
Regards
Attune University.

8. To debug your process in Eclipse, place a debug point within your code at the location from where you want to start debugging, as shown in the following screenshot:

```
public class TrainingProcessTest {
    @Rule
    public ActivitiRule activitiRule = new ActivitiRule();

    @Test
    @Deployment
    public void trainingRequest() {

        Date date = new Date();
        Map<String, Object> variables = new HashMap<String, Object>();
        variables.put("customerName", "Irshad");
        variables.put("customerEamil", "irshad.mansuri@attuneinfocom.com");
        variables.put("trainingTopic", "Activiti");
        variables.put("trainingDate", date);
        variables.put("trainerName", "Irshad");
        variables.put("trainerMailId","Soheb.pathan@attuneinfocom.com");

        RuntimeService runtimeService = activitiRule.getRuntimeService();
        runtimeService.startProcessInstanceByKey("traininngProcess",variables);

        TaskService taskService = activitiRule.getTaskService();
        Task task = taskService.createTaskQuery().singleResult();

        assertEquals("Business Development Executive", task.getName());

        taskService.complete(task.getId());
        assertEquals(0, runtimeService.createProcessInstanceQuery().count());
    }
}
```

9. To start debugging the process, execute your test code in the debug mode. So, when you right-click on your test class file, there will be an option for debugging. Select it and the debugging process will start. You can continue debugging using the *F6* key:



What just happened?

We just saw how to test and debug our business process before deploying it into the production environment. With the help of testing and debugging, we can easily identify whether or not our process will work fine. If there are any issues with the process, we can identify them with the help of a debugging operation.

Have a go hero

Having gone through the chapter, feel free to attempt the following tasks:

- Create new users and add them into groups using an API
- Find out when a process was started and completed
- Claim a task using an API
- Perform Junit testing for all the examples from our previous chapters

Pop quiz – the Activiti ProcessEngine API

Q1. Which service is at the heart of Activiti?

1. FormService
2. ProcessEngine
3. IdentityService
4. RuntimeService

Q2. Which service is used to assign a group to a specific user?

1. HistoryService
2. IdentityService
3. ProcessEngine
4. TaskService

Q3. What is the name of the service that is used for the start process instance?

1. ManagementService
2. IdentityService
3. FormService
4. RuntimeService

Summary

We learned about various Activiti services in this chapter. We have also learned how to deploy and start a process instance using APIs. We have seen how to list out various allocated tasks for a specific user. We have seen how to use HistoryService and RuntimeService. In the next chapter, we will learn about the REST API. We will see how to use various Activiti services using the REST API.

7
Working with the REST API

In the previous chapter, we provided information regarding the Activiti Process Engine's APIs and how to use the Process Engine's JAR file and APIs. In this chapter, we will work with the Process Engine using the REST service.

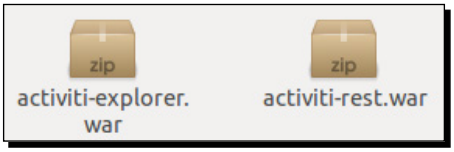
In this chapter, we will go through the following topics:

- An overview of the Activiti REST API
- Implementing REST services
- Working with Repository APIs as REST
- Using Process APIs as REST
- Using Task APIs as REST
- Working with users using REST
- Using management services as REST

With the help of REST services, we can access the process engine from any sort of device implemented in any of the languages. So, with the help of REST, there is no barrier to communicating with the Activiti Engine.

Overview of the Activiti REST API

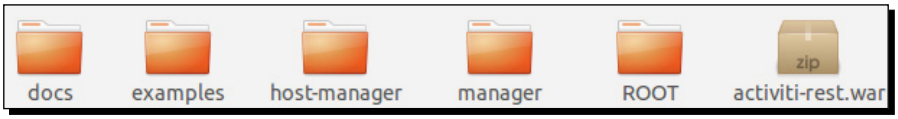
In *Chapter 1*, *Installing Activiti*, we saw that Activiti provides `activiti-rest.war` to allow you to access the process engine as a REST Service. As the `activiti-rest.war` file is provided by Activiti, we need not create a separate web service for the process engine. In *Chapter 6*, *The Activiti ProcessEngine API*, we saw that we can access the process engine using APIs by adding the respective JAR files. We can also access the Process Engine APIs without importing the respective JAR files, that is, using REST services. To access the process engine as REST, you need to deploy the `activiti-rest.war` file into a web server such as `apache-tomcat`. You will find `activiti-rest.war` in the `wars` folder at the path where you extracted `activiti-5.x.zip/activiti-5.13.0/wars`, as shown in the following screenshot:



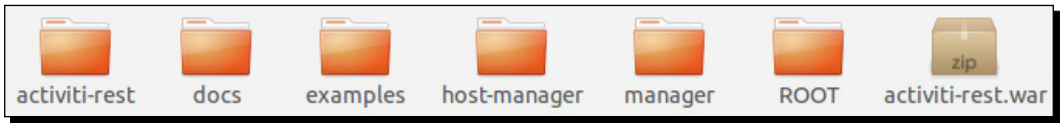
Time for action – implementing the REST service

To access Activiti as a REST service, we need to perform some configurations in our Tomcat server. Now, we will take a look at these configurations. Perform the following steps to implement the REST services:

1. As we want to access the REST services of Activiti, we will deploy the `activiti-rest.war` file in the `apache-tomcat` web server. Copy the `activiti-rest.war` file and paste it to the `webapps` folder that can be found at `apache-tomcat ../apache-tomcat-7.0.37/webapps`, as shown in the following screenshot:



2. If your Tomcat server is already running, the `activiti-rest.war` file will be deployed automatically. If Tomcat is not running, start your Tomcat server; then, your `activiti-rest.war` file will be deployed and you will be able to find the deployed folder (`activiti-rest`) in apache-tomcat's `webapps` folder (`apache-tomcat ../apache-tomcat-7.0.3/webapps`), as shown in the following screenshot:



3. After the successful deployment of the `activiti-rest` file, we need to connect the `activiti-rest` file to the database that we had configured in *Chapter 4*, *Management and Monitoring Using the Activiti Explorer*. Browse to the `classes` folder of `activiti-rest` (`../apache-tomcat-7.0.37/webapps/activiti-rest/WEB-INF/classes`) and edit the `db.properties` file shown in the following screenshot:

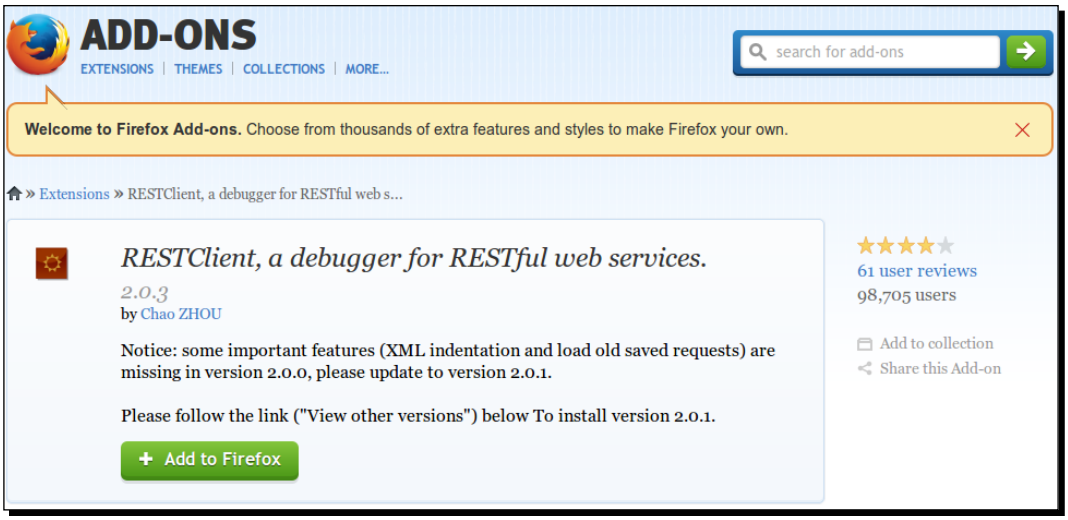
4. By default, the `db.properties` file will be connected to the H2 database; we need to change it to `mysql` (which we have configured in *Chapter 4*, *Management and Monitoring*, using the Activiti Explorer), as shown in the following code:

```
db=mysql
jdbc.driver=com.mysql.jdbc.Driver
jdbc.url=jdbc:mysql://localhost:3306/activiti
jdbc.username=root
jdbc.password=root
```

5. After executing all the preceding configurations, you need to restart your web server so that the changes are applied.

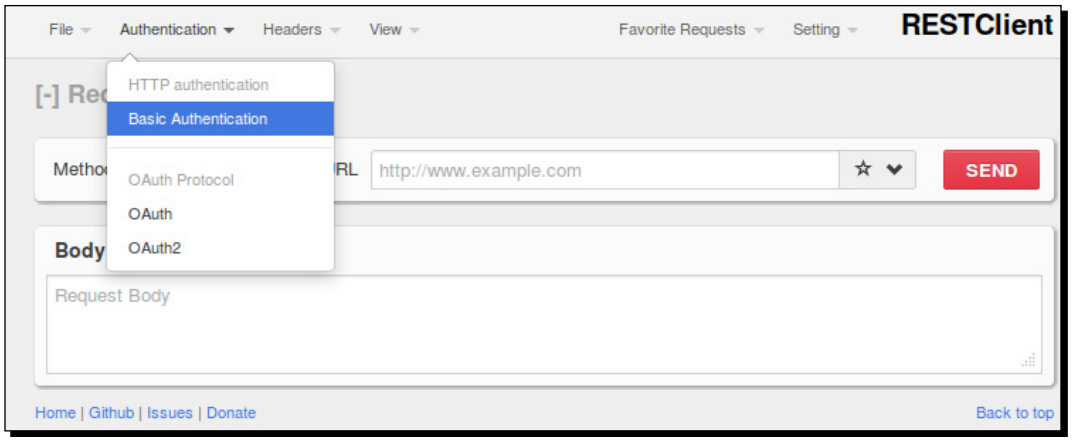
6. To check whether or not the REST service can be accessed, you need to add the RESTClient plugin to your browser.

7. To install **RESTClient** in the Firefox browser, we need to download it from the URL `https://addons.mozilla.org/en-US/firefox/addon/restclient/`. This URL will take you to the page shown in the following screenshot:

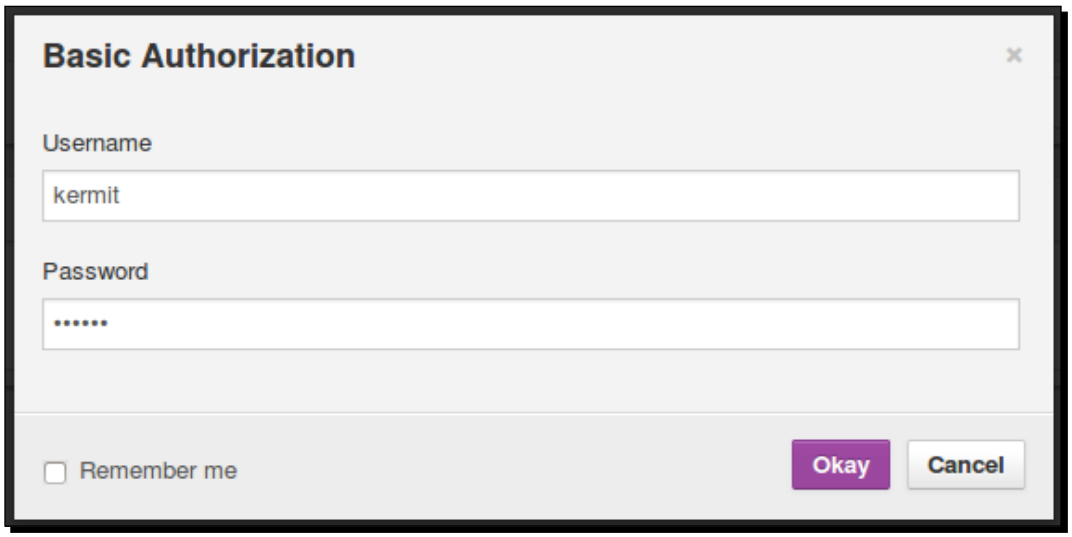


8. Once you have successfully installed the plugin, the REST icon will be displayed on the toolbar as shown in the following screenshot; you can open the RESTClient by clicking on this icon:

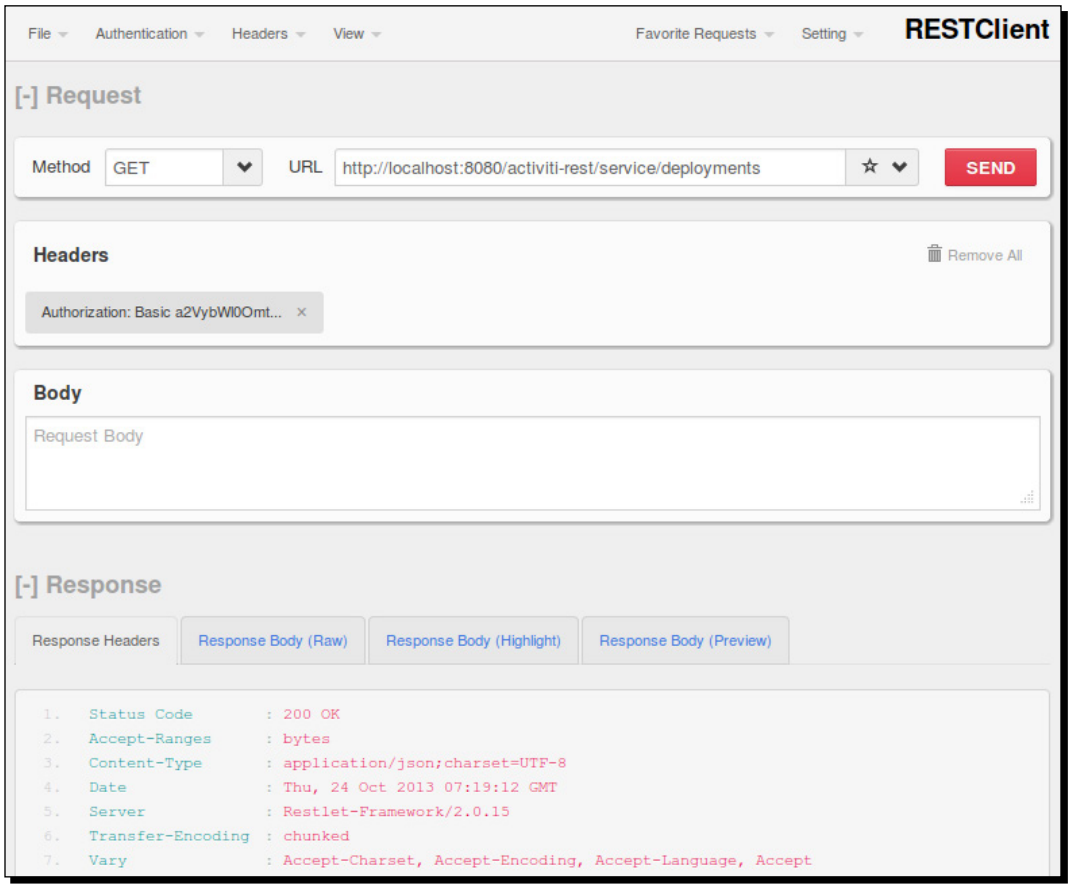
9. On opening the RESTClient, the following screenshot will appear; we then need to perform the authentication to be able to access the REST services of the Activiti Engine. Navigate to **Authentication** | **Basic Authentication**, as shown in the following screenshot:



10. A **Basic Authorization** window will pop up, where we need to specify the username and password. Enter `kermit` in the **Username** and **Password** fields for authentication, as shown in the following screenshot:



11. On successful authentication, you can access the REST API of the Activiti Engine. For testing, you can provide the URL `http://localhost:8080/activiti-rest/service/deployments` in the **URL** field in the **RESTClient** window and the **Method** field should be set to **GET**. On executing this, you should get the output in the **Response Headers** section, as shown in the following screenshot:



Before execution, the Tomcat server in which you have deployed the `activiti-rest.war` file should be running; otherwise, you won't be able to see the output.

What just happened?

We just went through how to configure `activiti-rest` and access the Activiti Process Engine by performing REST calls using RESTClient.

Working with REST API services

As we have configured the RESTClient in the browser, we will now start accessing the REST API using RESTClient.

Time for action – working with Repository APIs as REST

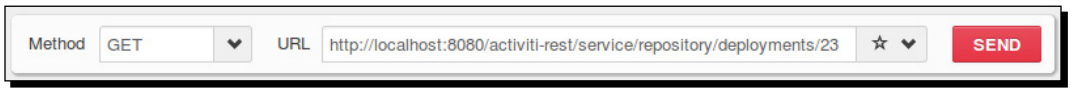
As in *Chapter 6*, *The Activiti ProcessEngine API*, we have gone through the deployment of a business process and used the Repository API for deployment purposes. In this section, we will perform the following steps to install and deploy the business process using the Repository REST API:

1. We will start by fetching all the deployed processes from the repository. For this operation, we will enter the URL `url/repository/deployments` in the browser with the **GET** method; the response will fetch all the deployed processes from the repository.

2. All the processes deployed in the process engine will be displayed when we navigate to the preceding URL, as shown in the following screenshot:

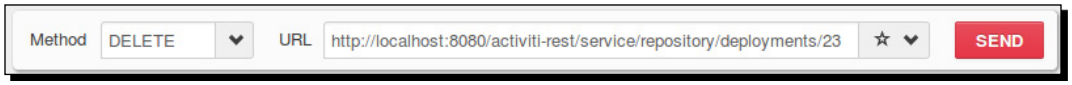

```
{
  "data": [
    {
      "id": "23",
      "name": "Demo processes",
      "deploymentTime": "2013-10-18T23:50:49IST",
      "category": null,
      "url": "http://localhost:8080/activiti-rest/service/repository/deployments/23"
    },
    {
      "id": "41",
      "name": "Demo reports",
      "deploymentTime": "2013-10-18T23:50:50IST",
      "category": null,
      "url": "http://localhost:8080/activiti-rest/service/repository/deployments/41"
    }
  ],
  "total": 2,
  "start": 0,
  "sort": "id",
  "order": "asc",
  "size": 2
}
```

3. To fetch a specific process or resource from a deployment, we need to specify the process ID with the URL, for example, `/repository /deployments/{deploymentId}`, with the **GET** method. In the following screenshot, we have provided the deployment ID `23`:

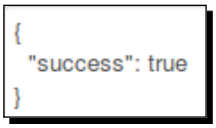


4. On navigating to the preceding URL, only the process that we want to retrieve will be displayed, as shown in the following screenshot:

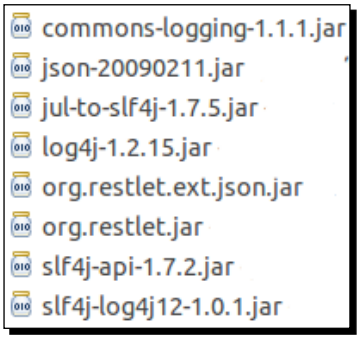
5. Apart from fetching the resources for and displaying the process, we can also delete it. To delete the process, we need to set the **Method** field to **DELETE** with the **URL** field set to `url /repository/deployment/{deploymentId}`, as shown in the following screenshot:



6. On navigating to the preceding URL, the business process will be deleted from the repository and the response will be as follows:



7. We can also achieve the preceding functionality by performing class implementation. Create a new project in Eclipse and add some external JAR files, as shown in the following screenshot, that will be required to execute this example as mentioned previously:



8. We will create a class file `Repository_Deployment` with the following code:

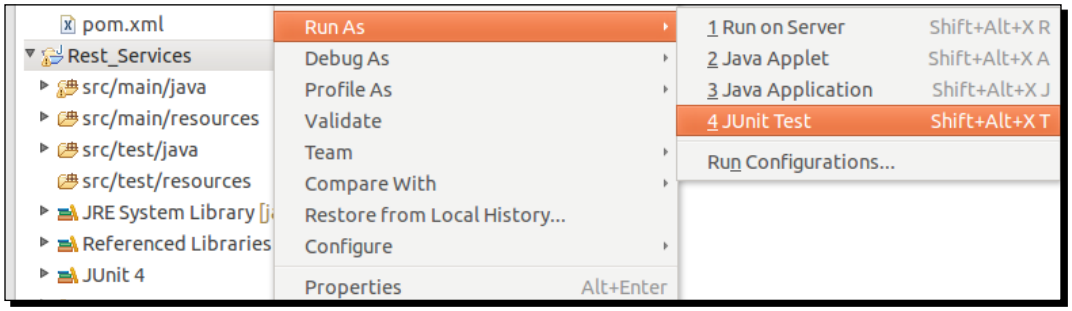
```
private static String REST_URI = "http://localhost:8080/activiti-
rest/service";

  private static ClientResource getClientResource(String uri) {
    ClientResource resource = new ClientResource(uri);
    resource.setChallengeResponse(ChallengeScheme.HTTP_BASIC,
"kermit", "kermit");
    return resource;
  }

  public static JSONArray getDeployments() throws JSONException,
IOException {
    String deploymentURI = REST_URI + "/repository/deployments";
    Representation response = getClientResource(deploymentURI).
get(
        MediaType.APPLICATION_JSON);
    JSONObject object = new JSONObject(response.getText());
    if (object != null) {
      JSONArray dataArray = (JSONArray) object.get("data");
      return dataArray;

    }
    return null;
  }
```

9. In the preceding code, we created a link to the REST URL and authenticated it with the user `kermit`; the `getDeployments()` method will fetch all the deployed processes from the engine and return it as a `JSONArray` object.

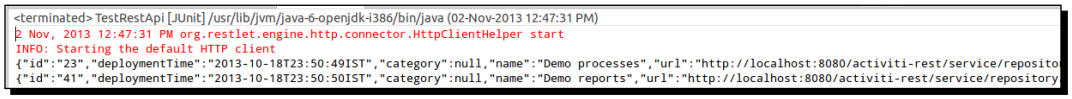
10. Now, to test the REST services, we need to create a `Test` Rest API class within `src/test/java`, which will have the following code:

```
@Test
  public void deploymentTest() throws JSONException, IOException {
    JSONArray procesArray = Repository_Deployment.
getDeployments();
    for (int i = 0; i < processArray.length(); i++) {
      JSONObject jasonObject = (JSONObject) processArray.get(i);
      assertNotNull(jasonObject);
      System.out.println(jasonObject.toString());
    }
  }
```

11. To test the given code, you can right-click on the project and navigate to **Run As** | **JUnit Test**, as shown in the following screenshot:



12. On successful execution of the REST service, the output will show the number of processes deployed into the process engine, as shown in the following screenshot:



13. Now we will provide a URL with a process ID to retrieve a specific process from the repository. We will create a method in the `Repository_Deployment` class that takes the process ID as an input parameter and provides the result for that process, as shown in the following code:

```
public static JSONObject getDeployments(int id) throws
JSONException, IOException {
    String deploymentURI = REST_URI + "/repository/deployments/" +
id;
    Representation response = getClientResource(deploymentURI).
get(
        MediaType.APPLICATION_JSON);
    JSONObject object = new JSONObject(response.getText());
    if (object != null) {
      return object;
    }
    return null;
  }
```

14. To test our business process, we will create a method through which we will pass the process ID and retrieve the details of that process, as shown in the following code:

```
    @Test
  public void deploymentById() throws JSONException, IOException {
    JSONObject processObject = Repository_Deployment.
getDeployments(23);
    assertNotNull(processObject);
    System.out.println(processObject.toString());
  }
```

15. We have provided an ID to fetch a unique process, so on executing the test file the following output will be displayed:

```
<terminated> TestRestApi [JUnit] /usr/lib/jvm/java-6-openjdk-i386/bin/java (02-Nov-2013 1:04:49 PM)
2 Nov, 2013 1:04:49 PM org.restlet.engine.http.connector.HttpClientHelper start
INFO: Starting the default HTTP client
{"id":"23","deploymentTime":"2013-10-18T23:50:49IST","category":null,"name":"Demo processes","url":"http://localhost:8080/activiti-rest/service/repositor
```

16. So far, we have seen how to get the process deployment files; now we will learn how to delete a process from the repository. Create the `deleteDeployment` method in the `Repository_Deployment` class; here in the response, we are calling the `delete` method, as shown in the following code:

```
public static void deleteDeployment(int id) {
    String deploymentURI = REST_URI + "/repository/deployments/" +
id;
  Representation response = getClientResource(deploymentURI).
delete(
        MediaType.ALL);
  }
```

17. Now, to access this method in the test file, create a test method from which you will make a call to the `deleteDeployment` method of `Repository_Deployment`; on performing the test, the process with the ID `23` will be deleted from the repository, as shown in the following code:

```
@Test
  public void deleteDeployment() throws JSONException, IOException
{
    Repository_Deployment.deleteDeployment(23);
       }
```

18. To retrieve the resources from a specific deployment, we have to navigate to `url /repository/deployments/{deploymentId}/resources`.

19. We will create the `getDeploymentResources` method in the `Repository_Deployments` class, where we will pass the deployment ID to access its resources, as shown in the following code:

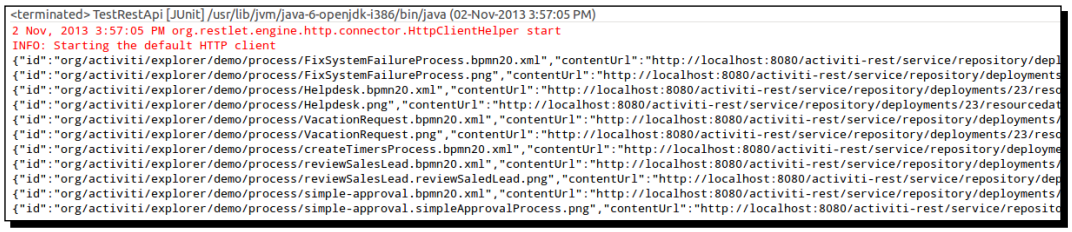
```
public static JSONArray getDeploymentResources(int id) throws
IOException,
      JSONException {
    String deploymentURI = REST_URI + "/repository/deployments/" +
id + "/resources";
    Representation response = getClientResource(deploymentURI).
get( MediaType.APPLICATION_JSON);

    String json = "{data :" + response.getText() + "}";

    JSONObject object = new JSONObject(json);

    if (object != null) {
      JSONArray arr = (JSONArray) object.get("data");
```

```
        return arr;
      }
      return null;
    }
```

20. To test, we will create the `deploymentResource` method in the `Repository_Deployment` class as follows:

```
@Test
  public void deploymentResource() throws JSONException,
IOException {

    JSONArray processArray = Repository_Deployment.
getDeploymentResources(23);
    assertNotNull(processArray);

    for (int i=0;i<processArray.length();i++){
      JSONObject object = (JSONObject) processArray.get(i);
      assertNotNull(object);
      System.out.println(object.toString());
    }
  }
```

21. On running the test file, we should be able to view all the resources of the deployment, as shown in the following screenshot:



What just happened?

We just saw how to use the REST API to deploy and delete the process. We have also fetched the resources from a specified deployment and created some test cases to test our REST API calls.

Time for action – working with processes

We are comfortable with fetching deployments from the repositories. Now, we will become familiar with fetching the `process-definitions` folder from the repository; you can paste the URL `http://localhost:8080/activiti-rest/service/repository/process-definitions` in RESTClient. If an error 400 occurs, as shown in the following screenshot, we will have to modify the `activiti-rest` folder:

To resolve this error, we have to perform the following steps:

1. If you get the preceding error, first of all, modify your `activiti-context.xml` file located in the `../apache-tomcat-7.0.37/webapps/activiti-rest/WEB-INF/classes` folder, as shown in the following screenshot:

2. Edit the `actviti-context.xml` file by placing the following code in the `processEngineConfiguration` bean:

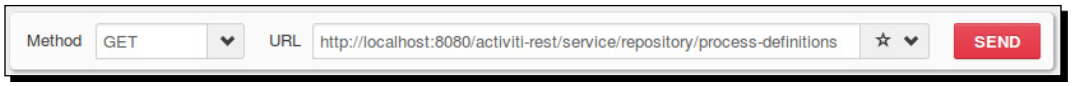
```
    <property name="customFormTypes">
      <list>
        <bean class="org.activiti.explorer.form.UserFormType"/>
        <bean class="org.activiti.explorer.form.
ProcessDefinitionFormType"/>
        <bean class="org.activiti.explorer.form.MonthFormType"/>
      </list>
    </property>
```

3. Your modified `activiti-context.xml` file should look as shown in the following screenshot:

```
<bean id="processEngineConfiguration" class="org.activiti.spring.SpringProcessEngineConfiguration">
        <property name="dataSource" ref="dataSource" />
        <property name="transactionManager" ref="transactionManager" />
        <property name="databaseSchemaUpdate" value="true" />
        <property name="mailServerHost" value="localhost" />
        <property name="mailServerPort" value="5025" />
        <property name="jobExecutorActivate" value="false" />
        <property name="customFormTypes">
         <list>
                <bean class="org.activiti.explorer.form.UserFormType"/>
                <bean class="org.activiti.explorer.form.ProcessDefinitionFormType"/>
                <bean class="org.activiti.explorer.form.MonthFormType"/>
        </list>
        </property>
        </bean>
```

4. After editing the `activity-context.xml` file, you need to add the `activity-explorer{version}.jar` file to the `../apache-tomcat-7.0.37/webapps/activiti-rest/WEB-INF/lib` folder.

5. Now, restart the Tomcat server and from the RESTClient, you can retrieve `process-definitions` by setting the **GET** method along with the URL `../repository/process-definitions`, as shown in the following screenshot:



6. On navigating to the preceding REST URL, you should be able to view the list of process definitions available within the repository as shown in the following screenshot:

```
{
  "data": [
    {
      "id": "createTimersProcess:1:37",
      "url": "http://localhost:8080/activiti-rest/service/repository/process-definitions/createTimersProcess%3A1%3A37",
      "key": "createTimersProcess",
      "version": 1,
      "name": "Create timers process",
      "description": "Test process to create a number of timers.",
      "deploymentId": "23",
      "deploymentUrl": "http://localhost:8080/activiti-rest/service/repository/deployments/23",
      "resource": "http://localhost:8080/activiti-rest/service/repository/deployments/23/resources
/org%2Factiviti%2Fexplorer%2Fdemo%2Fprocess%2FcreateTimersProcess.bpmn20.xml",
      "diagramResource": null,
      "category": "Examples",
      "graphicalNotationDefined": false,
      "suspended": false,
      "startFormDefined": false
    },
    {
      "id": "employee-productivity-report:1:47",
      "url": "http://localhost:8080/activiti-rest/service/repository/process-definitions/employee-productivity-report%3A1%3A47",
      "key": "employee-productivity-report",
      "version": 1,
      "name": "Employee productivity",
      "description": null,
      "deploymentId": "41",
      "deploymentUrl": "http://localhost:8080/activiti-rest/service/repository/deployments/41",
      "resource": "http://localhost:8080/activiti-rest/service/repository/deployments/41/resources
/org%2Factiviti%2Fexplorer%2Fdemo%2Fprocess%2Freports%2FemployeeProductivity.bpmn20.xml",
      "diagramResource": null,
      "category": "activiti-report",
```


7. To access `process-definitions` within a class, we need to create a `Process_Deployment` class in which we will create a method to fetch the process definitions, as shown in the following code:

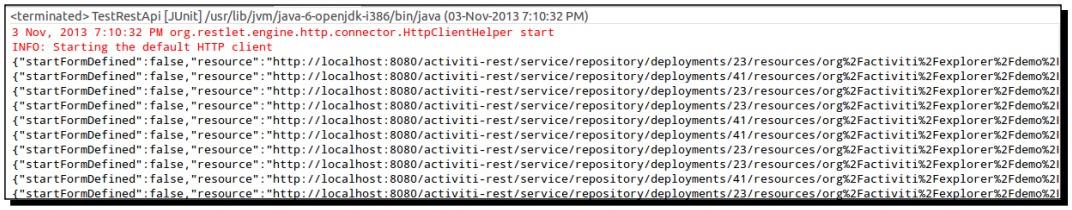
```
private static String REST_URI = "http://localhost:8080/activiti-
rest/service";
  private static ClientResource getClientResource(String uri) {
    ClientResource resource = new ClientResource(uri);
  resource.setChallengeResponse(ChallengeScheme.HTTP_BASIC,
"kermit", "kermit");
    return resource;
  }
  public static JSONArray getProcessDefinitions() throws
JSONException, IOException {
    String uri = REST_URI + "/repository/process-definitions";
```

```
     Representation response = getClientResource(uri).
get(MediaType.ALL);
    JSONObject object = new JSONObject(response.getText());
    if (object != null) {
      JSONArray processArray = (JSONArray) object.get("data");
      return processArray;
    }
    return null;
  }
```

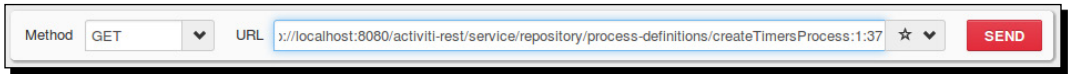
8. To fetch the process definitions, we need to create a test method in the test file as follows:

```
@Test
  public void processDefinitons() throws Exception {
    JSONArray processArray = Process_Deployment.
getProcessDefinitions();
    for (int i = 0; i < processArray.length(); i++) {
      JSONObject jsonObject = (JSONObject) processArray.get(i);
      assertNotNull(jsonObject);
      System.out.println(jsonObject.toString());
    }
  }
```

9. On testing the method, we should get all the processes deployed in the repository. All the processes will be displayed, as shown in the following screenshot:



10. We can also fetch a specific process from the repository by providing the process ID with the **GET** method and the URL `repository/process-definition/{processeId}`, as shown in the following screenshot:



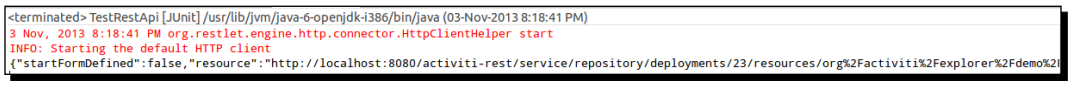
11. We will create an implementation class with the `getProcessDefinitionById` method in the `Process_Deployment` class, which will retrieve the process from the repository based on the process ID provided, as shown in the following code:

```
public static JSONObject getProcessDefinitionById(String
processId) throws JSONException, IOException {
    String uri = REST_URI + "/repository/process-definitions/"+
processId;
    Representation response = getClientResource(uri).
get(MediaType.APPLICATION_JSON);
    JSONObject object = new JSONObject(response.getText());
    if (object != null) {
      return object;
    }
    return null;
  }
```

12. Now, we will create the `processDefinitionById` method in the Test REST API class to test the results, as shown in the following code:

```
@Test
  public void processDefinitionById() throws Exception {
    JSONObject jsonObject = Process_Deployment.getProcessDefinitio
nById("createTimersProcess:1:37");
    assertNotNull(jsonObject);
    System.out.println(jsonObject.toString());
  }
```

13. So, on execution of the test case, we get the details of the process specified within it, as shown in the following screenshot:



14. Using the REST API, we can not only fetch the process definitions, but also update the processes by setting the **Method** type to **PUT**. We will create the `updateProcess` method to update the process in the `Process_Deployment` class and provide the `processId` argument for which we want to perform the update, as shown in the following code:

```
  public static JSONObject updateProcess(String processId) throws
JSONException, IOException {

    String uri = REST_URI + "/repository/process-definitions/" +
processId;
    JSONObject updateData = new JSONObject();
    updateData.put("category", "Updated Category");
```

```
    Representation response = getClientResource(uri).
put(updateData);
    JSONObject object = new JSONObject(response.getText());
    if (object != null) {
      return object;
    }
    return null;
  }
```

15. Create a test method `update` in the Test REST API class file to perform the testing of the `update` method, as shown in the following code:

```
@Test
  public void update() throws Exception{
System.out.println("Before Updating Category");
JSONObject object1 = Process_Deployment. getProcessDefinitionById
("vacationRequest:1:33");
    assertNotNull(object1);
    System.out.println(object1.toString());
    System.out.println("After Updating Category");
JSONObject object2 = Process_Deployment.UpdateProcess("vacationReq
uest:1:33");
    assertNotNull(object2);
    System.out.println(object2.toString());
}
```

16. On executing the test class, the `Category` parameter should be updated with the value provided in the `updateProcess` method, which will produce the output as shown in the following screenshot:

17. As we can update the process by setting the **Method** value to **PUT**, we can also manage the action parameter using the **PUT** method as follows:

```
public static JSONObject suspendProcessDefination(String id)
throws JSONException, IOException {
    String uri = REST_URI + "/repository/process-definitions/" +
id;

    JSONObject my_data = new JSONObject();
    my_data.put("action", "suspend");
```

```
    my_data.put("includeProcessInstances", "false");

    Representation response = getClientResource(uri).put(my_data);
    JSONObject object = new JSONObject(response.getText());
    if (object != null) {
      return object;
    }
    return null;
}
```

18. In the preceding code, we have set the `action` parameter to suspend the process. This action will suspend the process so that it cannot be executed. If you want to activate the process, you need to change the value of the `action` parameter from `suspend` to `activate`, as shown in the following code:

```
public static JSONObject activateProcessDefination(String id)
throws JSONException, IOException {
    String uri = REST_URI + "/repository/process-definitions/"+
id;
    JSONObject my_data = new JSONObject();
    my_data.put("action", "activate");
    my_data.put("includeProcessInstances", "true");

    Representation response = getClientResource(uri).put(my_data);
    JSONObject object = new JSONObject(response.getText());
    if (object != null) {
      return object;
    }
    return null;
  }
```

19. Using the REST API, we can also assign users to the business process. Now, we will see how to assign users to the business process. We will create the `addCandidate` method in the `Process_Deployment` class as follows:

```
  public static JSONObject addCandidate(String id) throws
JSONException, IOException {
    String uri = REST_URI + "/repository/process-definitions/" +
id + "/identitylinks";
    JSONObject my_data = new JSONObject();
    my_data.put("user", "kermit");
    Representation response = getClientResource(uri).post(my_
data);
    JSONObject object = new JSONObject(response.getText());
    if (object != null) {
      return object;
     }
     return null;
  }
```

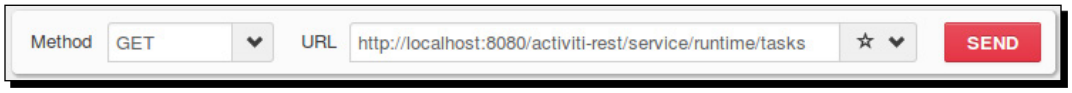
What just happened?

So far, we have used the REST API to retrieve and edit the process using the `PUT` method. We have also gone through updating the `action` parameter by specifying an action to the process, such as suspending and activating the process or adding users to it.

Time for action – working with tasks

We have used the REST API for fetching, updating, and adding actions to the business process and will also use it when we want to work with tasks. To access a task, we have to use the GET method with the URL `/runtime/tasks` and perform the following steps:

1. First, we have to fetch processes by setting the **Method** field to **GET** and the **URL** field to `/runtime/tasks`, as shown in the following screenshot:



2. By navigating to the preceding URL, we will get a list of the tasks available in the repository with all the details, as shown in the following screenshot:

3. We can also use the REST API to access the tasks from the repository by adding the `getTasks` method in the `Process_Deployment` class, which will provide the list of tasks, as shown in the following code:

```
public static JSONArray getTasks() throws JSONException,
IOException {
    String uri = REST_URI + "/runtime/tasks";
    Representation response = getClientResource(uri).
get(MediaType.APPLICATION_JSON);
    JSONObject object = new JSONObject(response.getText());
```

```
    if (object != null) {
      JSONArray taskArray = (JSONArray) object.get("data");
      return taskArray;
    }
    return null;
  }
```

4. We can fetch a specific process based on its process ID; similarly, we can retrieve a specific task based on its task ID. For this, we have to specify this URL `../runtime/tasks/{taskId}` in combination with the `GET` method. To execute the task, we need it to be claimed by a specific person using the `POST` method. For that, we need to set values for the `action` and `assignee` parameters. We can create the `claimTask` method in the `Process_Deployment` class as follows:

```
public static void claimTask(int id) throws JSONException,
IOException {
    String uri = REST_URI + "/runtime/tasks/" + id;
      JSONObject my_data = new JSONObject();
      my_data.put("action", "claim");
      my_data.put("assignee", "kermit");
      Representation response = getClientResource(uri).post(my_
data);
  }
```

5. After claiming the task, we have to make sure that it is completed to make the process work smoothly. We need to set the `action` parameter to `complete` along with the `POST` method, as shown in the following code:

```
public static void completeTask(int id) throws JSONException,
IOException {
    String uri = REST_URI + "/runtime/tasks/" + id;
    JSONObject my_data = new JSONObject();
     my_data.put("action", "complete");
     Representation response = getClientResource(uri).post(my_
data);
    }
```

What just happened?

We are done learning about the Task API: we got the list of tasks available within the repository, fetched a specific task based on the task ID, and then changed the `action` parameter for claiming the task. After claiming the task, the task was completed.

Time for action – working with users

To access user-related information, we have to use an identity-related URL, that is, `/identity/users`. Now, we will use the identity service. Perform the following steps to archive the identity service:

1. Add a user to the Activiti Engine using the `createUser` method in the `Process_Deployment` class through which we will pass the details of the user as follows:

```
public static JSONObject createUser() throws JSONException,
IOException {
    String uri = REST_URI + "/identity/users";
    JSONObject my_data = new JSONObject();
    my_data.put("id", "irshad");
    my_data.put("firstName", "Irshad");
    my_data.put("lastName", "Mansuri");
    my_data.put("email", "irshad@localhost.com");
    my_data.put("password", "irshad");
    Representation response = getClientResource(uri).post(my_
data);
    JSONObject object = new JSONObject(response.getText());
    if (object != null) {
      return object;
    }
    return null;
  }
```

2. We can create a user using the REST API and also update and delete them with it.
3. To update a user, we can use the same method as we did for creating a user, except that we have to provide a user `id` for the update. The URL for the update is `/identity/users/{userId}`, as shown in the following code:

```
public static JSONObject updateUser(String id) throws
JSONException, IOException {
    String uri = REST_URI + "/identity/users/" + id;
    JSONObject my_data = new JSONObject();
    my_data.put("email", "irshad.mansuri@localhost.com");
    Representation response = getClientResource(uri).put(my_data);
    JSONObject object = new JSONObject(response.getText());
    if (object != null) {
      return object;
    }
    return null;
}
```

4. To delete a user from the repository, we have to use the `deleteUser` method. We will add the following method to archive the delete functionality:

```
public static void deleteUser(String id) throws JSONException,
IOException {
    String uri = REST_URI + "/identity/users/" + id;
             Representation response = getClientResource(uri).
delete(MediaType.ALL);
}
```

5. With the help of the identity service, we can not only access the users but also create and update groups. For groups, we have to provide the URL `/identity/groups`.

6. Now we will use the `createGroup` method to create a group in the repository, which will contain information regarding the group, as shown in the following code:

```
  public static JSONObject createGroup() throws JSONException,
IOException {
    String uri = REST_URI + "/identity/groups";

    JSONObject my_data = new JSONObject();
    my_data.put("id", "attunegroup");
    my_data.put("name", "Attune Group");
    my_data.put("type", "security-role");
    Representation response = getClientResource(uri).post(my_
data);
    JSONObject object = new JSONObject(response.getText());
    if (object != null) {
      return object;
    }
    return null;
  }
```

7. As we know how to update a user using identity services, we can also update a group.

8. Once we have created a group, we need to add some users to it. For that, we need to create the `addGroupMember` method where we will provide the group ID as the parameter in which we want to add users, as follows:

```
  public static JSONObject addGroupMember(String id) throws
JSONException, IOException {
    String uri = REST_URI + "/identity/groups/" + id + "/members";

    JSONObject my_data = new JSONObject();
    my_data.put("userId", "irshad");

    Representation response = getClientResource(uri).post(my_data);
```

```
    JSONObject object = new JSONObject(response.getText());
    if (object != null) {
      return object;
    }
    return null;
  }
```

9. Just as we can add a member to a group, we can also delete them from it. For that, we need to provide the `groupID` as well as `userId` values that we want to delete, as shown in the following code:

```
  public static void deleteGroupMember(String groupId, String
userId) throws JSONException, IOException {
    String uri = REST_URI + "/identity/groups/" + groupId + "/
members/"  + userId;
    Representation response = getClientResource(uri).
delete(MediaType.ALL);
  }
```

What just happened?

By the end of this section, you should be able to list, add, and manage users and groups in the repository.

Time for action – working with management

Now we are familiar with all the APIs for fetching processes from the repository, updating and assigning users to a business process, creating and managing users with the Identity API, and creating groups and assigning users to it. We have used the Task API to claim it and complete a task. Now we will have a look at the Management API, which we will use to manage our process engine tables. To work with management, we have to provide the URL `/management/tables` using the GET method and perform the following steps to archive the management-related information:

1. We will access the management-related tables in the process engine by providing the **URL** value `/management/tables` with the **GET** method, as shown in the following screenshot:

2. The response generated by the preceding URL will be the list of tables available within the process engine, as shown in the following screenshot:


```
[
 {
  "name": "ACT_RU_VARIABLE",
  "url": "http://localhost:8080/activiti-rest/service/management/tables/ACT_RU_VARIABLE",
  "count": 0
 },
 {
  "name": "ACT_RU_EVENT_SUBSCR",
  "url": "http://localhost:8080/activiti-rest/service/management/tables/ACT_RU_EVENT_SUBSCR",
  "count": 0
 },
 {
  "name": "ACT_RE_DEPLOYMENT",
  "url": "http://localhost:8080/activiti-rest/service/management/tables/ACT_RE_DEPLOYMENT",
  "count": 2
 },
 {
  "name": "ACT_RE_PROCDEF",
  "url": "http://localhost:8080/activiti-rest/service/management/tables/ACT_RE_PROCDEF",
  "count": 10
 },
 {
  "name": "ACT_HI_TASKINST",
  "url": "http://localhost:8080/activiti-rest/service/management/tables/ACT_HI_TASKINST",
  "count": 5112
 },
 {
  "name": "ACT_HI_ACTINST",
```


3. To fetch a specific table from the list, we need to provide its name with the help of the URL `/management/tables/{tableName}` and the `get` method, as shown in the following code:

```
public static JSONObject getDbTables(String tableName) throws
JSONException, IOException {
    String uri = REST_URI + "/management/tables/" + tableName;
    Representation response = getClientResource(uri).get(
MediaType.APPLICATION_JSON);

    JSONObject object = new JSONObject(response.getText());
    if (object != null) {
      return object;
    }
    return null;
  }
```

4. We need to create the `getSpecificTable` method to fetch a specific table from the list in the `Test` REST API class. The following code fetches the `ACT_RU_VARIABLE` value, which contains the list of variables that are currently running:

```
@Test
  public void getSpecificTable() throws Exception {
    JSONObject object = Process_Deployment.getDbTables("ACT_RU_
VARIABLE");
    assertNotNull(object);
    System.out.println(object.toString());
  }
```

5. The Management API is not limited to only fetching tables; it can also fetch columns from a given table. The URL for fetching columns is `/management/tables/{tableName}/columns` and is used along with the GET method.

6. We will create the `getDbTablesColumn` method in the `Process_Deployment` class to fetch the columns of a given table, as shown in the following code:

```
public static JSONObject getDbTablesColumn(String name) throws
JSONException, IOException {
    String uri = REST_URI + "/management/tables/" + name + "/
columns";
    Representation response = getClientResource(uri).get(
MediaType.APPLICATION_JSON);

    JSONObject object = new JSONObject(response.getText());
    if (object != null) {
      return object;
    }
    return null;
  }
```

7. With the help of the Management API, we can also get the properties of the Activiti Process Engine that is running. For that, we can use the URL `/management/properties` in combination with the `get` method.

8. We will create the `getEngineProperty` method to fetch the engine properties in the `Process_Deployment` class as follows:

```
public static JSONObject getEngineProperty() throws JSONException,
IOException {
    String uri = REST_URI + "/management/properties";
    Representation response = getClientResource(uri).get(
MediaType.APPLICATION_JSON);

    JSONObject object = new JSONObject(response.getText());
```

```
    if (object != null) {
      return object;
    }
    return null;
  }
```

9. In the Test REST API, we will create the `getProperties` method, which will test the fetching of the engine properties, as shown in the following code:

```
@Test
  public void getProperties() throws Exception {
    JSONObject object = Process_Deployment.getEngineProperty();
    assertNotNull(object);
    System.out.println(object.toString());
  }
```

10. The response generated on the execution of the `Test` class will give us the details of our database history and the version of the tables that we are using for our process engine, as shown in the following screenshot:

```
<terminated> TestRestApi [JUnit] /usr/lib/jvm/java-6-openjdk-i386/bin/java (05-Nov-2013 12:05:56 PM)
5 Nov, 2013 12:05:58 PM org.restlet.engine.http.connector.HttpClientHelper start
INFO: Starting the default HTTP client
{"next.dbid":"34301","schema.history":"create(5.13)","schema.version":"5.13"}
```

11. We can also fetch engine-related information using the Management API; the URL to access the information is `/management/engine`, as shown in the following code:

```
public static JSONObject getEngineInfo() throws JSONException,
IOException {
    String uri = REST_URI + "/management/engine";
    Representation response = getClientResource(uri).get(
MediaType.APPLICATION_JSON);
    JSONObject object = new JSONObject(response.getText());
    if (object != null) {
      return object;
    }
    return null;
  }
```

12. To test our `getEngineInfo` method, we need to create the test method `getEngineInfo` in the Test REST API class as follows:

```
@Test
  public void getEngineInfo() throws Exception {
    JSONObject object = Process_Deployment.getEngineInfo();
    assertNotNull(object);
    System.out.println(object.toString());
  }
```

13. The output of our test class will provide information regarding the version of Activiti and the path of the `activiti-context.xml` file, as shown in the following screenshot:

```
activiti-rest/WEB-INF/classes/activiti-context.xml","version":"5.13"}
```

What just happened?

We learned the use of the Management REST API by fetching tables from the repository and columns from a specific table—all this using the same REST API from which we can fetch the properties and engine-related information.

Have a go hero

Now that we have completed the chapter, you should be able to attempt the following tasks using the REST APIs:

- Perform delete, add, and update operations for variables on the tasks.
- Using the Process API, we can start a process instance and execute queries using the Process API. Using the Identity API, we can update and delete users and groups.

Pop quiz – the REST API

Q1. Which API is used to fetch deployment information from the repository?

1. The Repository API
2. The Task API
3. The Process API
4. None of the above

Q2. The Process API is used for fetching which of the following information from the repository?

1. process-definitions
2. deployments
3. users
4. groups

Q3. Which method is used in combination with the Process API to update any information?

1. GET
2. PUT
3. POST
4. DELETE

Summary

In this chapter, we have gained knowledge regarding the use of the REST API. Using the REST API, we can perform all the operations that we can perform using the ProcessEngine API. We have understood the use of the Task, Process, Identity, and Management REST APIs and used them for developing our process. In the next chapter, we will learn how to integrate various third-party tools with Activiti, such as the Liferay Portal, Business Rules, and OSGI.

8
Integrating Activiti with Third-party Services

In the previous chapter, we worked with the REST API provided by Activiti to implement and execute our business process. In this chapter, we will integrate various third-party tools with our Activiti Engine.

In this chapter, we will go through the following topics:

- Using the Liferay Portal with Activiti (`http://www.liferay.com/`)
- Integrating business rules (Drools Expert) with Activiti (`http://www.jboss.org/drools/drools-expert.html`)
- Deploying Activiti as an OSGi bundle (`http://www.OSGi.org/Main/HomePage`)

So far, we just focused on executing process definitions using the Activiti process engine. Now, let's say there's a requirement to manage the documents in Alfresco using Activiti; therefore, we will need to integrate them.

Using the Liferay Portal with Activiti

To start using Activiti with the Liferay Portal, we need to understand what a portal is and what the Liferay Portal is. So, we will first discuss portals, then the Liferay Portal, and finally, integrate Activiti with Liferay.

Understanding portals

A portal is where we can get information on various different topics from a single location. In the web world, a web portal refers to a website that offers access to various services, such as search results for other websites, and information regarding finance, health, the weather, and so on. The difference between a website and a web portal is that a web portal can store large volumes of data whereas a website can store only limited quantities of data. An example of a web portal is `www.yahoo.com`, where we can access various kinds of information, including the latest news, the weather, finance, and so on.

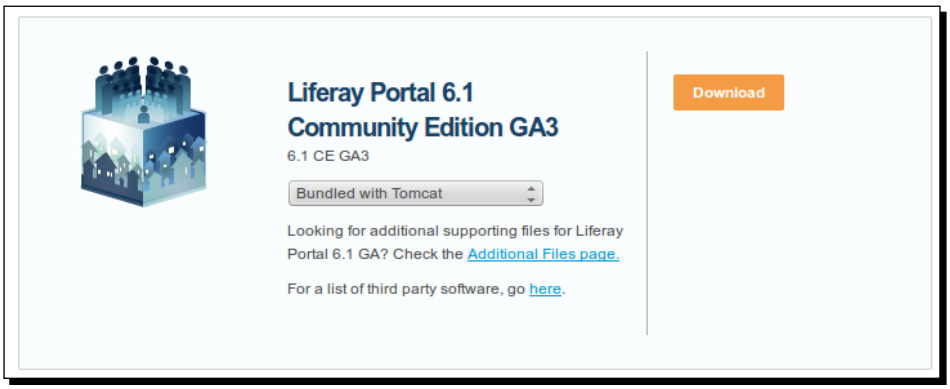
Exploring the Liferay Portal

Liferay is a web-based Java portal. Using Liferay, we can establish a portal on various platforms, such as the web platform, a **web content management** (**WCM**) system, the Integration platform, the Collaboration platform, or the Social platform. In this section, we will use Liferay-6.1 to implement our workflow. To do so, we will use Activiti.

Time for action – integrating Liferay with Activiti

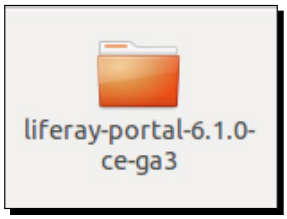
To start integrating Liferay with Activiti, we need to install Liferay on our system. For the installation, we need to perform the following steps:

1. We can download the Liferay Version 6.1 Community Edition with the Tomcat bundle from `http://www.liferay.com/downloads/liferay-portal/available-releases`, as shown in the following screenshot:

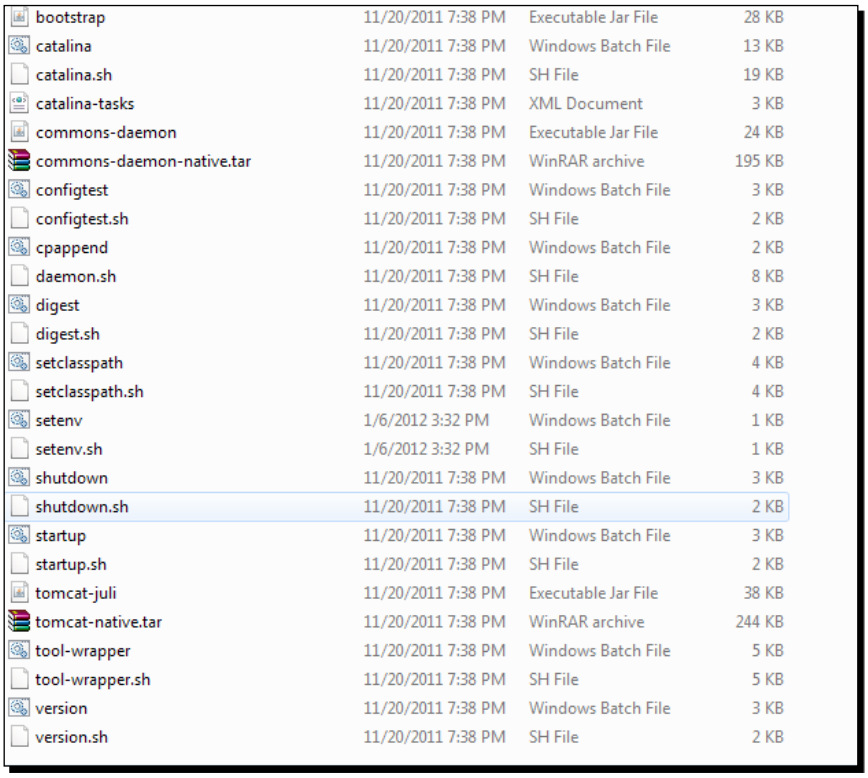


2. Along with Liferay, you will get various ZIP files downloaded to your system. Within those ZIP files, there is a `liferay-portal-tomcat-6.1.1-ce-ga3.zip` file that contains the Tomcat bundled with Liferay.

3. On extracting the ZIP file to a specific location, you will get a folder structure as seen in the following screenshot, which will contain the folder in it:



4. To start the Liferay Portal, browse to the `bin` folder in `liferay-portal-6.1.0-ce-ga3 ../liferay-portal-6.1.0-ce-ga3/tomcat-7.0.23/bin`. If you are using a Windows system, you will have to execute the `startup.bat` file from the `bin` folder, as shown in the following screenshot:



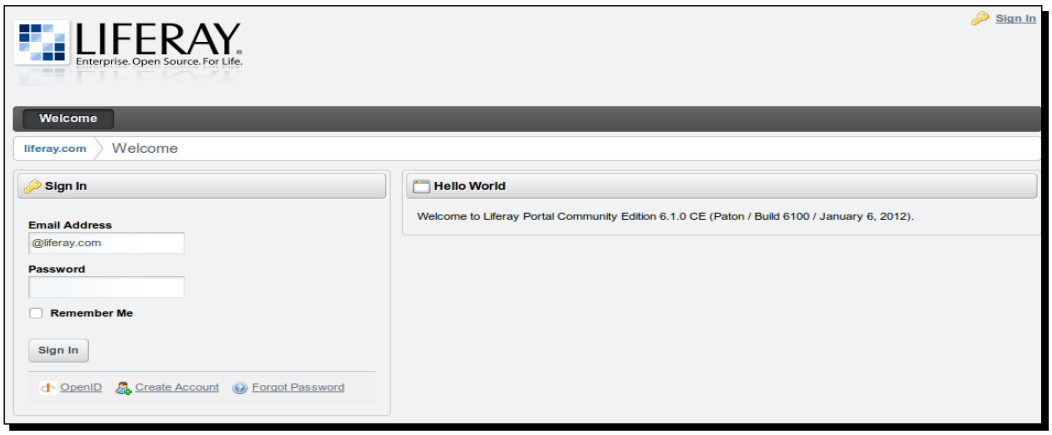
bootstrap	11/20/2011 7:38 PM	Executable Jar File	28 KB
catalina	11/20/2011 7:38 PM	Windows Batch File	13 KB
catalina.sh	11/20/2011 7:38 PM	SH File	19 KB
catalina-tasks	11/20/2011 7:38 PM	XML Document	3 KB
commons-daemon	11/20/2011 7:38 PM	Executable Jar File	24 KB
commons-daemon-native.tar	11/20/2011 7:38 PM	WinRAR archive	195 KB
configtest	11/20/2011 7:38 PM	Windows Batch File	3 KB
configtest.sh	11/20/2011 7:38 PM	SH File	2 KB
cpappend	11/20/2011 7:38 PM	Windows Batch File	2 KB
daemon.sh	11/20/2011 7:38 PM	SH File	8 KB
digest	11/20/2011 7:38 PM	Windows Batch File	3 KB
digest.sh	11/20/2011 7:38 PM	SH File	2 KB
setclasspath	11/20/2011 7:38 PM	Windows Batch File	4 KB
setclasspath.sh	11/20/2011 7:38 PM	SH File	4 KB
setenv	1/6/2012 3:32 PM	Windows Batch File	1 KB
setenv.sh	1/6/2012 3:32 PM	SH File	1 KB
shutdown	11/20/2011 7:38 PM	Windows Batch File	3 KB
shutdown.sh	11/20/2011 7:38 PM	SH File	2 KB
startup	11/20/2011 7:38 PM	Windows Batch File	3 KB
startup.sh	11/20/2011 7:38 PM	SH File	2 KB
tomcat-juli	11/20/2011 7:38 PM	Executable Jar File	38 KB
tomcat-native.tar	11/20/2011 7:38 PM	WinRAR archive	244 KB
tool-wrapper	11/20/2011 7:38 PM	Windows Batch File	5 KB
tool-wrapper.sh	11/20/2011 7:38 PM	SH File	5 KB
version	11/20/2011 7:38 PM	Windows Batch File	3 KB
version.sh	11/20/2011 7:38 PM	SH File	2 KB

If you are executing Liferay in a Linux system, you need to execute the `catalina.sh` file in your command prompt.

5. In the Linux command prompt, browse to the `bin` folder of `lifeary-portal-6.1-ce-ga3 ../liferay-portal-6.1.0-ce-ga3/tomcat-7.0.23/bin` and type the `sh.catalina.sh run` command, as shown in the following screenshot:

```
irshad@irshad:~/Desktop/liferay-portal-6.1.0-ce-ga3/tomcat-7.0.23/bin$ sh catalina.sh run
Using CATALINA_BASE:   /home/irshad/Desktop/liferay-portal-6.1.0-ce-ga3/tomcat-7.0.23
Using CATALINA_HOME:   /home/irshad/Desktop/liferay-portal-6.1.0-ce-ga3/tomcat-7.0.23
Using CATALINA_TMPDIR: /home/irshad/Desktop/liferay-portal-6.1.0-ce-ga3/tomcat-7.0.23/temp
Using JRE_HOME:        /usr/lib/jvm/java-7-openjdk-i386
Using CLASSPATH:       /home/irshad/Desktop/liferay-portal-6.1.0-ce-ga3/tomcat-7.0.23/bin/
0-ce-ga3/tomcat-7.0.23/bin/tomcat-juli.jar
```

6. To check whether Liferay has started, open a web browser, and type the URL, `localhost:8080`, which will display the home page of Liferay, as you can see in the following screenshot:

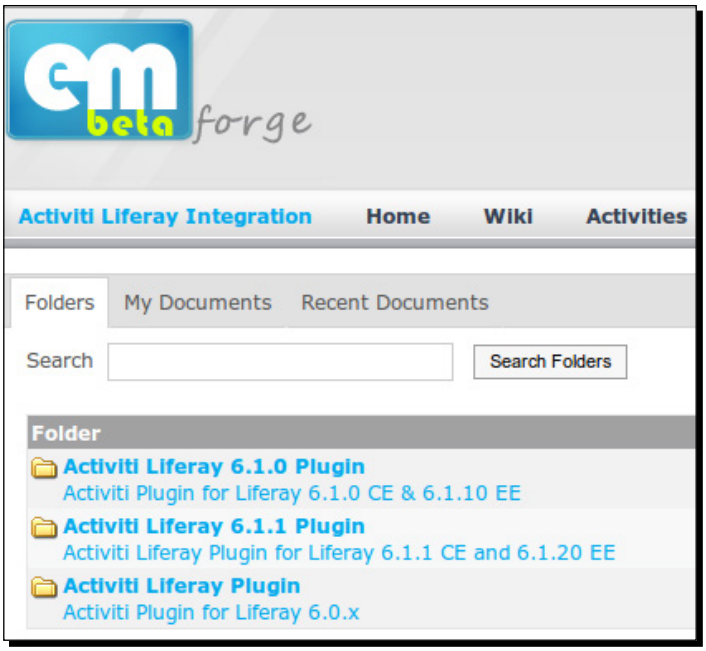


7. Log in with the following credentials:

   Email Address: `test@liferay.com`

   Password: `test`

8. To integrate Liferay with Activiti, we need to get an Activiti web application WAR file, which is provided by `emforge.net` at `http://www.emforge.net/web/activiti-liferay/files`, as shown in the following screenshot:



9. As we are using Liferay Version 6.1.0, we need to download the Activiti Liferay 6.1.0 plugin.

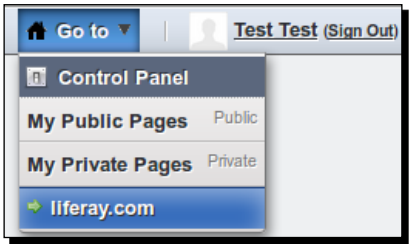
10. On downloading the Activiti plugin, you will get the `activiti-web-6.1.0.11.war` file in your `Downloads` folder, as you can see in the following icon:

11. Now, we will deploy the `activiti-web-6.1.0.11.war` file on the Liferay Portal. As we have Liferay Portal already running, we will log in with the following credentials:

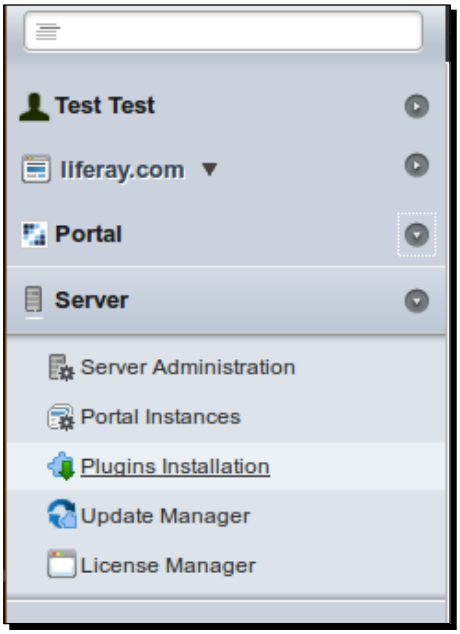
Email Address: `test@liferay.com`

Password: `test`

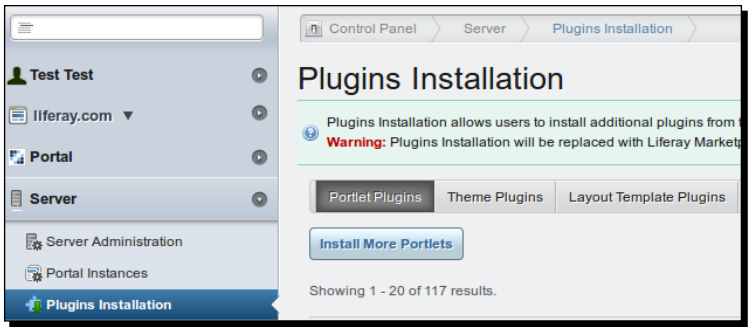
12. After logging into the Liferay Portal, we need to browse to the **Control Panel** tab from the **Go to** menu available at the top-right corner of the Liferay Portal; that is, navigate to **Go to** | **Control Panel**, as shown in the following screenshot:



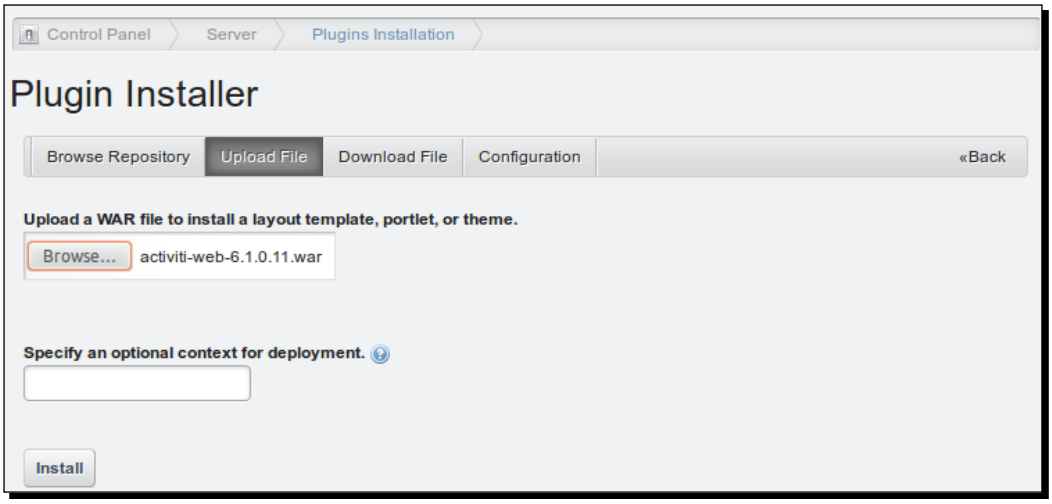
13. Selecting the **Control Panel** tab will redirect you to a page where we need to select the **Plugins Installation** option from the **Server** tab, as shown in the following screenshot:



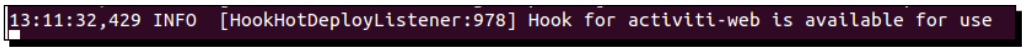
14. When we select the **Plugins Installation** option, it will open a page where we need to click on the **Install More Portlets** button (as shown in the following screenshot) to install the `activiti-web-6.1.0.11.war` file:



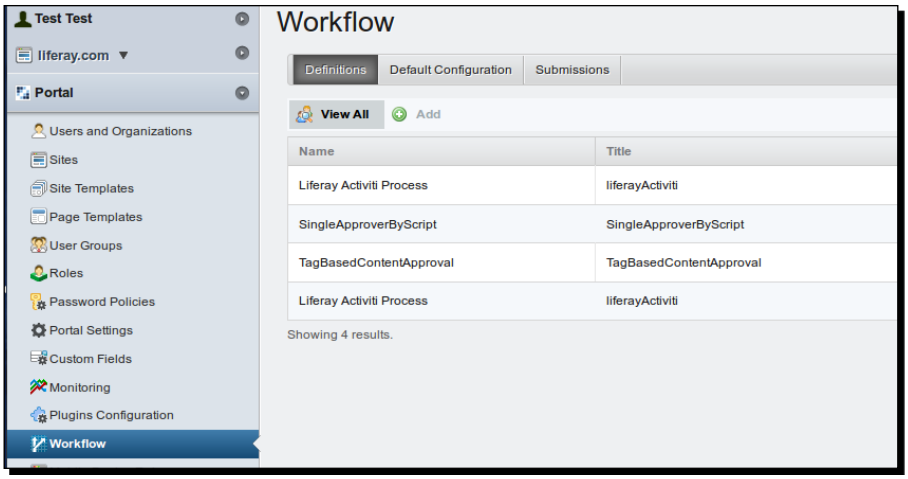
15. The **Install More Portlets** option will redirect you to a page that has various options. From those options, we need to select **Upload File**; in **Upload a WAR file...**, we need to select the `activiti-web-6.1.0.11.war` file and click on the **Install** button, as shown in the following screenshot:



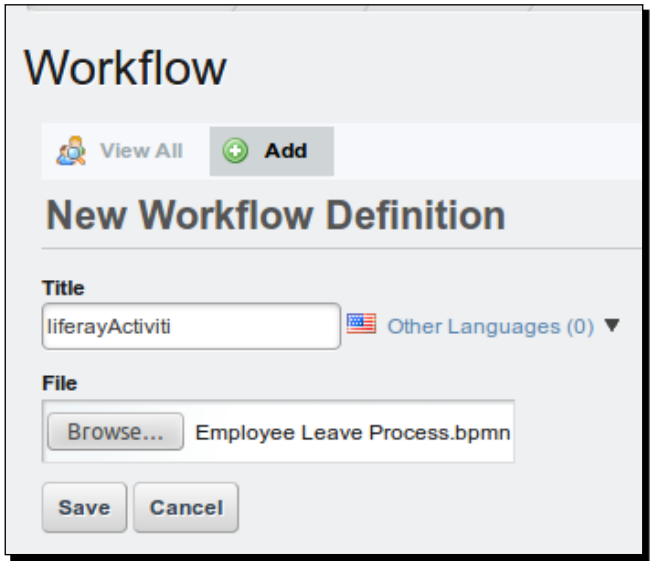
16. On successful installation of the WAR file, a message regarding the successful installation of the file will be displayed on the console, as shown in the following screenshot:



17. Once the `.war` file is deployed successfully, you can view some of the default workflow files (as shown in the following screenshot) by navigating to **Go to** | **Control Panel** | **Portal** | **Workflow**:



18. To upload a new workflow, you can select the **Add** option, provide a name for your business process, and upload a process file to Liferay, as shown in the following screenshot:



19. On successfully uploading the process file, you can use it with any of the resources available for workflow implementation.

What just happened?

In this section, we went through the integration of Activiti with the Liferay Portal. We used a web plugin to configure Activiti with Liferay and uploaded a business process file into the Liferay Portal.

Integrating business rules in Activiti

Business rule is a vast topic and there are various books available that discuss it. In this section, we will give you a brief introduction to business rules and how to use and implement them.

We can think of a business rule as a plain set of words that say something meaningful about the business. For example: customers making a payment with a platinum card should be given a 20 percent discount on the purchase of products. This is a business rule, but it has been explained in simple words that a lay person can also understand.

To create and implement business rules, we require a **business rule management system** (**BRMS**) and Drools is an open source BRMS. To know more about Drools, you can refer to *Drools JBoss Rules 5.0 Developer's Guide,* by *Packt Publishing*. In this chapter, we will cover just a brief overview of Drools. It is a JBoss project with the Apache license. The project consists of various modules, such as Drools Expert, Drools Guvnor, Drools Fusion, and Drools Planner.

Drools Expert

Drools Expert is a JSR-94-standard rule engine that is responsible for defining business rules and executing them. The rules are written using a **domain specific language** (**DSL**).

Drools Guvnor is a web-based user interface which we can use to manage business rules. It also supports the testing and authoring of business rules. It is considered to be a centralized repository for storing Drools rules.

Drools Fusion

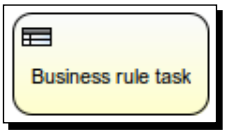
The Drools Fusion module is used for handling **complex event processing** (**CEP**). The action taken on a set of data is based on the pattern matched on analyses of events that have occurred.

Drools Planner

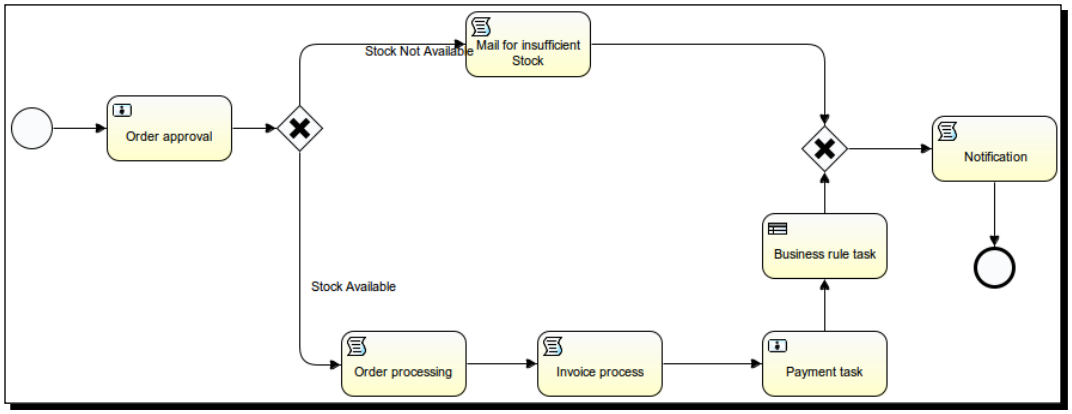
The Drools Planner module is responsible for optimizing automated planning solutions.

Time for action – integrating and implementing business rules with Activiti

In this section, we will focus on the Drools Expert module. We can work with the Drools Expert module without configuring the rest of the modules. Activiti provides a **business rule task** to implement business rules in our process, as shown in the following screenshot:

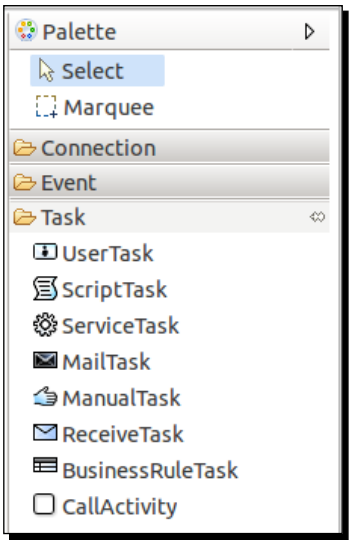


In this chapter, we will modify the laptop ordering process that we created in the *Time for action – designing your first process* section of *Chapter 3*, *Designing Your Process Using the Activiti Designer*. We will add a business rule task after the payment task in our process, as you can view in the following screenshot:

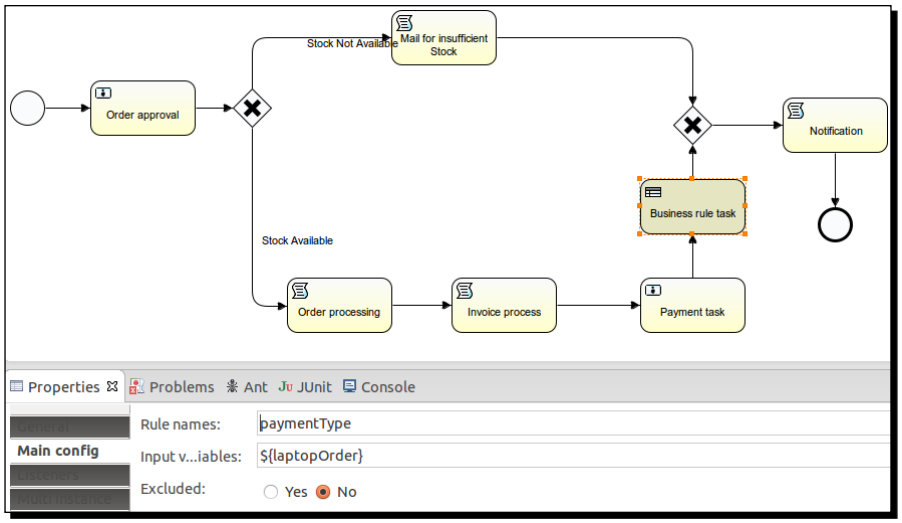


We will perform the following steps to proceed with the implementation:

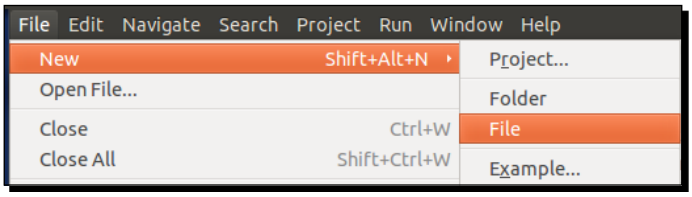
1. We will open our existing **Laptop Human Process** tab and add **BusinessRuleTask** from the **Palatte** | **Task** tab, as shown in the following screenshot:



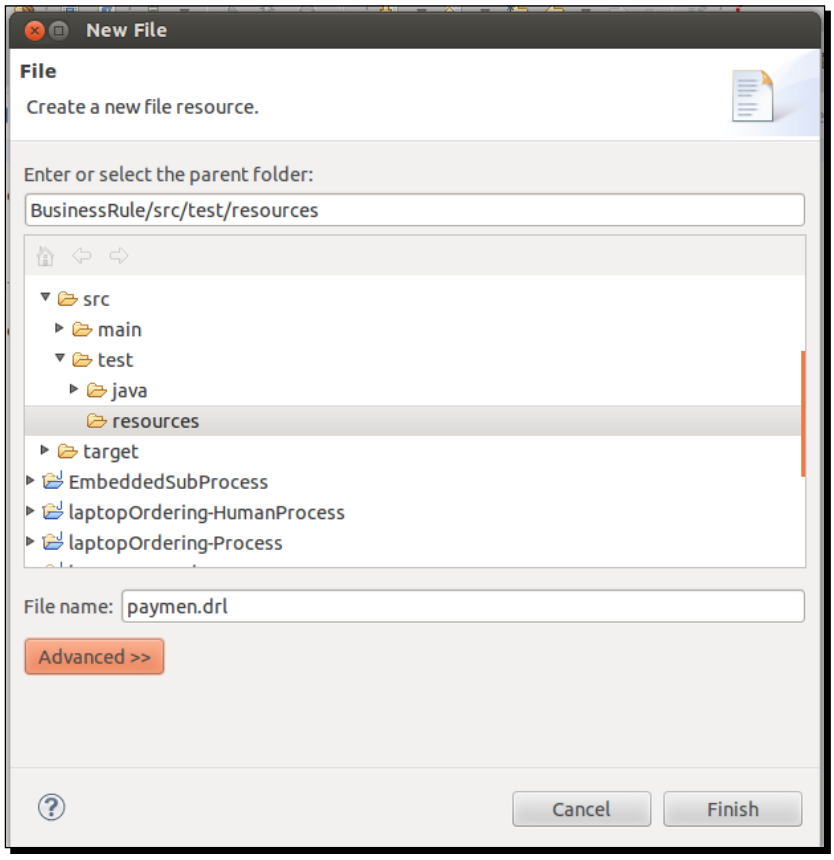
2. On adding the **BusinessRuleTask**, we need to set its properties. Select **BusinessRuleTask**, open the **Properties** window, and select the **Main config** tab. We need to populate the **Rule names** and **Input variables** properties, as shown in the following screenshot:



3. In the **Rule names** field, we need to provide the name of the rule that we want to call from **BusinessRuleTask**.
4. Now, we will create a rule file with the `.drl` extension in which we will have our business logic written.
5. We can create a new file in Eclipse by navigating to **File** | **New** | **File**, as shown in the following screenshot:



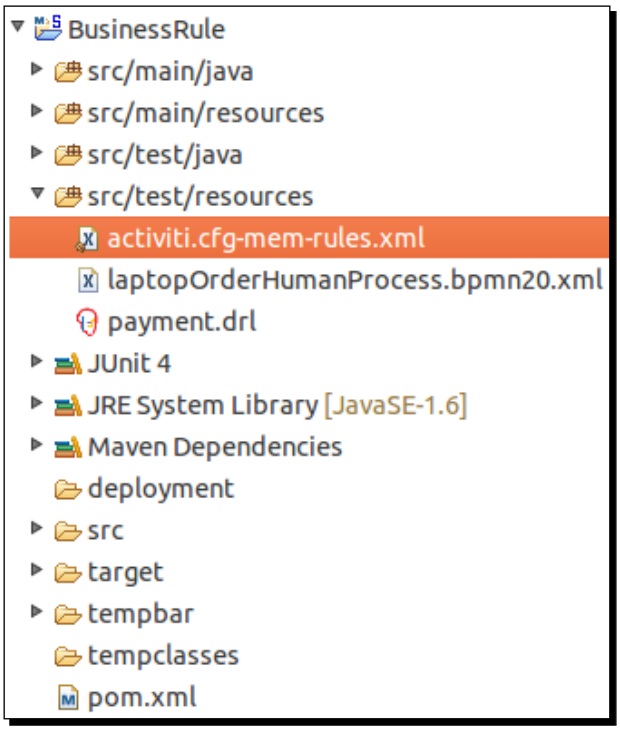
6. The **File** option opens a pop-up window where we will create a file named `payment.drl` on a specific path, as shown in the following screenshot:



7. In the `payment.drl` file, we will write our business logic. The logic here is as follows: if the payment is made by credit card, provide a 5 percent discount on the price of the laptop. The following lines of code show the logic of this:

```
package org.activiti.book;
import com.book.activiti.LaptopOrder;

rule "paymentType"
when
  lapOrder : LaptopOrder(paymentMode == "creditCard")
then
  lapOrder.setAmtToBePaid
    ((lapOrder.getLaptopAmount()*
    lapOrder.getLaptopQuantity())-
    ((lapOrder.getLaptopAmount()*
    lapOrder.getLaptopQuantity())*5/100));
end
```

8. We need to create an `activiti.cfg-mem-rules.xml` file in the `src/test/resources` folder of our application to enable the rule engine, as shown in the following screenshot:



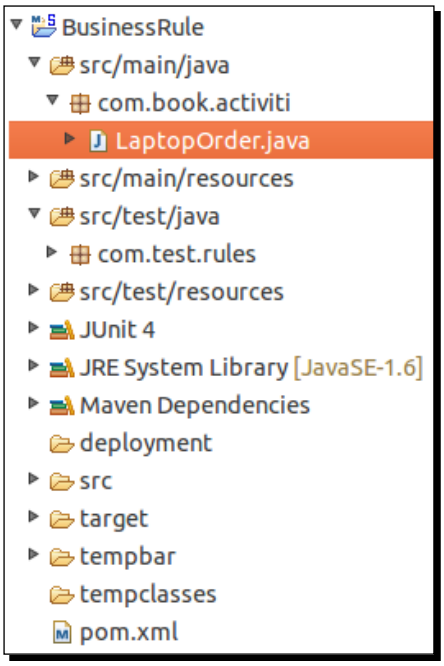
9. In the `activiti.cfg-mem-rules.xml` file, we need to provide an `org.activiti.engine.impl.rules.RulesDeployer` class, as shown in the following code snippet; this class is responsible for executing our business rule:

```
<bean id="processEngineConfiguration"
    class="org.activiti.engine.impl.cfg.
StandaloneInMemProcessEngineConfiguration">

    <property name="databaseSchemaUpdate" value="true" />
    <property name="customPostDeployers">
      <list>
        <bean
          class="org.activiti.engine.impl.rules.RulesDeployer"
           />
      </list>
    </property>
  </bean>
```

10. After adding the tags shown in the preceding code within the XML file, we will get the following output:

```
<?xml version="1.0" encoding="UTF-8"?>

<beans xmlns="http://www.springframework.org/schema/beans"
    xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
    xsi:schemaLocation="http://www.springframework.org/schema/beans   http://www.springframework.org

    <bean id="processEngineConfiguration"
        class="org.activiti.engine.impl.cfg.StandaloneInMemProcessEngineConfiguration">

        <property name="databaseSchemaUpdate" value="true" />
        <property name="customPostDeployers">
            <list>
                <bean class="org.activiti.engine.impl.rules.RulesDeployer" />
            </list>
        </property>
    </bean>
</beans>
```

11. Now, we will create a `LaptopOrder` class file, as shown in the following screenshot, that will contain the getter and setter methods of our variables:



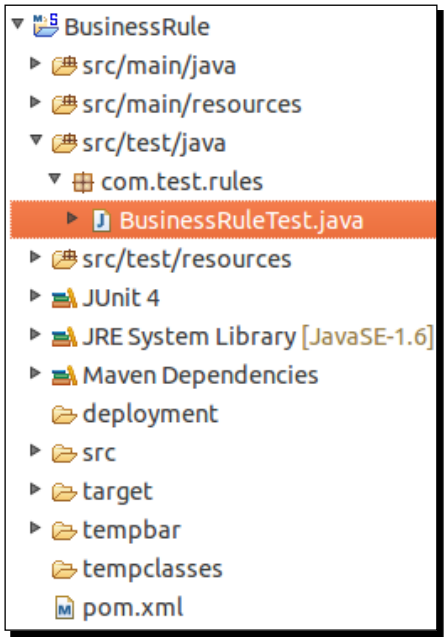
12. The code in the `LaptopOrder` class will contain information regarding the customer and the laptop details, as follows:

```
public class LaptopOrder implements Serializable {

  private static final long serialVersionUID = 1L;

  private String customerName;
  private String laptopName;
  private long laptopQuantity;
  private long laptopModelNo;
  private String paymentMode;
  private long amtToBePaid;
  private long laptopAmount;
  public long getLaptopAmount() {
    return laptopAmount;
  }
```

```
public void setLaptopAmount(long laptopAmount) {
  this.laptopAmount = laptopAmount;
}

public long getAmtToBePaid() {
  return amtToBePaid;
}

public void setAmtToBePaid(long amtToBePaid) {
  this.amtToBePaid = amtToBePaid;
}

public String getCustomerName() {
  return customerName;
}

public void setCustomerName(String customerName) {
  this.customerName = customerName;
}

public String getLaptopName() {
  return laptopName;
}

public void setLaptopName(String laptopName) {
  this.laptopName = laptopName;
}

public long getLaptopQuantity() {
  return laptopQuantity;
}

public void setLaptopQuantity(long laptopQuantity) {
  this.laptopQuantity = laptopQuantity;
}

public long getLaptopModelNo() {
  return laptopModelNo;
}

public void setLaptopModelNo(long laptopModelNo) {
  this.laptopModelNo = laptopModelNo;
}
```

```
  public String getPaymentMode() {
    return paymentMode;
  }

  public void setPaymentMode(String paymentMode) {
    this.paymentMode = paymentMode;
  }
}
```

13. To execute the business process, we will create a `BusinessRuleTest` class file, as shown in the following screenshot, using which we will test our business process:



14. The code for testing our business process is as follows:

```
public class BusinessRuleTest {

  @Rule
  public ActivitiRule activitiRule = new ActivitiRule(
      "activiti.cfg-mem-rules.xml");

  @Test
  @Deployment(resources = { "laptopOrderHumanProcess.bpmn20.xml",
      "payment.drl" })
  public void testLaptopProcess() {
```

```
    Map<String, Object> variableMap = new HashMap<String,
Object>();
    Map<String, Object> completeOrder = new HashMap<String,
Object>();

    LaptopOrder laptopOrder = new LaptopOrder();
    laptopOrder.setCustomerName("Irshad");
    laptopOrder.setLaptopModelNo(3420);
    laptopOrder.setLaptopName("Dell");
    laptopOrder.setLaptopQuantity(2);
    laptopOrder.setPaymentMode("creditCard");
    laptopOrder.setLaptopAmount(30000);

    variableMap.put("laptopOrder", laptopOrder);

    ProcessInstance processInstance = activitiRule.
getRuntimeService()
      .startProcessInstanceByKey("laptopHumanProcess",
variableMap);
    assertNotNull(processInstance);

    TaskService taskService = activitiRule.getTaskService();

    Task task = taskService.createTaskQuery().singleResult();
    completeOrder.put("acceptOrder", "true");
    taskService.complete(task.getId(), completeOrder);

    task = taskService.createTaskQuery().singleResult();
    taskService.complete(task.getId(), variableMap);

  }
}
```

15. Now, we can perform a test on our business process by executing the `BusinessRuleTest.java` file. We can run the class file as a JUnit test.

16. On successful execution of the test file, we should see the output on the console, which should look similar to that shown in the following screenshot:

```
25 Nov, 2013 11:50:03 AM org.springframework.beans.factory.xml.XmlBeanDefinitionReader loadBeanDefinitions
INFO: Loading XML bean definitions from class path resource [activiti.cfg-mem-rules.xml]
25 Nov, 2013 11:50:05 AM org.activiti.engine.impl.db.DbSqlSession executeSchemaResource
INFO: performing create on engine with resource org/activiti/db/create/activiti.h2.create.engine.sql
25 Nov, 2013 11:50:05 AM org.activiti.engine.impl.db.DbSqlSession executeSchemaResource
INFO: performing create on history with resource org/activiti/db/create/activiti.h2.create.history.sql
25 Nov, 2013 11:50:05 AM org.activiti.engine.impl.db.DbSqlSession executeSchemaResource
INFO: performing create on identity with resource org/activiti/db/create/activiti.h2.create.identity.sql
25 Nov, 2013 11:50:05 AM org.activiti.engine.impl.ProcessEngineImpl <init>
INFO: ProcessEngine default created
25 Nov, 2013 11:50:05 AM org.activiti.engine.impl.bpmn.deployer.BpmnDeployer deploy
INFO: Processing resource payment.drl
25 Nov, 2013 11:50:05 AM org.activiti.engine.impl.bpmn.deployer.BpmnDeployer deploy
INFO: Processing resource laptopOrderHumanProcess.bpmn20.xml
25 Nov, 2013 11:50:06 AM org.activiti.engine.impl.bpmn.parser.BpmnParse parseDefinitionsAttributes
INFO: XMLSchema currently not supported as typeLanguage
25 Nov, 2013 11:50:06 AM org.activiti.engine.impl.bpmn.parser.BpmnParse parseDefinitionsAttributes
INFO: XPath currently not supported as expressionLanguage
25 Nov, 2013 11:50:08 AM org.activiti.engine.impl.rules.RulesDeployer deploy
INFO: Processing resource payment.drl
25 Nov, 2013 11:50:10 AM org.activiti.engine.impl.rules.RulesDeployer deploy
INFO: Processing resource laptopOrderHumanProcess.bpmn20.xml

 === Order Accepted and forwarded for processing ===

 === Generate Invoice ===

 === Notification  for the order request Amount to be Paid === 57000
```

What just happened?

In this section, we configured Activiti to use Drools rule engine in which we wrote our business logic.

Deploying Activiti as an OSGi bundle

Before starting with the implementation, we will get a brief introduction to OSGi. The following definition is given on Wikipedia:

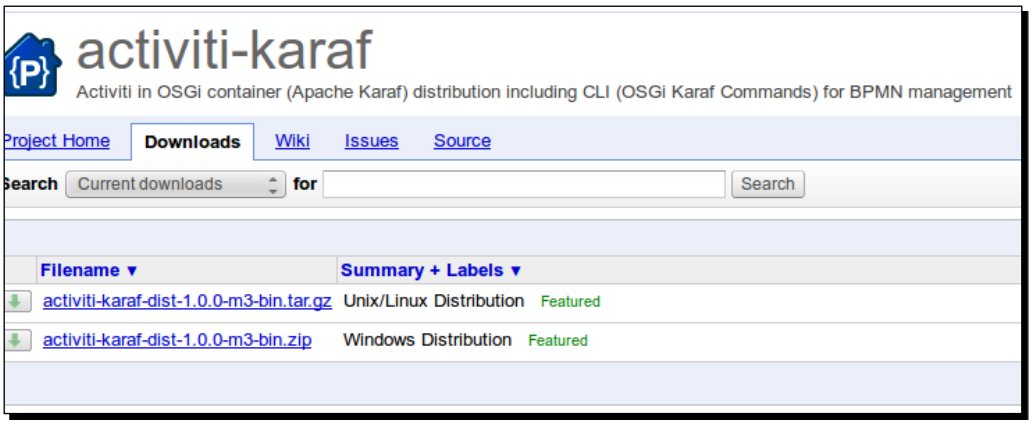
> *"The OSGi framework is a module system and service platform for Java programming language that implements a complete and dynamic component model, something that does not exist in standalone Java/VM environments. Applications or components (coming in the form of bundles for deployment) can be remotely installed, started, stopped, updated, and uninstalled without requiring a reboot; management of Java packages/classes is specified in great detail. Application life cycle management (start, stop, install, etc.) is done via APIs that allow for remote downloading of management policies. The service registry allows bundles to detect the addition of new services, or the removal of services, and adapt accordingly."*

For more information, you can refer to *Instant OSGi Starter*, *Packt Publishing*.

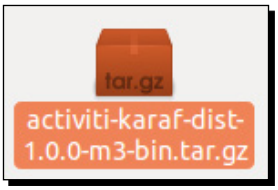
Time for action – integrating Activiti with Apache Karaf

Apache Karaf is a small OSGi bundle with a lightweight container and is capable of deploying various applications and components. To deploy Activiti as an OSGi, we have to carry out the following steps:

1. To start the integration of Activiti with Karaf, there are some prerequisite installations to be done on our system. JDK should be installed with Version 1.5 or higher and Maven 3.0.3 should be configured in your system.
2. We need to download Activiti Karaf from `http://code.google.com/p/activiti-karaf/downloads/list`. This link contains files for both Linux and Windows operating systems, as shown in the following screenshot:



3. You can download the files corresponding to the operating system you are using. As we are working with a Linux system, we will download the Linux distribution file.



4. Now, we will extract the file from a specific path. To use Activiti with Karaf, we need to do some configuration. You can browse to the `../activity-karaf-dist-1.0.0-SNAPSHOT/sources/activity-bpmn-archtype` folder; there will be only an `src` folder and a `pom.xml` file, as shown in the following screenshot:

5. As this archetype is currently not available in the public Maven repository, we need to install it in the public repository by executing the `mvn install` command from the command prompt, as shown in the following screenshot:

```
irshad@irshad:~/Desktop/activiti-karaf-dist-1.0.0-SNAPSHOT/sources/activiti-bpmn-archetype$ mvn install
[INFO] Scanning for projects...
```

6. The `mvn install` command will start installing the `activiti-bpmn-archetype` in the local repository. The installation process will take some time. Once the installation is completed, you should get a **BUILD SUCCESS** message as shown in the following screenshot:

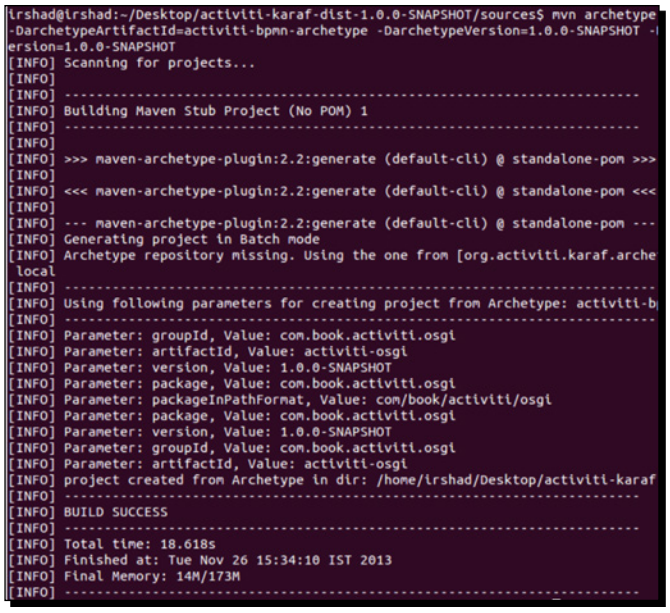
```
[INFO]
[INFO] --- maven-archetype-plugin:2.2:update-local-catalog (default-update-local-catalog) @ activiti-bpmn-archetype ---
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time: 4:09.300s
[INFO] Finished at: Tue Nov 26 14:57:16 IST 2013
[INFO] Final Memory: 12M/100M
[INFO] ------------------------------------------------------------------------
```

7. After the successful installation of `activiti-bpmn-archetype`, we can create a `bpmn` project using `activiti-bpmn-archtype`.

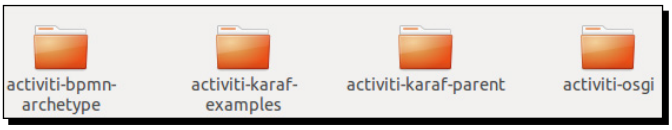
8. To create the project, browse to the `sources` folder of your `activiti-karaf-dist-1.0.0-SNAPSHOT` path and type the following command:

```
mvn archetype:generate -B -DarchetypeGroupId=org.activiti.
karaf.archetypes -DarchetypeArtifactId=activiti-bpmn-archetype
-DarchetypeVersion=1.0.0-SNAPSHOT -DgroupId=com.book.activiti.OSGi
-DartifactId=activiti-OSGi -Dversion=1.0.0-SNAPSHOT
```

9. On the successful execution of the preceding command, you will get a **BUILD SUCCESSFULL** message on the console, as shown in the following screenshot, and a project will be created in your `sources` folder:



10. Our project `activiti-OSGi` will be created in the `sources` folder of `activiti-karaf-dist-1.0.0-SNAPSHOT`. You can browse to `../activiti-karaf-dist-1.0.0-SNAPSHOT/sources` and you should have folders mentioned in the following screenshot:



11. Now, we need to modify the `pom.xml` file of our `activiti-OSGi` project. The `pom.xml` file is located at the `../activiti-karaf-dist-1.0.0-SNAPSHOT/sources/activiti-OSGi` path.

12. We need to make changes in the `<parent></parent>` element of the `pom.xml` file with the given code snippet:

```
<parent>

    <groupId>org.activiti.karaf</groupId>

    <artifactId>activiti-karaf-parent</artifactId>
```

```
    <version>1.0.0-SNAPSHOT</version>

    <relativePath>../activiti-karaf-parent</relativePath>

  </parent>
```

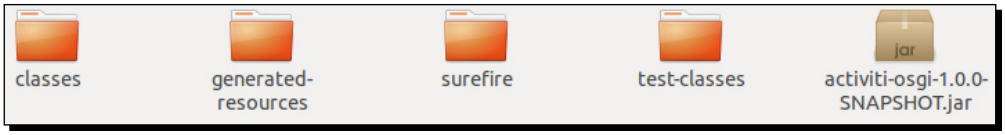
13. By providing the previous `groupId` and `artifactId`, our project becomes an OSGi Activiti project and we can run it on the OSGi container.

14. In your `../activiti-karaf-dist-1.0.0-SNAPSHOT/sources/activiti-OSGi/src/main/process` path, you will find a `bpmn` file named `activiti-OSGi.bpmn20.xml`, which is created in the project by default.

15. Now in order to execute our project, we need to create a package structure. To create a package structure, we have to type the `mvn clean package` command in our project folder path as mentioned in the following screenshot:

```
/Desktop/activiti-karaf-dist-1.0.0-SNAPSHOT/sources/activiti-osgi$ mvn clean package
```

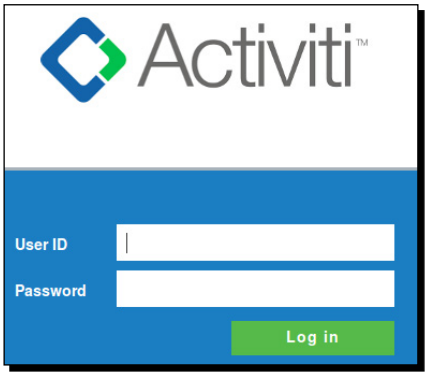
16. On the successful execution of the previous command, a target folder is created within `../activiti-karaf-dist-1.0.0-SNAPSHOT/sources/activiti-OSGi`, which contains the `activiti-OSGi-1.0.0-SNAPSHOT.jar` file and various classes, as shown in the following screenshot:



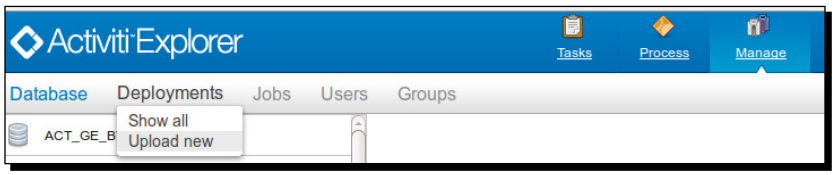
17. Now, we will deploy our `activiti-OSGi` project on Activiti Karaf. To start `activiti-karaf`, we have to execute the `./karaf` command from the command prompt by browsing to the `../activiti-karaf-dist-1.0.0-SNAPSHOT/bin` path as viewed in the following screenshot:

```
irshad@irshad:~/Desktop/activiti-karaf-dist-1.0.0-SNAPSHOT/bin$ ./karaf
                  _ _       _ _        _
        /\       | (_)     (_) |  (_)
       /  \   ___| |_ ___   ___| |_ _
      / /\ \ / __| __| \ \ / / | __| |
     / ____ \ (__| |_| |\ V /| | |_| |
    /_/    \_\___|\__|_| \_/ |_|\__|_|

  Activiti Karaf (1.0.0-SNAPSHOT)

Hit '<tab>' for a list of available commands
and '[cmd] --help' for help on a specific command.
Hit '<ctrl-d>' or 'osgi:shutdown' to shutdown Activiti Karaf.

karaf@root>
```

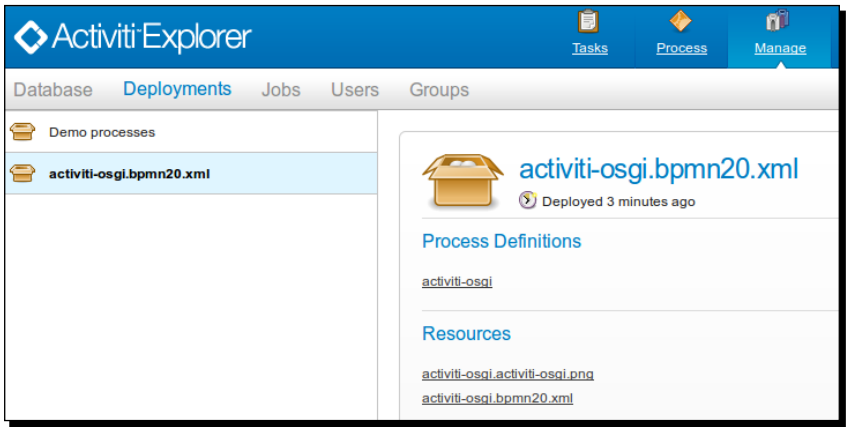
18. On starting Activiti Karaf, we have started the Activiti OSGi engine.
19. Now, open a web browser and type `http://localhost:8181/activiti-explorer`. This URL will open the Activiti engine and you will get the login page, as shown in the following screenshot:



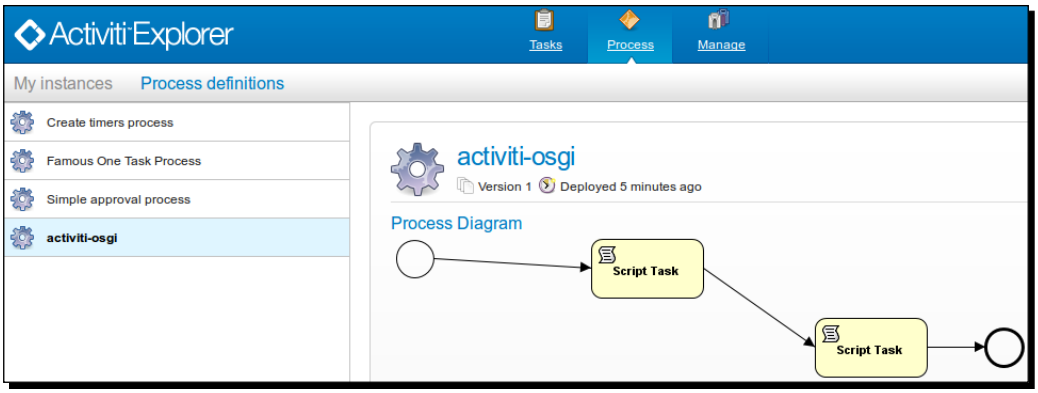
20. You can log in with the user ID `kermit` and password `kermit`. Then, browse to the **Manage** | **Deployments** | **Upload new** link and upload the `activiti-OSGi.bpmn20.xml` file, as shown in the following screenshot:



21. When the file is uploaded successfully, it will be available in **Manage** | **Deployments**, as shown in the following screenshot:



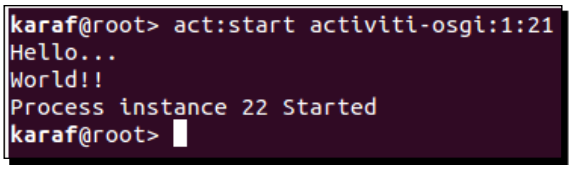
22. To view the process diagram, you can browse to **Process** | **Processs definitions** | **activiti-OSGi** where you will see the graphical representation of your business process, as shown in the following screenshot:



23. As we have deployed the process on the Activiti engine, we will now go back to the command prompt and check the list of processes deployed by executing the `act:list` command on the command prompt; we should be able to view our `activiti-OSGi` process in the list, as shown in the following screenshot:

```
karaf@root> act:list

BPMN Deployments
----------------
 ID   Name                      Deployment Time
[10 ][Demo processes          ][26 Nov, 2013 5:17:26 PM]
[18 ][activiti-osgi.bpmn20.xml][26 Nov, 2013 5:34:04 PM]

BPMN Process Definitions
------------------------
 Definition ID              Name                    Ver  Resource
[createTimersProcess:1:15  ][Create timers process  ][1  ][org/activiti/explorer/demo/process/createTimersProcess.bpmn20.xml]
[simpleApprovalProcess:1:16][Simple approval process][1  ][org/activiti/explorer/demo/process/testProcess.bpmn20.xml        ]
[oneTaskProcess:1:17       ][Famous One Task Process][1  ][org/activiti/explorer/demo/process/oneTaskProcess.bpmn20.xml     ]
[activiti-osgi:1:21        ][activiti-osgi          ][1  ][activiti-osgi.bpmn20.xml                                         ]

History of BPMN Process Instances
---------------------------------
No History on BPMN Processes.

Active BPMN Process Instances
-----------------------------
No Active BPMN Process Instances Found.
```

24. We can also start the process from the OSGi command prompt by executing the `act:start [process definition id]` command, as shown in the following screenshot:



What just happened?

In this section, we used an OSGi container called Apache Karaf to implement our Activiti business process.

Summary

In this chapter, we have come across configuring the Liferay Portal to use the Activiti `.bpmn` files, using a Drools rule engine within an Activiti business process, and using an OSGi container called Apache Karaf to implement an Activiti business process. In the next chapter, we will see how to implement advanced workflows using a Sub-Process and parallel gateways, and monitoring using BAM.

9
Implementing Advanced Workflows

In the previous chapter, we went through the integration of Activiti with various third-party services such as portals, CMS, and OSGi. We were able to use the third-party service for the implementation of our business workflows. Now we can make use of some advanced notations for implementing advanced business workflows.

This chapter covers the following topics:

- Understanding parallel gateways
- Getting started with Sub-Processes
- Understanding multi-instance processes
- Working with execution and task listeners
- Monitoring workflows using BAM and CEP
- Monitoring using Esper

We can design and implement some advanced and complex workflows in Activiti with the help of Sub-Processes, parallel gateways, and multi-instance processes.

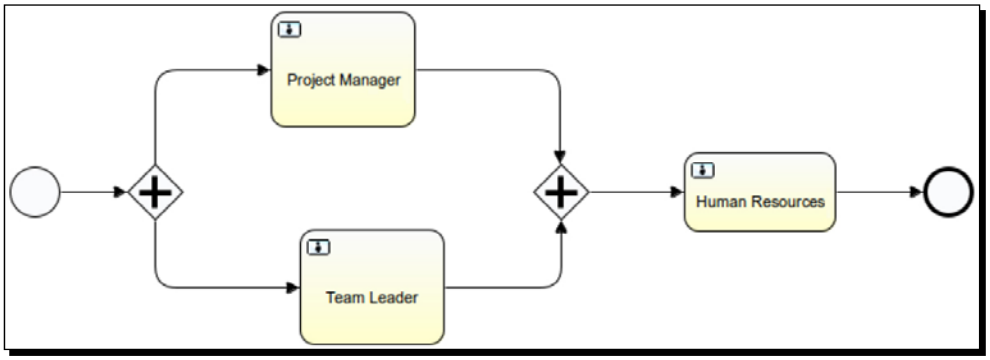
Understanding parallel gateways

We can make use of parallel gateways to implement concurrency in a business process. The difference between a parallel gateway and other gateways is that we do not require any condition to be satisfied for the execution of the flow. In a parallel gateway, all the outgoing flows are executed. A parallel gateway can be used for both the forking and joining of the business flow.

Time for action – implementing a parallel gateway

We will implement a process with a parallel gateway by performing the following steps:

1. We will use an example of a leave process. When an employee applies for leave, the request is sent to the project manager and team leader in parallel and then sent to Human Resources:



2. You can populate the form properties for accepting values from the employee on the start event, as shown in the following code:

```
<startEvent id="sid-2BB933AE-E0AE-48D2-9ACC-B5EC35AD3687">
      <extensionElements>
        <activiti:formProperty id="empName" name="Employee Name"
type="string" required="true"></activiti:formProperty>
        <activiti:formProperty id="empDesignation"
name="Employee Designation" type="string" required="true"></
activiti:formProperty>
        <activiti:formProperty id="empLeaveReason" name="Leave
Reason" type="string" required="true"></activiti:formProperty>
        <activiti:formProperty id="empLeaveDays" name="No of Days
Leave " type="long" required="true"></activiti:formProperty>
      </extensionElements>
    </startEvent>
```

3. Create a user task for the team leader and project manager, who will be able to view the details filled by the employee. Then, based on the details, they can either accept or reject the leave, as shown in the following code:

```
<userTask id="sid-03BC7128-4496-4027-88A9-E67D3DA63734"
name="Project Manager" activiti:assignee="gonzo">
      <extensionElements>
        <activiti:formProperty id="eName" name="Employee Name"
type="string" expression="${empName}" writable="false"></
activiti:formProperty>
```

```
        <activiti:formProperty id="eDesig" name="Employee
Designation" type="string" expression="${empDesignation}"
writable="false"></activiti:formProperty>
        <activiti:formProperty id="eLeaveReason" name="Leave
Reason" type="string" expression="${empLeaveReason}"
writable="false"></activiti:formProperty>
        <activiti:formProperty id="eLeaveDays" name="Leave Days"
type="long" expression="${empLeaveDays}" writable="false"></
activiti:formProperty>
        <activiti:formProperty id="pmDecision" name="Do You
Approve ?" type="enum" required="true">
          <activiti:value id="true" name="Yes"></activiti:value>
          <activiti:value id="false" name="No"></activiti:value>
        </activiti:formProperty>
        <activiti:formProperty id="pmComment" name="Your Comment"
type="string" required="true"></activiti:formProperty>
      </extensionElements>
    </userTask>
    <userTask id="sid-7581049C-894E-4FF9-B861-7DF44B7229E3"
name="Team Leader" activiti:assignee="fozzie">
      <extensionElements>
        <activiti:formProperty id="eName" name="Employee Name"
type="string" expression="${empName}" writable="false"></
activiti:formProperty>
        <activiti:formProperty id="eDesig" name="Employee
Designation" type="string" expression="${empDesignation}"
writable="false"></activiti:formProperty>
        <activiti:formProperty id="eLeaveReason" name="Leave
Reason" type="string" expression="${empLeaveReason}"
writable="false"></activiti:formProperty>
        <activiti:formProperty id="eLeaveDays" name="Leave Days"
type="long" expression="${empLeaveDays}" writable="false"></
activiti:formProperty>
        <activiti:formProperty id="tlDecision" name="Do You
Approve ?" type="enum" required="true">
          <activiti:value id="true" name="Yes"></activiti:value>
          <activiti:value id="false" name="No"></activiti:value>
        </activiti:formProperty>
        <activiti:formProperty id="tlComment" name="Your Comment"
type="string" required="true"></activiti:formProperty>
      </extensionElements>
    </userTask>
```

4. Now, create a parallel gateway to connect to the `Team Leader` and `Project Manager` tasks that are to be executed in parallel, as shown in the following code:

```
 <sequenceFlow id="flow1" sourceRef="sid-2BB933AE-E0AE-48D2-9ACC-
B5EC35AD3687" targetRef="parallelgateway1"></sequenceFlow>
    <parallelGateway id="parallelgateway1" name="Parallel
Gateway"></parallelGateway>
```

```
    <sequenceFlow id="flow2" sourceRef="parallelgateway1"
targetRef="sid-03BC7128-4496-4027-88A9-E67D3DA63734"></
sequenceFlow>
    <sequenceFlow id="flow3" sourceRef="parallelgateway1"
targetRef="sid-7581049C-894E-4FF9-B861-7DF44B7229E3"></
sequenceFlow>
```

5. Create a `Human Resource` user task where all the details from the employee, project manager, and team leader will be viewed, and based on the results of the `Project Manager` and `Team Leader` tasks, HR will either approve or reject the leave, as shown in the following code:

```
<userTask id="usertask1" name="Human Resource">
      <extensionElements>
        <activiti:formProperty id="eName" name="Employee Name"
type="string" expression="${empName}" writable="false"></
activiti:formProperty>
        <activiti:formProperty id="eDesig" name="Employee
Designation" type="string" expression="${empDesignation}"
writable="false"></activiti:formProperty>
        <activiti:formProperty id="eLeaveReason" name="Leave
Reason" type="string" expression="${empLeaveReason}"
writable="false"></activiti:formProperty>
        <activiti:formProperty id="eLeaveDays" name="Leave Days"
type="long" expression="${empLeaveDays}" writable="false"></
activiti:formProperty>
        <activiti:formProperty id="hrDecision" name="Do You
Approve ?" type="enum" required="true">
          <activiti:value id="true" name="Yes"></activiti:value>
          <activiti:value id="false" name="No"></activiti:value>
        </activiti:formProperty>
        <activiti:formProperty id="hrComment" name="Your Comment"
type="string" required="true"></activiti:formProperty>
        <activiti:formProperty id="tlComments" name="TL Comments"
type="string" expression="${tlComment}" writable="false"></
activiti:formProperty>
        <activiti:formProperty id="pmComments" name="Project
Manager Comments" type="string" expression="${pmComment}"
writable="false"></activiti:formProperty>
      </extensionElements>
    </userTask>
```

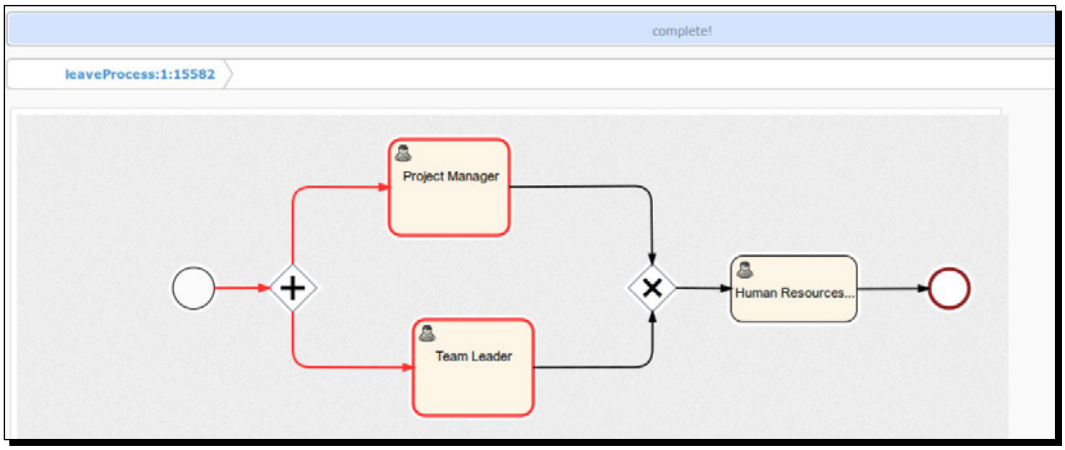
6. We need to join the parallel gateways so that we can get the appropriate results, as shown in the following code:

```
<sequenceFlow id="flow5" sourceRef="usertask1" targetRef="sid-
65043A85-6BAD-4616-AD1E-FF3FA8D64D4B"></sequenceFlow>
    <sequenceFlow id="flow6" sourceRef="sid-03BC7128-4496-4027-
88A9-E67D3DA63734" targetRef="parallelgateway2"></sequenceFlow>
    <sequenceFlow id="flow7" sourceRef="sid-7581049C-894E-4FF9-
```

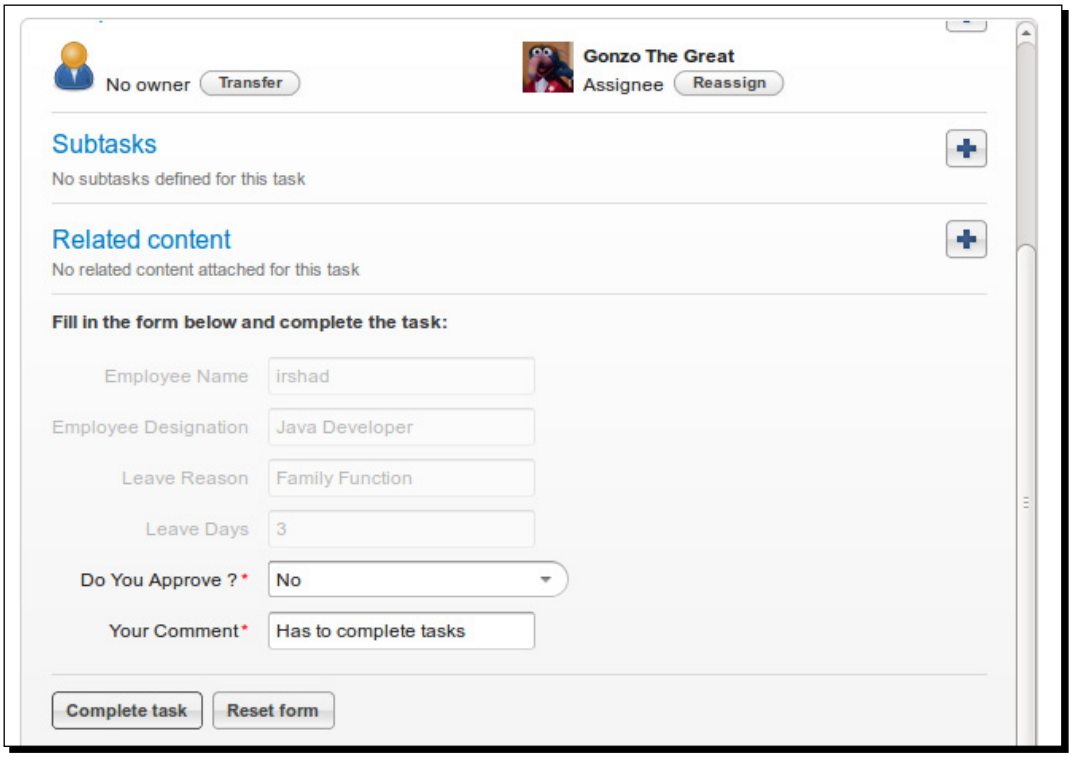
```
B861-7DF44B7229E3" targetRef="parallelgateway2"></sequenceFlow>
    <sequenceFlow id="flow8" sourceRef="parallelgateway2"
targetRef="usertask1"></sequenceFlow>
    <parallelGateway id="parallelgateway2" name="Exclusive
Gateway"></parallelGateway>
```

7. Now, to execute the process, deploy it into the Activiti Explorer, start it, and fill in the details for the leave, as shown in the following screenshot:

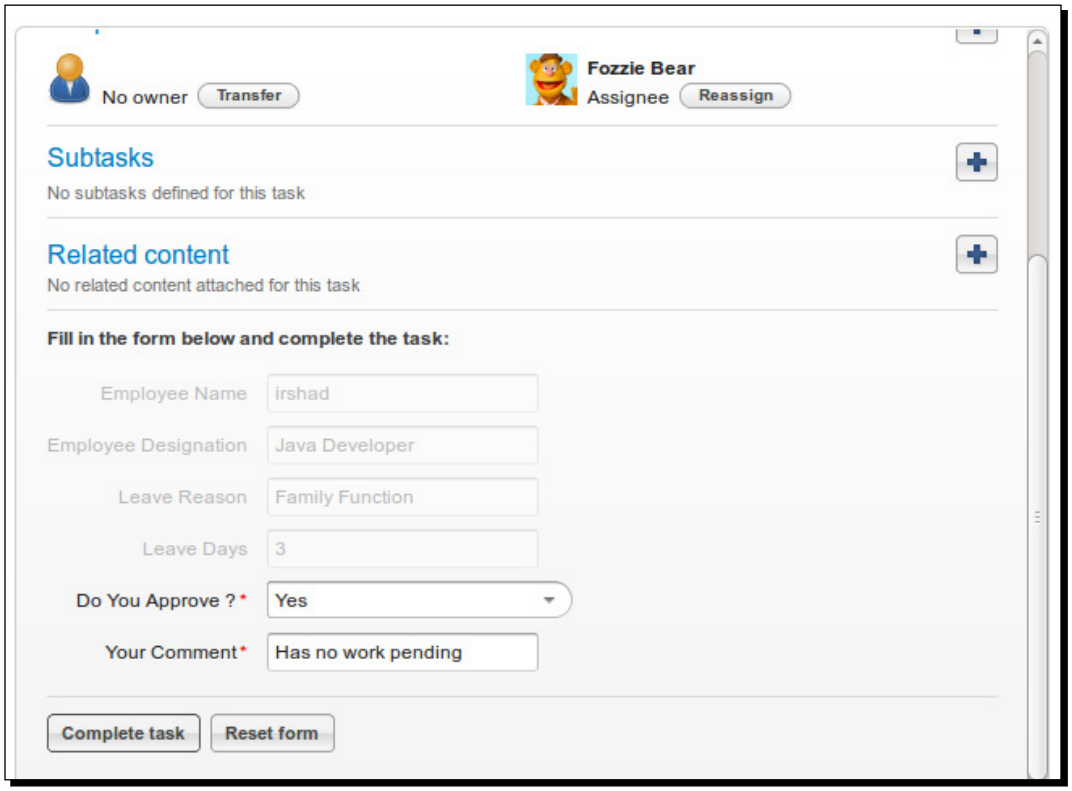
8. To view the state of your process, navigate to the **My Instances** tab. As seen in the following screenshot, the **Project Manager** and **Team Leader** process are in execution:



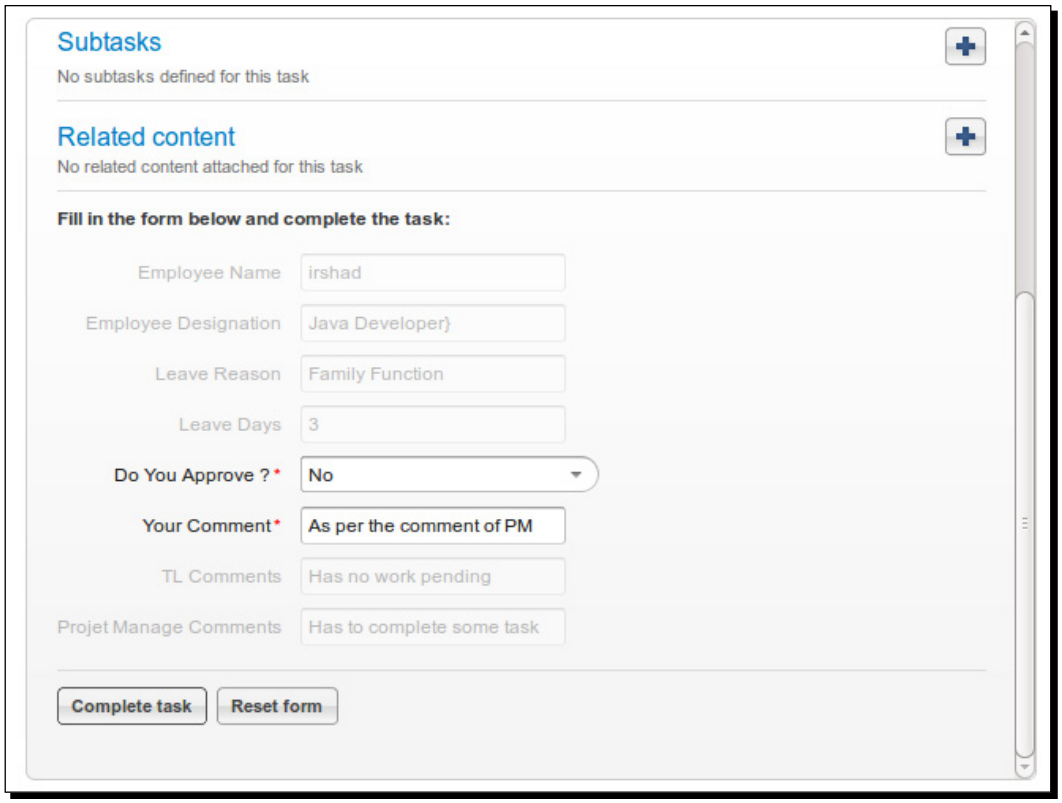
9. Log in with `gonzo` as the user, as we have assigned the `Project Manager` task to him. The values added by the employee will be visible to him; now, the PM can accept or reject the leave and provide comments as shown in the following screenshot:



10. Now log in with `fozzie`, as the `Team Leader` task is assigned to him. Here, too, the details added by the employee will be available, as you can view in the following screenshot:



11. Now log in with `kermit`, as he is assigned the `Human Resource` task. In this task, the details provided by the employee as well as the comments provided by the PM and TL will be available, as shown in the following screenshot:



What just happened?

We went through a process that had a certain task to be completed in parallel, and we used a parallel gateway to achieve this functionality. Now you must be aware of when to use a parallel gateway.

Getting started with Sub-Processes

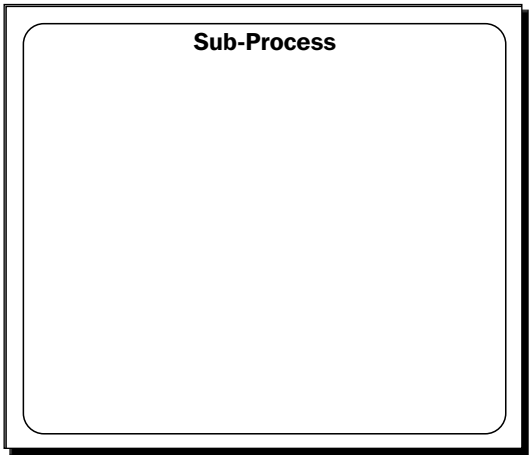
A Sub-Process is a process that is considered a part of a larger process. A Sub-Process comprises various activities, gateways, and events. As a Sub-Process is one business process, it consists of only one non-start event. Every Sub-Process must have at least one end event. There are three types of Sub-Processes, as follows:

- Embedded Sub-Processes

- Standalone Sub-Processes
- Event Sub-Processes

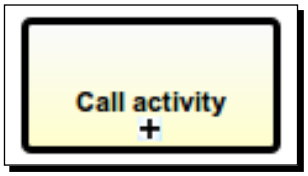
Embedded Sub-Processes

An embedded Sub-Process is defined within the parent process. An embedded Sub-Process is useful to divide the process into small processes and use them. The scope of an embedded Sub-Process is limited to the parent process only. The embedded Sub-Process cannot be accessed outside the parent process. Activiti provides a graphical representation of the **Sub Process** node (as shown in the following diagram) to help create an embedded Sub-Process:



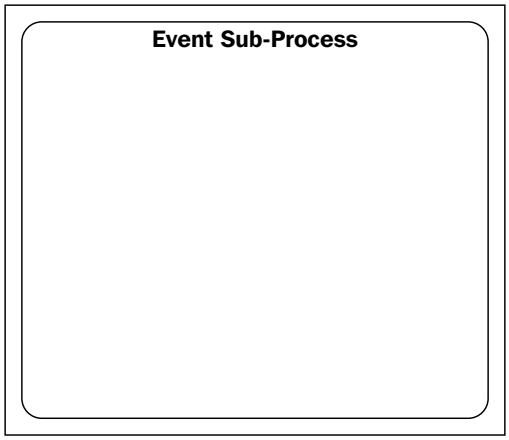
Standalone Sub-Process

We already know how an embedded Sub-Process is defined and used within the parent process. The standalone Sub-process is the exact opposite of the embedded Sub-Process. A standalone Sub-Process refers to a process that is already available within the Activiti Engine and that you want to reuse within your process. Activiti provides a graphical representation called **Call activity** to implement a standalone Sub-Process, as shown in the following diagram:



Event Sub-Process

The two Sub-Processes discussed so far require a start event to trigger the Sub-Process, but the event Sub-Process is a process that can be triggered by the following events: message events, error events, timer events, compensation events, and signal events. An event Sub-Process doesn't have any incoming or outgoing connections. The graphical representation provided by Activiti for an event Sub-Process is **Event Sub-Process** as shown in the following diagram:



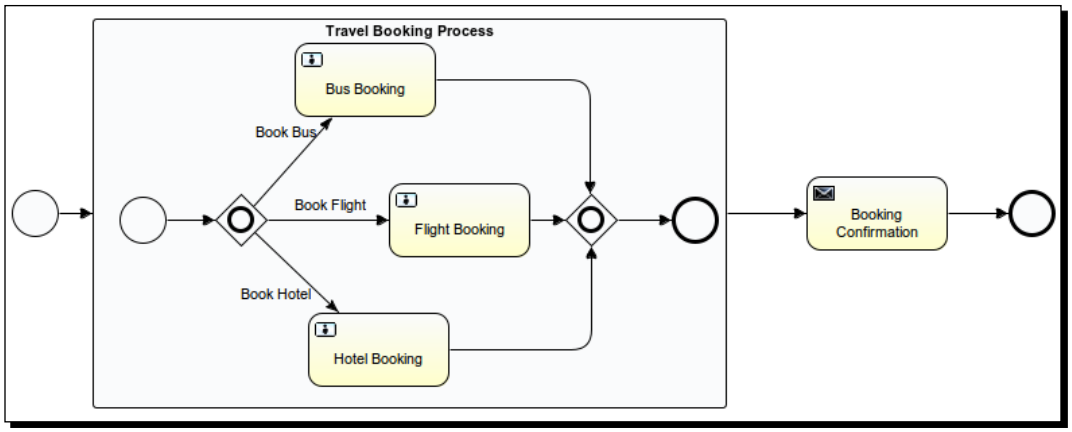
Time for action – working with BPMN 2.0 Sub-Processes

Now that we are aware of the various Sub-Processes available in Activiti, we will create some examples for understanding the working of a Sub-Process. We will split this example into two parts:

- Understanding an embedded Sub-Process
- Understanding a standalone Sub-Process

Understanding an embedded Sub-Process

An embedded Sub-Process represents a process designed within the main process itself. To understand it, we will create a business process as shown in the following screenshot:



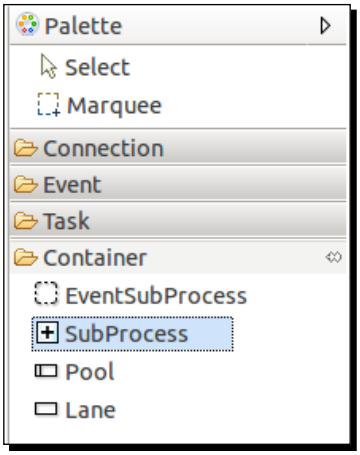
The process illustrated in the preceding screenshot describes a travel booking process where we have created an embedded Sub-Process for booking a flight, bus, and hotel. We will perform the following steps to design the process:

1. We will start with process creation. You can create a new Activiti project in Eclipse and an Activiti diagram within it.
2. We will create a form on the start event. This form will be filled by the potential customer who wishes to make the booking. To create a form, we will have to add the following code in the `.xml` file for the start event:

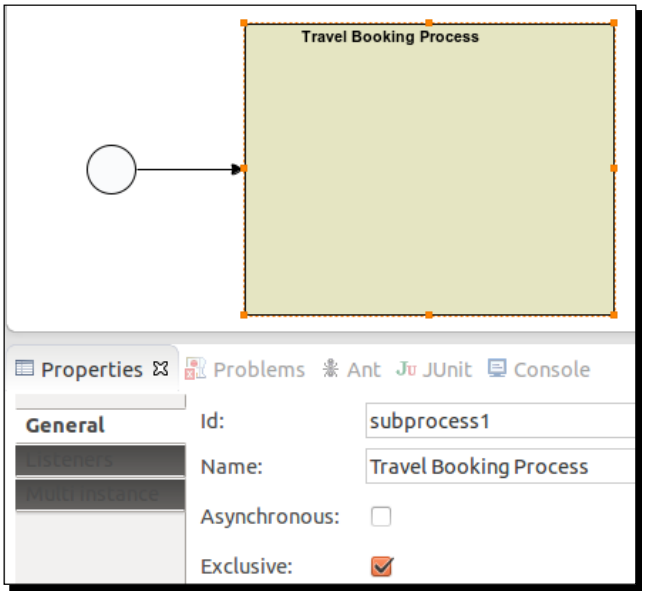
```
<startEvent id="startevent1" name="Start">
      <extensionElements>
        <activiti:formProperty id="customerName" name="Name"
type="string" required="true"></activiti:formProperty>
        <activiti:formProperty id="customerMailId" name="Email Id"
type="string" required="true"></activiti:formProperty>
        <activiti:formProperty id="hotelBooking" name="Hotel
Booking" type="enum">
          <activiti:value id="true" name="Yes"></activiti:value>
          <activiti:value id="false" name="No"></activiti:value>
        </activiti:formProperty>
        <activiti:formProperty id="busBooking" name="Bus Booking"
type="enum">
          <activiti:value id="true" name="Yes"></activiti:value>
          <activiti:value id="false" name="No"></activiti:value>
        </activiti:formProperty>
        <activiti:formProperty id="flightBooking" name="Flight
Booking" type="enum">
          <activiti:value id="true" name="Yes"></activiti:value>
          <activiti:value id="false" name="No"></activiti:value>
        </activiti:formProperty>
```

```
        <activiti:formProperty id="travelPlace"
name="Travel Destination" type="string" required="true"></
activiti:formProperty>
      </extensionElements>
    </startEvent>
```

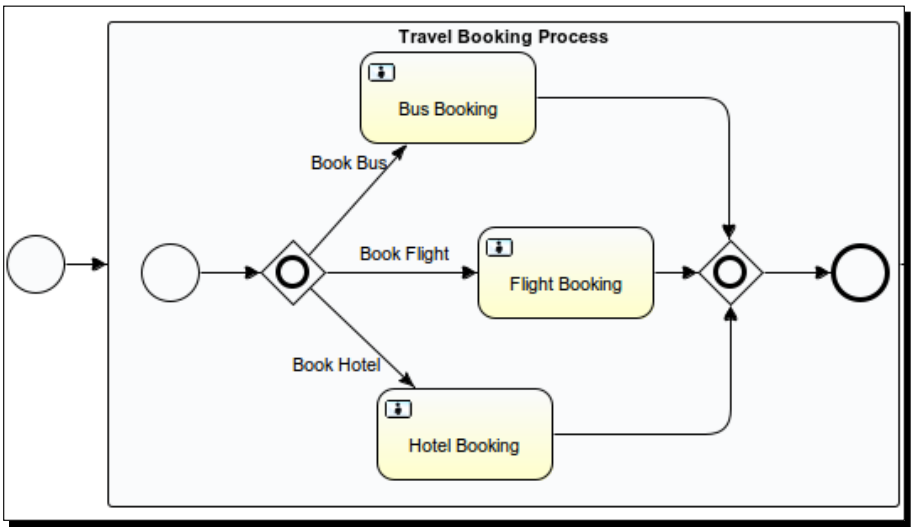
3. We can add a Sub-Process from the **Palette** tab available in Eclipse, as shown in the following screenshot:



4. We need to provide a name to the added Sub-Process as shown in the following screenshot:



5. As it is required to have at least one start event and one end event in a Sub-Process, we will create a traveling process in it with an inclusive gateway and three user tasks, **Bus Booking**, **Flight Booking**, and **Hotel Booking**, as shown in the following screenshot:



6. Now, as these tasks are to be performed by some users and the forms are to be displayed on the execution of these tasks, we will populate the properties of the `Bus Booking`, `Flight Booking`, and `Hotel Booking` user tasks as per the following code:

```
<userTask id="usertask1" name="Bus Booking" activiti:assignee="${c
ustomerName}">
        <extensionElements>
          <activiti:formProperty id="busBookingcustomerName"
name="Customer Name" type="string" expression="${customerName}"
writable="false"></activiti:formProperty>
          <activiti:formProperty id="busBookingDestination"
name="Travel Destination" type="string" expression="${travelPlace}"
writable="false"></activiti:formProperty>
          <activiti:formProperty id="busType" name="Bus Type"
type="enum" required="true">
            <activiti:value id="ac" name="A/c"></activiti:value>
            <activiti:value id="non_ac" name="Non A/C"></
activiti:value>
          </activiti:formProperty>
        </extensionElements>
      </userTask>

<userTask id="usertask2" name="Flight Booking" activiti:assignee="
${customerName}">
```

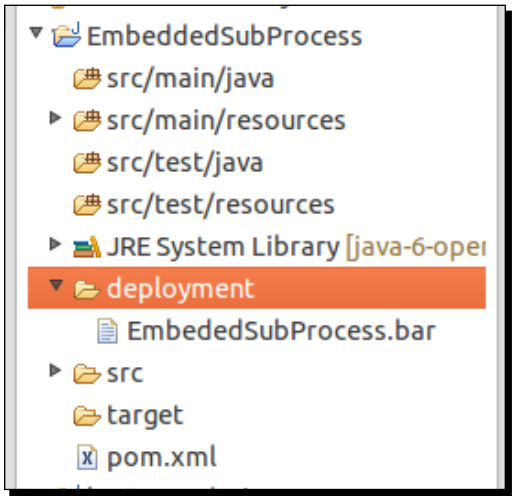
```
        <extensionElements>
          <activiti:formProperty id="flightBookingCustomerName"
name="Customer Name" type="string" expression="${customerName}"
writable="false"></activiti:formProperty>
          <activiti:formProperty id="filghtDestination"
name="Travel Destination" type="string" expression="${travelPlace}"
writable="false"></activiti:formProperty>
        </extensionElements>
      </userTask>
      <userTask id="usertask3" name="Hotel Booking" activiti:assig
nee="${customerName}">
        <extensionElements>
          <activiti:formProperty id="hotelBookingCustomerName"
name="Customer Name" type="string" expression="${customerName}"
writable="false"></activiti:formProperty>
          <activiti:formProperty id="hotelType" name="Type Of
Hotel" type="enum">
            <activiti:value id="5_star" name="5 Star"></
activiti:value>
            <activiti:value id="4_star" name="4 Star"></
activiti:value>
            <activiti:value id="3_star" name="3 Star"></
activiti:value>
          </activiti:formProperty>
          <activiti:formProperty id="hotelRoom" name="Room Type"
type="enum">
            <activiti:value id="ac" name="A/c"></activiti:value>
            <activiti:value id="non_ac" name="Non A/C"></
activiti:value>
          </activiti:formProperty>
        </extensionElements>
      </userTask>
```

7. Once we are done with populating the properties for `Travel Booking Process`, we want an e-mail to be sent to the customer regarding their request for a booking. Once this is done, we need to populate the properties using the following code:

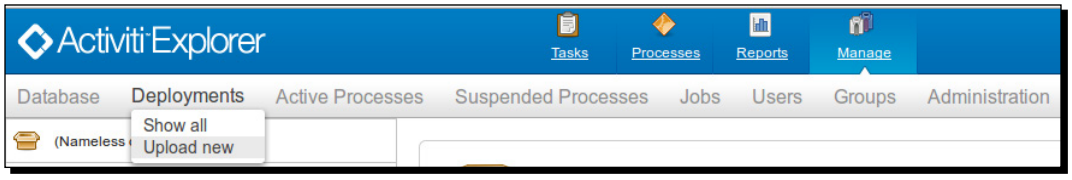
```
 <serviceTask id="mailtask1" name="Booking Confirmation"
activiti:type="mail">
      <extensionElements>
        <activiti:field name="to">
          <activiti:expression>${customerMailId}</
activiti:expression>
        </activiti:field>
        <activiti:field name="subject">
          <activiti:string>Information Regarding your Travel
Booking Process</activiti:string>
        </activiti:field>
```

```
        <activiti:field name="from">
          <activiti:string>contact@attuneuniversity.com</
activiti:string>
        </activiti:field>
        <activiti:field name="text">
          <activiti:expression>Hello ${customerName}

As per your request for traveling to ${travelPlace} following are
the list of hotels, bus and flight details</activiti:expression>
        </activiti:field>
      </extensionElements>
    </serviceTask>
```

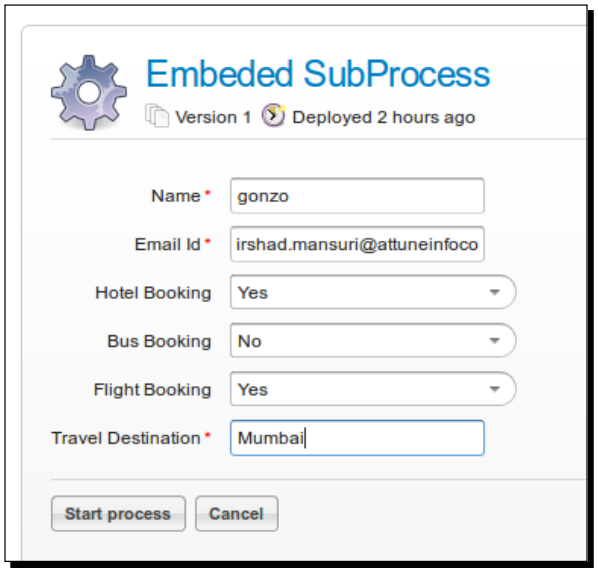
8. Now, to execute our process, create the deployment artifact of your process. To create a deployment artifact, right-click on your project and select the **Create deployment artifacts** option from the list. It will create a `.bar` file for your project in the `deployment` folder, as shown in the following screenshot:



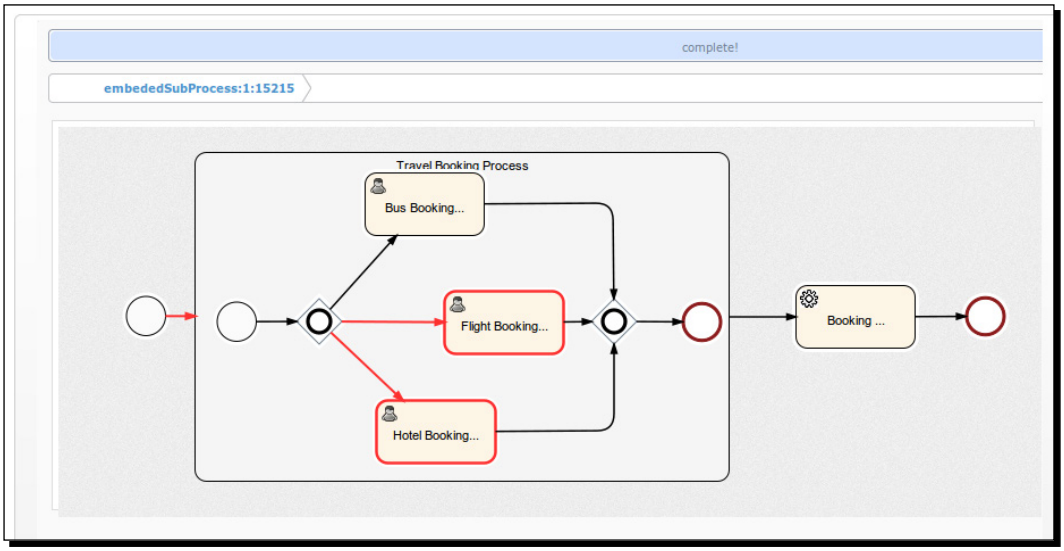
9. To deploy the process into the Explorer for execution, we have to upload the file into the Activiti Explorer. To upload the file, click on the **Upload new** button under **Deployments** as shown in the following screenshot:



10. Once the process is deployed successfully, we can start its execution. A form will pop up at the start of the process (as shown in the following screenshot), which is to be filled.



11. After submitting the preceding form, the Sub-Process will be executed. You can have a look at the **My Instance** tab of the Explorer to monitor your business process. As shown in the following screenshot, the `Flight Booking` and `Hotel Booking` tasks are in execution:

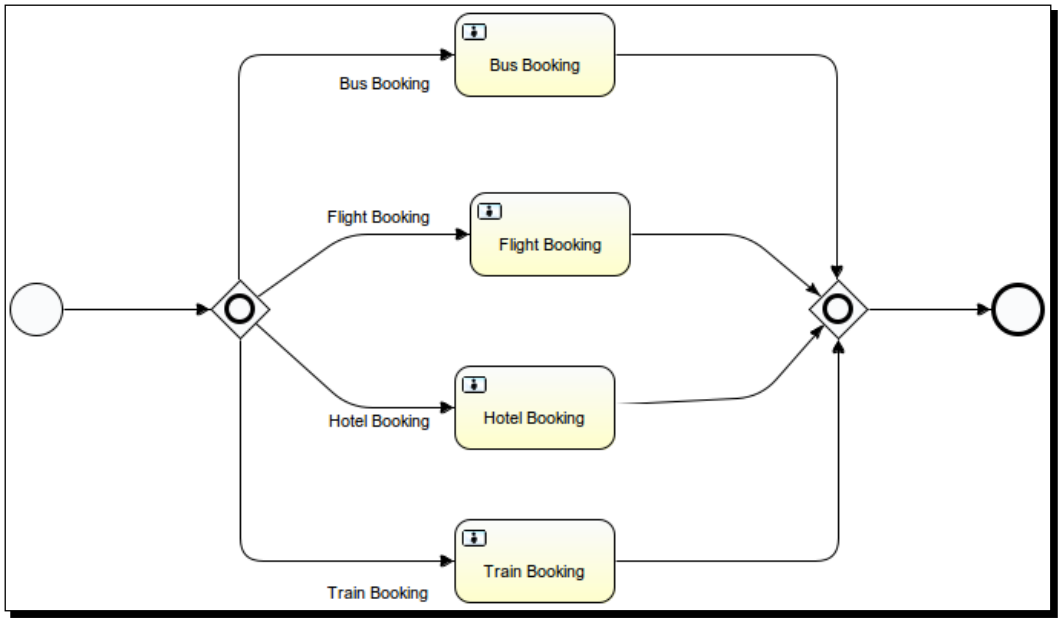


12. As we have populated the **Name** field with `gonzo`, the `Flight Booking` and `Hotel Booking` tasks will be assigned to `gonzo`.
13. Log in with `gonzo` as the user and execute the preceding two tasks. On the completion of the task, an e-mail will be sent to the user regarding the availability of the booking services.

Understanding a standalone Sub-Process

A standalone Sub-Process is also called a call activity. This task is used when we want to call a process that is already modeled and running within the process engine. We will create two separate processes to understand standalone Sub-Processes. We will perform the following steps to implement a standalone Sub-Process:

1. Create a new Activiti project in Eclipse and name it `Booking Process`.
2. Create an Activiti diagram in it and name it `bookingProcess`. This process will only contain the task for the various bookings, as shown in the following screenshot:

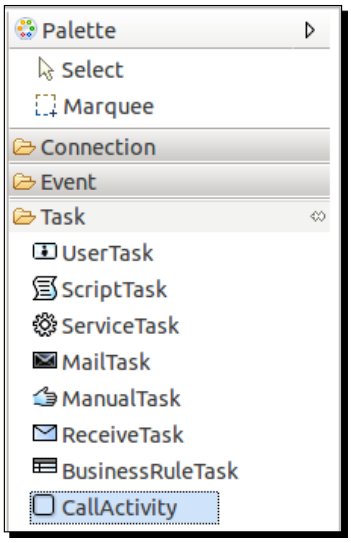


3. On the start event of `BookingProcess`, we will have a form that accepts the information for the type of booking required by the customer. So the following is the code for a start event with form properties:

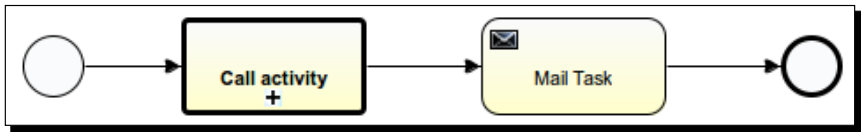
```
<startEvent id="startevent1" name="Start">
      <extensionElements>
        <activiti:formProperty id="bookingPersonName"
name="Booking Person Name" type="string" required="true"></
activiti:formProperty>
        <activiti:formProperty id="busBooking" name="Bus Booking"
type="enum">
          <activiti:value id="true" name="Yes"></activiti:value>
          <activiti:value id="false" name="No"></activiti:value>
        </activiti:formProperty>
        <activiti:formProperty id="flightBooking" name="Flight
Booking" type="enum">
          <activiti:value id="true" name="Yes"></activiti:value>
          <activiti:value id="false" name="No"></activiti:value>
        </activiti:formProperty>
        <activiti:formProperty id="hotelBooking" name="Hotel
Booking" type="enum">
          <activiti:value id="true" name="Yes"></activiti:value>
          <activiti:value id="false" name="No"></activiti:value>
        </activiti:formProperty>
        <activiti:formProperty id="travelDestination"
name="Travel Destination" type="string" required="true"></
activiti:formProperty>
        <activiti:formProperty id="trainBooking" name="Train
Booking" type="enum">
          <activiti:value id="true" name="Yes"></activiti:value>
          <activiti:value id="false" name="No"></activiti:value>
        </activiti:formProperty>
      </extensionElements>
    </startEvent>
```

4. Now create a new Activiti Project in Eclipse and name it `StandaloneSubProcess`.

5. Create an Activiti diagram named `TravelBooking process`. In this process, we will add a **Call Activity** node to call `BookingProcess`. The **Call Activity** task will be available in **Palette**, as shown in the following screenshot:



6. We can design the `TravelBooking` process as shown in the following screenshot:



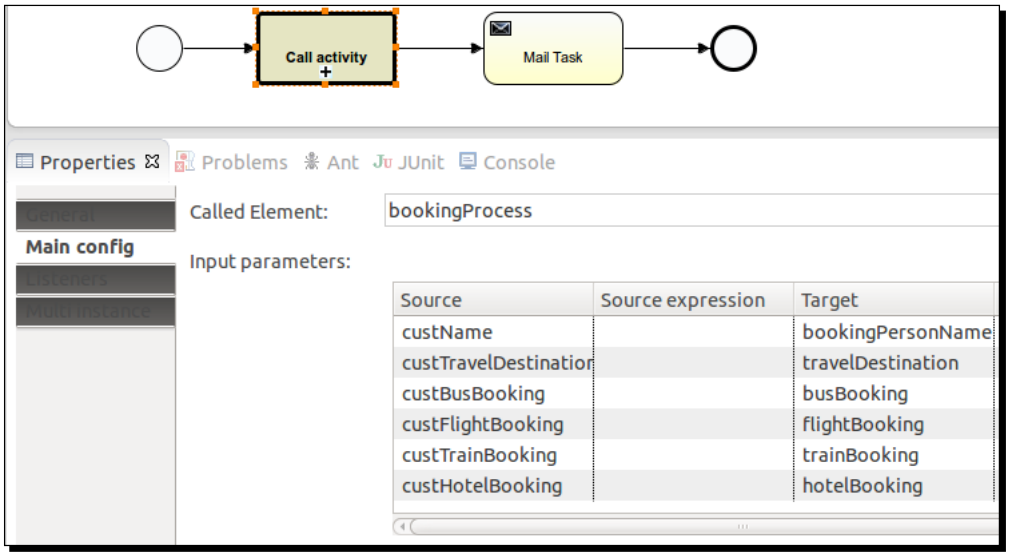
7. On the start event, we will have a form that accepts some input to be passed to the `BookingProcess`. The following code should be added to the start event:

```
<startEvent id="startevent1" name="Start">
      <extensionElements>
        <activiti:formProperty id="custName" name="Name"
type="string" required="true"></activiti:formProperty>
        <activiti:formProperty id="custMailId" name="Email Id"
type="string" required="true"></activiti:formProperty>
        <activiti:formProperty id="custTravelDestination"
name="Travel Destination" type="string" required="true"></
activiti:formProperty>
        <activiti:formProperty id="custBusBooking" name="Bus
Booking" type="enum">
          <activiti:value id="true" name="Yes"></activiti:value>
          <activiti:value id="false" name="No"></activiti:value>
        </activiti:formProperty>
        <activiti:formProperty id="custFlightBooking" name="Flight
Booking" type="enum">
          <activiti:value id="true" name="Yes"></activiti:value>
          <activiti:value id="false" name="No"></activiti:value>
        </activiti:formProperty>
```

```
        <activiti:formProperty id="custTrainBooking" name="Train
Booking" type="enum">
          <activiti:value id="true" name="Yes"></activiti:value>
          <activiti:value id="false" name="No"></activiti:value>
        </activiti:formProperty>
        <activiti:formProperty id="custHotelBooking" name="Hotel
Booking" type="string" default="false" readable="false"
writable="false"></activiti:formProperty>
      </extensionElements>
    </startEvent>
```

8. From the `TravelBooking` process, we will be calling the `BookingProcess` using the **Call activity** task.

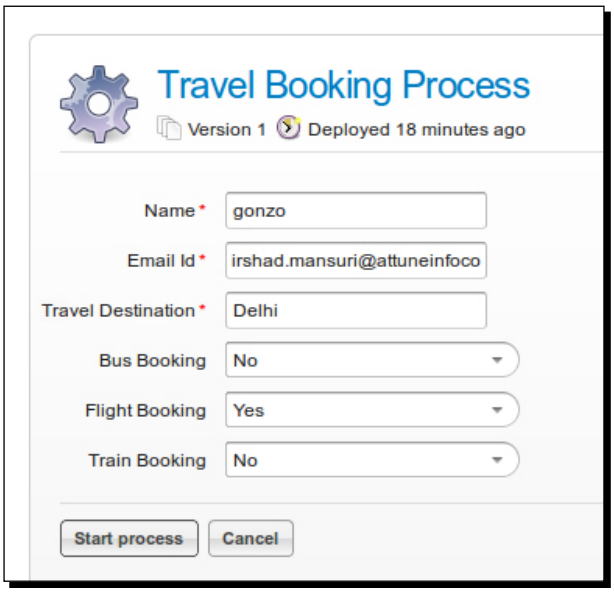
9. We will be populating the properties of **Call activity**. The **Called Element** field will contain the ID of the process to be called, `BookingProcess`, and the **Input parameters** table will be used for mapping the parameters provided at the start of the `TravelBooking` process to `BookingProcess`, as you can see the properties populated in the following screenshot:



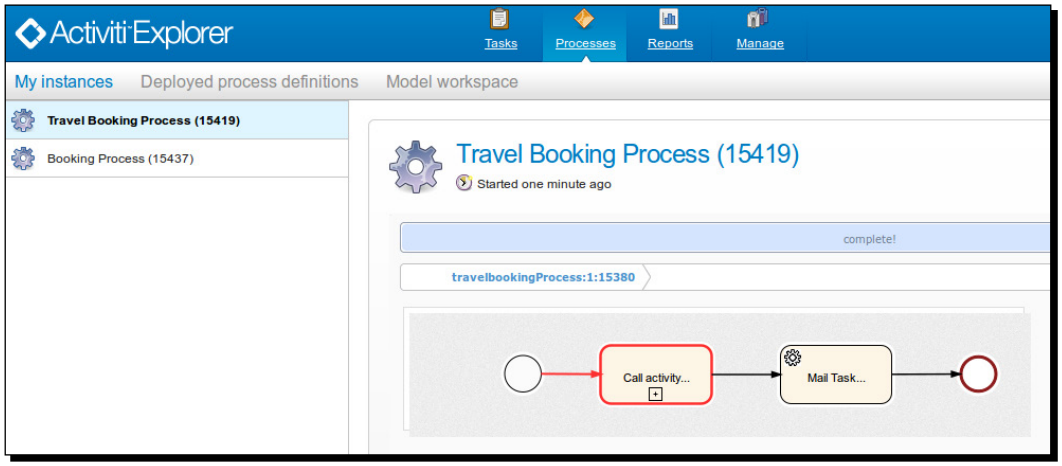
The values passed from the `TravelBooking` process to `BookingProcess` using **Call Activity** will be mapped with the variable of `BookingProcess`.

For executing the `TravelBooking` process, the `BookingProcess` should be deployed into the Activiti process engine, otherwise the `TravelBooking` process will not execute properly. So, first deploy the `BookingProcess` and then the `TravelBooking` process.

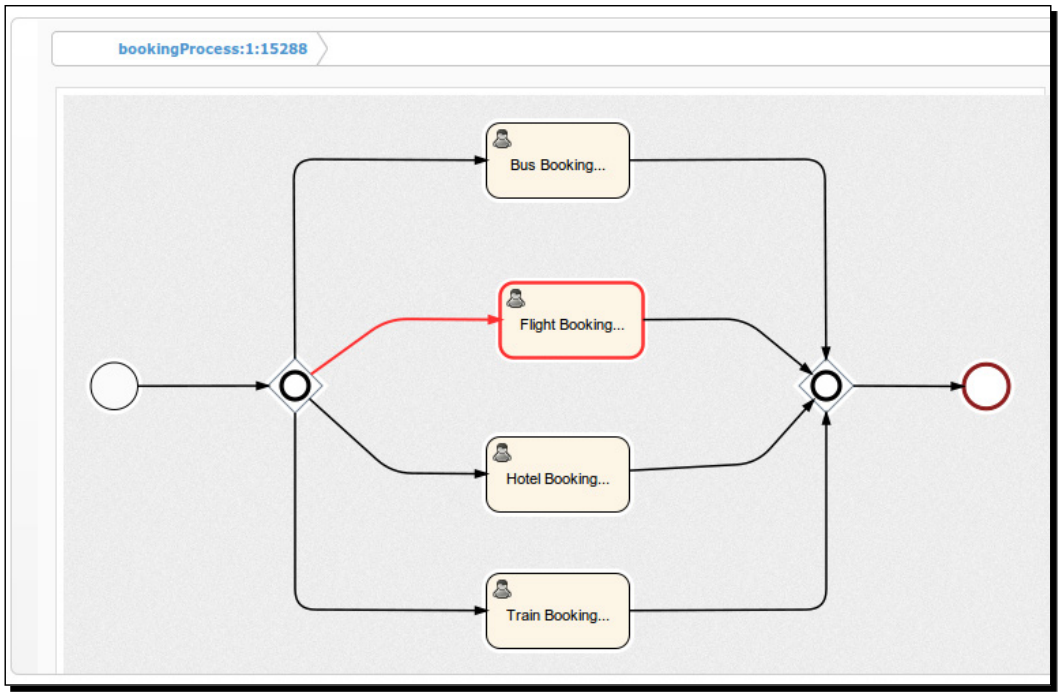
10. So, after completing the creation of the process, deploy the process into the Explorer and start the `TravelBooking` process. On the start of the process, a form will pop up as shown in the following screenshot, where you need to fill in the details of the booking:



11. On starting the `TravelBooking` process, the booking process will start automatically, as shown in the following screenshot:



12. When you select **Booking Process** from **My Instances**, you will find that the `Flight Booking` task is in execution. This is because we have provided the booking type as `Flight Booking`:



13. So, after the completion of the flight task, the process will return to the `TravelBooking` process and execute the mail task for sending an e-mail to the customer.

What just happened?

After a long implementation, we have completed the use of a Sub-Process as an embedded Sub-Process and as a standalone Sub-Process. So now you should be able to implement a process with Sub-Processes.

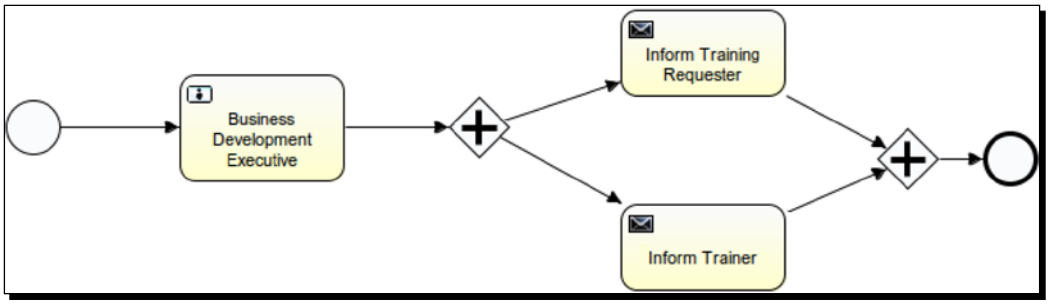
Understanding multi-instance processes

Multi-instance is a way through which we can repeat certain steps of a business process. In terms of programming, it can be considered similar to the `for` loop. The multi-instance functionality can be implemented on almost all of the activities. The list of activities that contain this property are user task, script task, service task, business rule task, e-mail task, Sub-Process, and call activity. Multi-instance can be parallel or sequential.

Time for action – implementing a multi-instance process

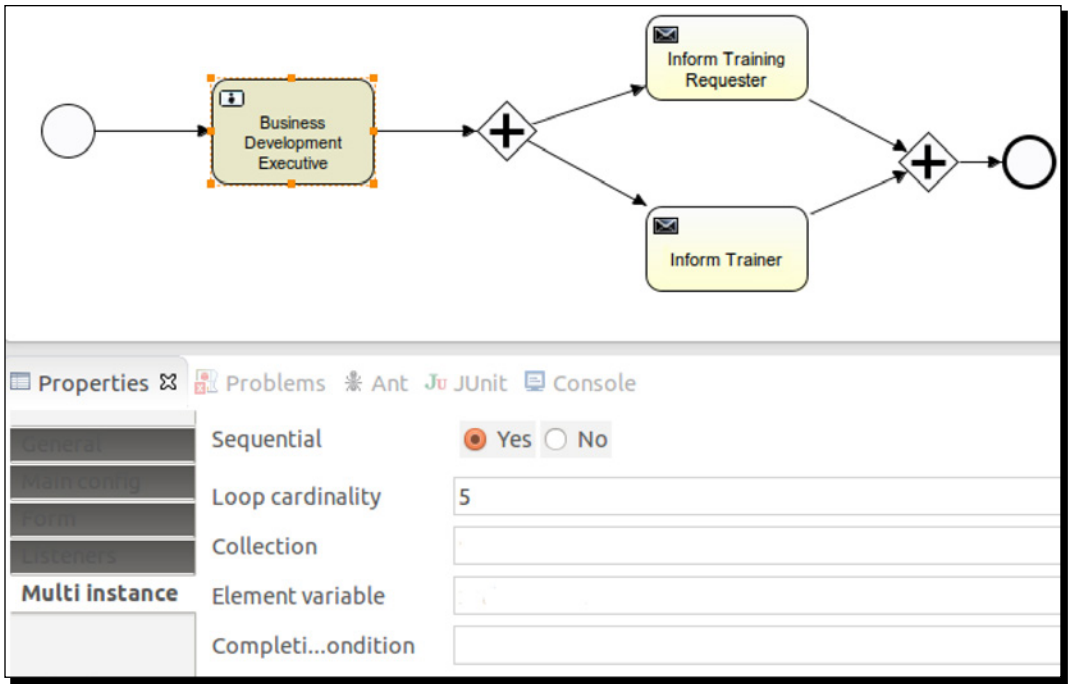
To implement a multi-instance process, we have to perform the following steps:

1. Here we will take the `TrainingRequest` process covered in *Chapter 6*, *The Activiti ProcessEngine API*. The `TrainingRequest` process is shown in the following screenshot:

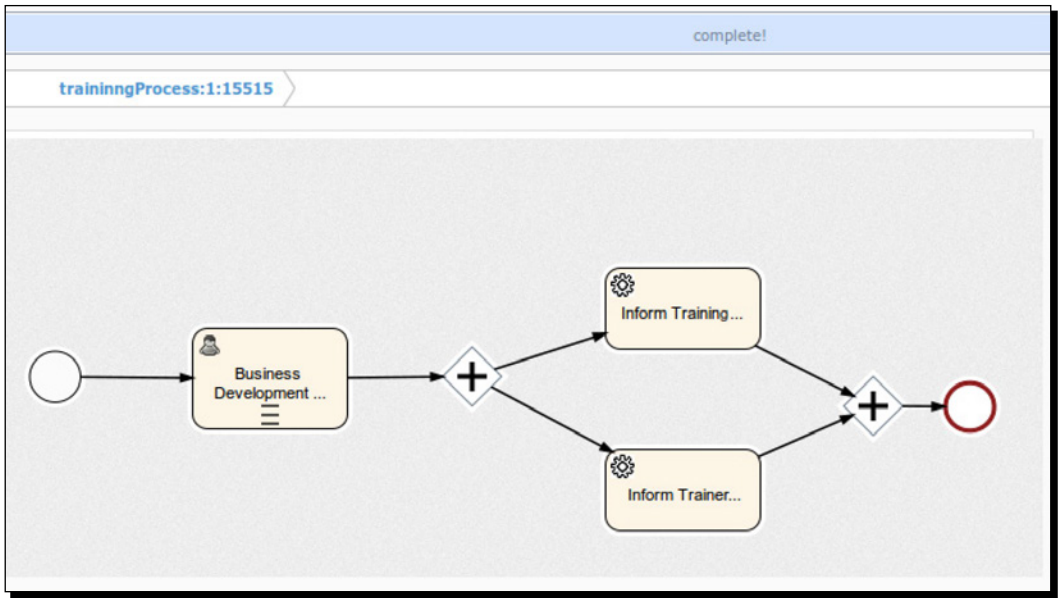


2. We can make the multi-instance process either sequential or parallel; for that, we need to set the `isSequential` parameter of multi-instance. If the value is `true`, Activiti will execute in a sequential order; if `false`, it will execute in parallel. By default, the value is set to `false`.

3. To iterate the `Business Development` task, we can set the **Multi Instance** property's **Loop cardinality** value to some positive value, as provided in the following screenshot:



4. On setting the **Multi Instance** property and deploying the process to the Activiti Engine, the graphical view of the process will be similar to the following screenshot:



5. On setting the `loopCardinality` value, the task will execute for that number of times.

6. Instead of setting the static value to `loopCardinality`, we can set a dynamic value too, as shown in the highlighted code in the following block:

```
 <userTask id="usertask1" name="Business Development Executive"
activiti:assignee="gonzo">
      <extensionElements>
        <activiti:formProperty id="custName" name="Customer Name"
type="string" expression="${customerName}" writable="false"></
activiti:formProperty>
        <activiti:formProperty id="trainTopic" name="Training
Topic" type="string" expression="${trainingTopic}"
writable="false"></activiti:formProperty>
        <activiti:formProperty id="trainDate" name="Training
Date" type="date" expression="${trainingDate}" writable="false"></
activiti:formProperty>
        <activiti:formProperty id="trainerName" name="Trainer
Name" type="string" required="true"></activiti:formProperty>
        <activiti:formProperty id="trainerMailId" name="Trainer
Id" type="string" required="true"></activiti:formProperty>
      </extensionElements>
```

```
        <multiInstanceLoopCharacteristics isSequential="true">
          <loopCardinality>${loopCounter}</loopCardinality>
        </multiInstanceLoopCharacteristics>
    </userTask>
```

7. You can also assign the collection of data for the multi-instance of a task as follows:

```
<userTask id="usertask1" name="Business Development Executive"
      activiti:assignee="gonzo">
      <extensionElements>
        <activiti:formProperty id="custName" name="Customer Name"
          type="string" expression="${customerName}"
writable="false"></activiti:formProperty>
        <activiti:formProperty id="trainTopic" name="Training
Topic"
          type="string" expression="${trainingTopic}"
writable="false"></activiti:formProperty>
        <activiti:formProperty id="trainDate" name="Training Date"
          type="date" expression="${trainingDate}"
writable="false"></activiti:formProperty>
        <activiti:formProperty id="trainerName" name="Trainer
Name"
          type="string" required="true"></activiti:formProperty>
        <activiti:formProperty id="trainerMailId"
          name="Trainer Id" type="string" required="true"></
activiti:formProperty>
      </extensionElements>
      <multiInstanceLoopCharacteristics
        isSequential="true" activiti:collection="BDMList"
        activiti:elementVariable="bdmName">
         </multiInstanceLoopCharacteristics>
    </userTask>
```

What just happened?

We have understood the use of the multi-instance property for Activiti and various ways in which we can use the multi-instance property.

Introducing execution and task listeners

Listeners are used to perform background execution. They are used to perform automatic tasks in the background, such as the autoexecution of any activity, autotransactions, and creating users. Listeners provide great hooks into process execution; they can be used for business process management and simpler things such as flexibly assigning a group of candidate users to a user task. It is used to execute external Java code. We can also perform some expressions during the occurrences of some events.

Activiti supports two types of listeners:

- Execution listeners
- Task listeners

Execution listeners

An execution listener is a class file implementation that can be used during the execution of a process. Let's say we want to fetch some data from an external API or implement some logic on the execution of a process; in this case, we can call it using an execution listener. It can be configured on the following:

- The process itself
- Activities
- Transition

These listeners can be executed on various events, such as the following:

- The start and end events of the process itself
- The start and end events of the activities
- The take event of the transition phase

Task listeners

A task listener is similar to an execution listener; the only difference is that it will execute on a task.

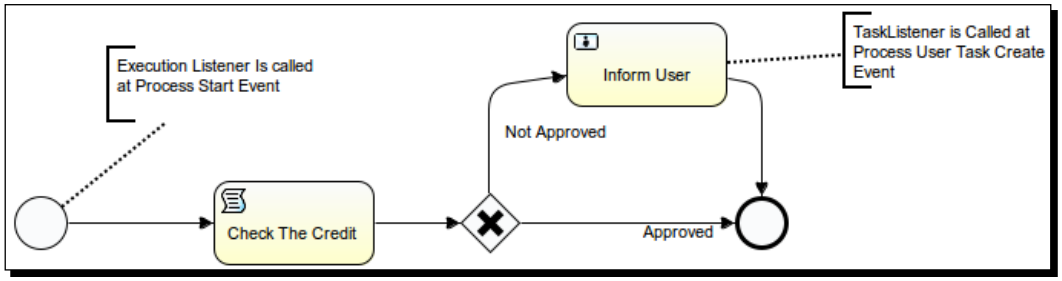
A task listener can be executed on the user task. We can use this listener to allocate tasks to a particular user at runtime. We can complete the task using the task listener.

A task listener can be executed on the create, assignment, and complete events of the user task.

Now let's model a process to use these listeners.

Time for action – implementing execution and task listeners

To get a better understanding of execution and task listeners, we will design a process as shown in the following screenshot:



We will perform the following steps to implement the preceding process:

1. Create an Activiti project in the Eclipse editor named `ExecutionTaskListener`.
2. Create a class `MyExecutionListener` with the following code:

```
public class MyExecutionListener implements ExecutionListener {

  @Override
  public void notify(DelegateExecution execution) throws Exception
{
    // TODO Auto-generated method stub
    long approved_amount_limit;

    approved_amount_limit=50000;
    System.out.println("******************the Start event Listener
is Called************");
    execution.setVariable("approved_amount_limit", approved_
amount_limit);


  }
}
```

3. Create one more class, `MyTaskListener`, for the task listener and write the following code in it:

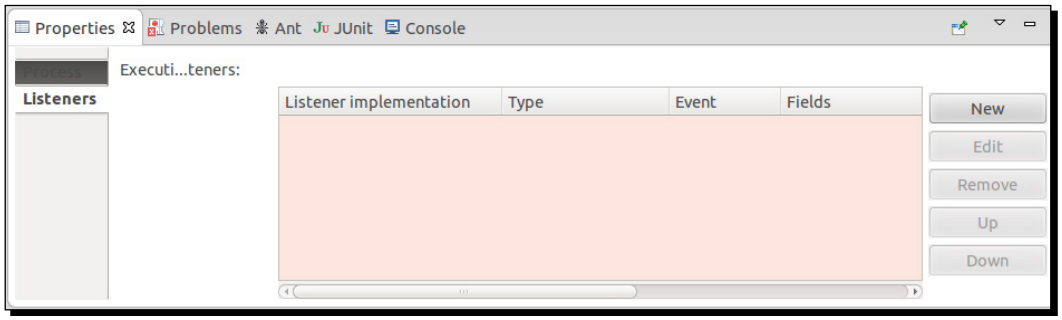
```
public class MyTaskListener implements TaskListener{

  @Override
  public void notify(DelegateTask task) {
    // TODO Auto-generated method stub

    task.setAssignee("kermit");

    System.out.println("Task Listener is called");
  }
}
```

4. Create yet another class, `MyExecution_Transition_Listener`, and place the following code in it:

```
public class MyExecution_Transition_Listener  implements
ExecutionListener{

  @Override
  public void notify(DelegateExecution execution) throws Exception
{
    // TODO Auto-generated method stub

    System.out.println("Your Request Is Approved");
  }
}
```

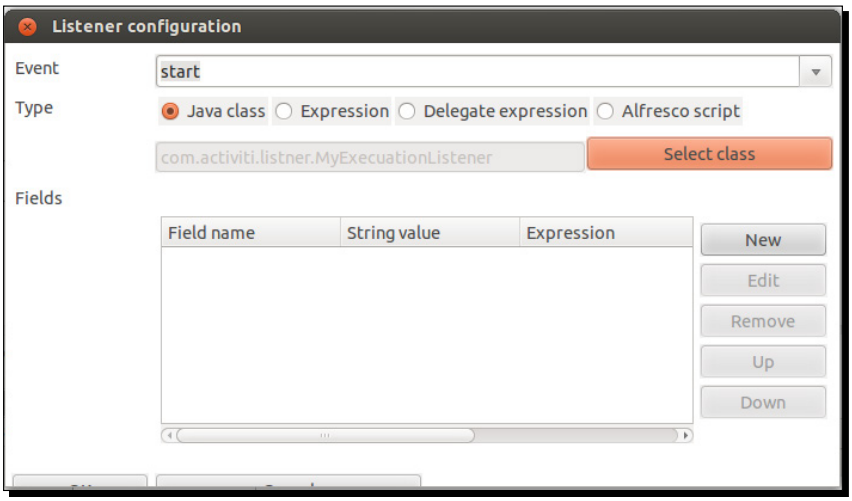
5. Now create a new the Activiti diagram within the project and design a process as displayed in the screenshot that follows.

6. Once finished with the modeling of the process, we will edit the properties of the process. So select the white canvas and open the **Properties** window as shown in the following screenshot:



7. In the **Listeners** option, click on the **New** button; it will pop up a window as shown in the following screenshot. In a process, we can assign listeners on either a start event or an end event. We will have it on the start event; now, we will select the `MyExecutionListener` class that is to be assigned on the start event as shown in the following screenshot:

If you want the listener class to be executed at the end of the process, set `end` as the event type.

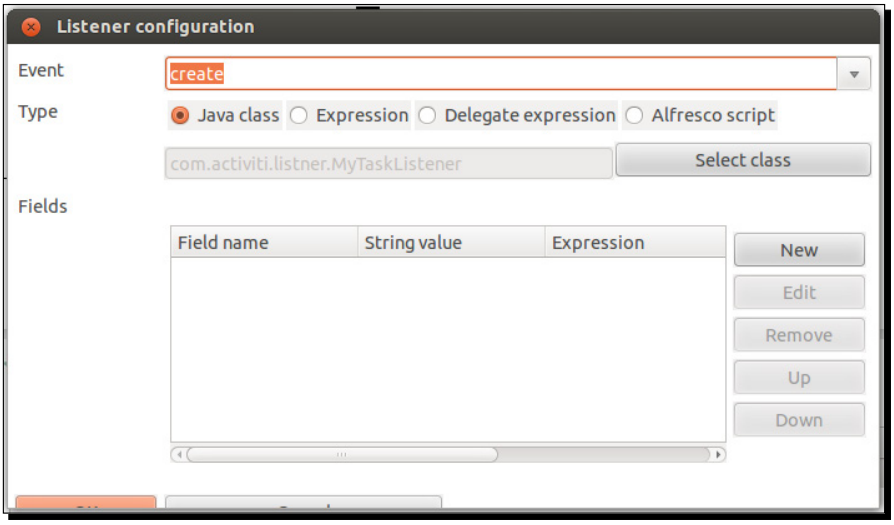


8. Now we will assign a listener class on `usertask`, open the **Properties** window, and go to **Listeners** | **New**.

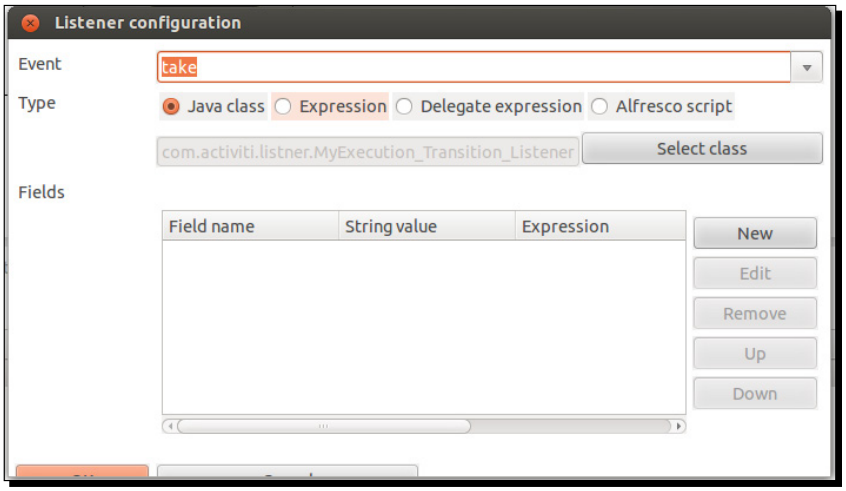
9. On a user task, we can have listeners on the `create`, `assignment`, `complete`, and `all` events. We will have a listener on the `create` event and will select the `MyTaskListener` class, so this class will be executed when `usertask` is created.

If you want the listener classes to be executed when the task is assigned to a user, select the `assignment` event; similarly, if the listener class should be executed on the completion of the user task, select the `complete` event; select `all` if you want the class to be executed on all the mentioned events.

So, we will have a listener on the `create` event and will select the `MyTaskListener` class as shown in the following screenshot:



10. We can also have listeners on the sequence flows. A sequence flow only has a `take` event. So whenever the sequence flow is executed, the class file will be called, as shown in the following screenshot:



11. Now we will create deployment artifacts in Eclipse, which will generate bar graphs for us. We will have to deploy these into the Activiti Explorer. Make sure that you test the process.

What just happened?

We have seen which types of listeners are available. We have implemented a process using the execution listener and task listener.

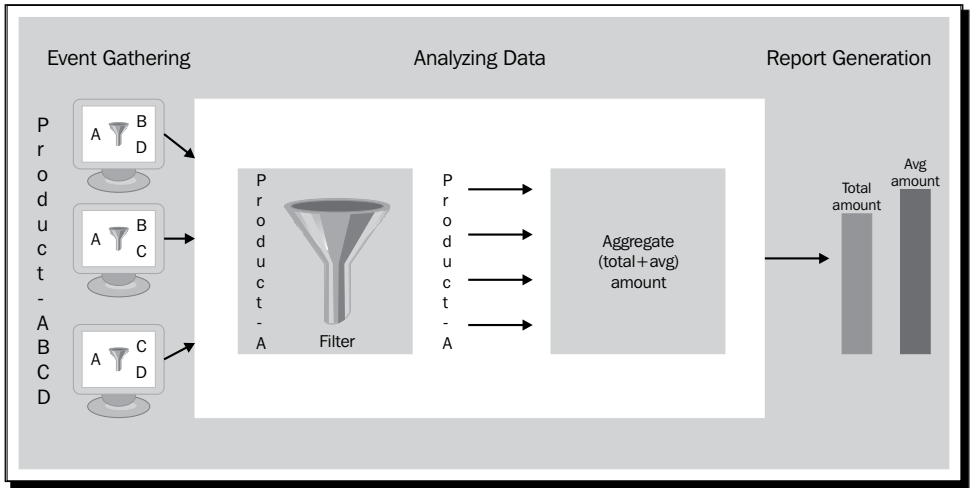
Monitoring workflows using BAM and CEP

There are multiple processes running in an organization, and it becomes difficult to visualize all the processes. To analyze a process, we need to monitor the process.

Understanding BAM

To have a real-time look at all the processes that are running within a business organization, you can make use of **Business Activity Monitoring** (**BAM**). With the help of BAM, we can easily identify the problems or risks and take preventive steps to resolve them and adapt the change. For the implementation of a BAM solution, perform the following steps:

- **Gathering of events**: This step involves the collection of data from events of various sources within an application. The data can be things like the purchase of five pairs of trousers for $50.
- **Analyzing gathered data**: Here we can do some filtering on the gathered data to get an output based on some conditions. For example, a garment store has announced a discount on all their products. Now, the store owner wants to know which products are sold more; for this, he can filter the sold products.
- **Generating reports**: In the previous step, we have seen an example of a garment store. After the analysis, the owner has the data of the most sold products. Based on the reports, he can make a decision as to which products are trending and focus on those.



Understanding CEP

CEP stands for complex event processing. In CEP, the main event can be identified based on the collection of certain basic events occurring simultaneously. For example, consider the following events:

- Church bells ring
- A man wearing a tuxedo appears, with a woman in a white gown at his side
- Rice flies through the air

From the preceding basic events, we can identify that there is a wedding going on. CEP filters, correlates, and performs calculations on real-time event data and generates a new event. CEP works as an inverted database, where the data is stored in the database and then we can fire queries to retrieve the data from the database.

The CEP engine performs different types of operations on the events, such as the filtering of events, the aggregation of events, and the joining of events. Based on the results provided, a pattern matching mechanism that looks for the data that is to be provided as an output is executed.

Monitoring using Esper

Esper is a component for complex event processing. Esper and **Event Processing Language** (**EPL**) provide a highly scalable, memory-efficient, in-memory computing, SQL-standard, minimally latent, real-time streaming Big Data processing engine for medium to high-velocity, high variety data. We can monitor the process using Esper. We will see how we can use Esper to monitor the required time for executing the process. For more information, you can go through this URL: `http://esper.codehaus.org/`.

Time for action – using Esper with Activiti

We will use Esper in Activiti by performing the following steps:

1. We will create a new Activiti project in Eclipse named `EsperinActiviti`, and within this project, we will model an Activiti diagram named `leaverequest_withespertest.bpm`. We will model the diagram as in the following screenshot and assign the user task to `fozzie`:

2. Now we will create two execution listener classes, `ProcessStartExecutionListener` and `ProcessEndExecutionListener`.

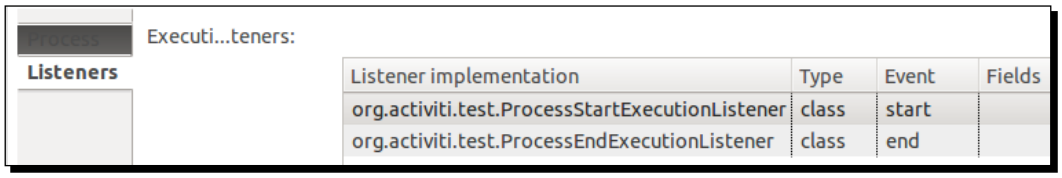
3. The `ProcessStartExecutionListener` class will consist of the following lines of code:

```
public class ProcessStartExecutionListener implements
ExecutionListener {

  @Override
  public void notify(DelegateExecution execution) throws Exception
{
    LeaveRequestReceivedEvent event = new
LeaveRequestReceivedEvent(
        execution.getId(),
        new Date().getTime(),
        (Integer) execution.getVariable("leaveDay"));
        EPServiceProviderManager.getDefaultProvider().
getEPRuntime()
        .getEventSender("LeaveRequestReceivedEvent")
        .sendEvent(event);
  }
}
```

4. The `ProcessEndExecutionListener` class will consist of the following lines of code:

```
public class ProcessEndExecutionListener implements
ExecutionListener {

  @Override
  public void notify(DelegateExecution execution) throws Exception
{
    LeaveRequestProcessedEvent event = new
LeaveRequestProcessedEvent (
      execution.getId(),
      new Date().getTime(),
      (Boolean) execution.getVariable("requestApproved"),
      (Integer) execution.getVariable("leaveDay"));
    EPServiceProviderManager.getDefaultProvider().getEPRuntime()
      .getEventSender("LeaveRequestProcessedEvent")
      .sendEvent(event);
  }
}
```

5. Now we will assign these listeners on the start and end events of the process, as displayed in the following screenshot:



6. We will create a class file named `LeaveRequestProcessedEvent` with the following code. This class file checks the time taken by the process to complete.

```
public class LeaveRequestProcessedEvent {

  // For correlating the events.
  @XmlElement
  private final String processInstanceId;
  @XmlElement
  private final long processedTime;
  @XmlElement
  private final boolean requestApproved;
  @XmlElement
  private final int requestedDay;

  @SuppressWarnings("unused")
  private LeaveRequestProcessedEvent() {
    processInstanceId = null;
    processedTime = 0;
    requestApproved = false;
    requestedDay = 0;
  }

  public LeaveRequestProcessedEvent(String processInstanceId, long
processedTime, boolean requestApproved, int requestedDay) {
    this.processInstanceId = processInstanceId;
    this.processedTime = processedTime;
    this.requestApproved = requestApproved;
    this.requestedDay = requestedDay;
  }

LeaveRequestProcessedEvent method used to take process instance,
processed time and request approve.

  public String getProcessInstanceId() {
    return processInstanceId;
  }

  public long getProcessedTime() {
```

```
    return processedTime;
  }

  public boolean isRequestApproved() {
    return requestApproved;
  }

  public int getRequestedDay() {
    return requestedDay;
  }

  @Override
  public String toString() {
    return "LeaveRequestProcessedEvent{processInstanceId="+process
InstanceId+",processedTime="+processedTime+",requestApproved="+req
uestApproved+",requestedDay="+requestedDay+"}";
  }
}
```

7. Create a `LeaveRequestReceivedEvent` class with the following code. This class file will check the time taken to receive the request for the process.

```
public class LeaveRequestReceivedEvent {

  // For correlating the events.
  @XmlElement
  private final String processInstanceId;
  @XmlElement
  private final long receiveTime;
  @XmlElement
  private final int requestedDay;

  @SuppressWarnings("unused")
  private LeaveRequestReceivedEvent() {
    processInstanceId = null;
    receiveTime = 0L;
    requestedDay = 0;
  }

  public LeaveRequestReceivedEvent(String processInstanceId, long
receiveTime, int requestedDay) {
    this.processInstanceId = processInstanceId;
    this.receiveTime = receiveTime;
    this.requestedDay = requestedDay;
  }

  public String getProcessInstanceId() {
    return processInstanceId;
```

```
  }

  public long getReceiveTime() {
    return receiveTime;
  }

  public int getRequestedDay() {
    return requestedDay;
  }

  @Override
  public String toString() {
    return "LeaveRequestReceivedEvent{processInstanceId="+processI
nstanceId+",receiveTime="+receiveTime+",requestedDay="+requestedD
ay+"}";
  }
}
```

8. Now we will create a test class, `TimeactivitiTest`, for unit testing. We will split this class for better understanding. We create the `TimeactivitiTest` class as follows:

```
public class TimeactivitiTest {

}
```

9. Place the following code in the test class to configure the Esper engine in Activiti:

```
  private EPRuntime epRuntime;
  private EPAdministrator epAdmin;

  @Before
  public void startEsper() {
    Configuration configuration = new Configuration();
    configuration.addEventTypeAutoName("org.activiti.event");
    EPServiceProvider epService = EPServiceProviderManager.getDefa
ultProvider(configuration);
    epRuntime = epService.getEPRuntime();
    epAdmin = epService.getEPAdministrator();
  }
```

10. Now we will create some variables to fetch the average and maximum process time of the process, as shown in the following code:

```
private Queue<Double> avgProcessDurationQueue = new
LinkedList<Double>();
  private Queue<Long> maxProcessDurationQueue = new
LinkedList<Long>();
```

11. Create a test method to perform the testing of the process execution, as shown in the following code:

```
@Test
  public void monitorProcessDuration() {
    epRuntime.sendEvent(new TimerControlEvent(ClockType.CLOCK_
EXTERNAL));

    EPStatement epStatement = epAdmin.createEPL(new StringBuffer()
        .append("select avg(endEvent.processedTime - beginEvent.
receiveTime) as avgProcessDuration, ")
        .append("max(endEvent.processedTime - beginEvent.
receiveTime) as maxProcessDuration ")
        .append("from pattern [every beginEvent=LeaveRequestReceiv
edEvent -> endEvent=LeaveRequestProcessedEvent(processInstanceId=b
eginEvent.processInstanceId)].win:time(5 sec)")
        .toString());

    epStatement.addListener(new UpdateListener () {
      public void update(EventBean[] newEvents, EventBean[]
oldEvents) {
        Assert.assertEquals(1, newEvents.length);
        Assert.assertNull(oldEvents);
        Double avgProcessDuration = (Double) newEvents[0].
get("avgProcessDuration");
        Long maxProcessDuration = (Long) newEvents[0].
get("maxProcessDuration");
        System.out.println
("avgProcessDuration="+avgProcessDuration+", maxProcessDuration="+
maxProcessDuration);
        avgProcessDurationQueue.add(avgProcessDuration);
        maxProcessDurationQueue.add(maxProcessDuration);
      }
    } );

    sendLeaveRequestReceivedEvent (   0, "1", 100);
    assertMonitoredProcessDuration(null, null);

    sendLeaveRequestReceivedEvent ( 300, "2", 200);
    assertMonitoredProcessDuration(null, null);

    epStatement.destroy();
  }
```

12. Add the following methods as we are using them in our test case:

```
private void assertMonitoredProcessDuration(Double
avgProcessDuration, Long maxProcessDuration) {
    Assert.assertEquals(avgProcessDuration,
avgProcessDurationQueue.poll());
    Assert.assertEquals(maxProcessDuration,
maxProcessDurationQueue.poll());
  }

  private void sendLeaveRequestReceivedEvent(long time, String
processInstanceId, int requestedAmount) {
    sendEvent(time, new LeaveRequestReceivedEvent(processInstance
Id, time, requestedAmount));
  }

  private void sendLeaveRequestProcessedEvent(long time, String
processInstanceId, boolean requestApproved, int LeaveedAmount) {
    sendEvent(time, new LeaveRequestProcessedEvent(processInstance
Id, time, requestApproved, LeaveedAmount));
  }

  private void sendEvent(long time, Object event) {
    System.out.printf(" %1$4d : %2$s\n", time, event);
    epRuntime.sendEvent(new CurrentTimeEvent(time));
    epRuntime.sendEvent(event);
  }
```

What just happened?

We went through how to use Esper in Activiti for monitoring our business process. We have also monitored a process by executing an Esper query.

Have a go hero

Having gone through the chapter, feel free to attempt the following. You can refer to the examples implemented earlier.

- Implement a Sub-Process, either embedded or standalone, to issue a driving license based on the applicant's age, and use the same Sub-Process for the vehicle registration process
- Design a process and use task listeners to complete the user task

Pop quiz – the Activiti Process Engine

Q1. Which Sub-process can be used in any main process?

1. Embedded Sub-Process
2. Multi-instance Sub-Process
3. Standalone Sub-Process
4. None

Q2. Which tab is used to implement a multi-instance task?

1. Listener
2. Multi-instance
3. Form
4. None

Q3. Which gateway doesn't require conditions?

1. Exclusive
2. Inclusive
3. Parallel
4. None

Summary

In this chapter, we have learned about advanced workflows. We have seen how a process can be further divided into Sub-Processes. We have implemented processes using multi-instance and parallel gateways. We have also learned about BAM and CEP for Activiti. We can now implement processes using different listeners.

Pop Quiz Answers

Chapter 3, Designing Your Process Using the Activiti Designer

Pop quiz – the Activiti Designer

Q1	2
Q2	2.3

Chapter 4, Management and Monitoring Using the Activiti Explorer

Pop quiz – the Activiti Explorer

Q1	3. Whoever is part of the admin user group
Q2	2. Using the My instances tab
Q3	2. .bar
Q4	2. db.properties
Q5	3. The Archived tab

Chapter 5, Development Using the Process Engine

Pop quiz – the Activiti Process Engine

Q1	3. ProcessEngineConfiguration
Q2	2. activiti-standalone-context.xml
Q3	1. Activiti Spring layer

Chapter 6, The Activiti Process Engine API

Pop quiz – the Activiti ProcessEngine API

Q1	2.Engine Service
Q2	2.Identity Service
Q3	4.Runtime Service

Chapter 7, Working with the REST API

Pop quiz – the REST API

Q1	1. Repository API
Q2	1. process-definitions
Q3	3. POST

Chapter 9, Implementing Advanced Workflows

Pop quiz – Activiti Process Engine

Q1	3. Standalone subprocess
Q2	2. Multi instance
Q3	3. Parallel

Index

A

Activiti
business rules, integrating with 195
deploying, as OSGi bundle 205
Esper with 245-251
integrating, with Apache Karaf 206-211
Liferay, integrating with 188-194
Liferay Portal, using with 187
Activiti BPM
Activiti Designer 10
Activiti Engine 8-10
Activiti Explorer 10
Activiti framework, installing 12
Activiti Modeler 10
Activiti REST 10
downloading 11
downloading, prerequisites 11
module overview 8
starting with 8
Activiti BPM, prerequisites
Eclipse Juno 12
Indigo 12
JDK 6 11
Activiti Designer
about 10, 52
downloading 52, 53
first process, designing 54-65
installing 52, 53
pop quiz 70
process, importing from Activiti Modeler 68-70
process, testing 66, 67
URL 53
Activiti development environment
setting up 105-108
Activiti Engine
about 8, 104
features 8
model, deploying into 48, 49
Process Engine 9
running 122, 123
Activiti Engine layer 105
activities
about 30
call activity 31
Sub-Process 30
task 30
Activiti Explorer
about 10, 74, 75, 103
features 74
manage tab 75
processes, managing with 83
processes tab 75
process instance, starting 76-79
process managing, ways 86, 87
process with 75, 76
production ready database, changing to 99
reporting with 88-92
reports tab 75
tasks tab 74
task with 80
used, for administration 92-98
used, for deploying process 100, 101
user task, working with 80-83

Activiti framework
first process, creating 15-20
installing 12-15
unit testing 21, 22
Activiti | Generate unit test 21
Activiti Modeler
about 35
BPM used 35-44
model, deploying in the Activiti Engine 48, 49
models, exporting 45
models, importing 45
process, importing to the Activiti Designer 68-70
Activiti Process Engine
about 103
configuration, logging 112, 113
configuring 109, 110
database, configuring 110, 111
mail server, configuring 113-121
pop quiz 124
Activiti REST 10
Activiti REST API. *See* **REST API**
Activiti Spring layer 105
addGroupMember method 179
administration
Activiti Explorer used 92-98
annotation 34
Apache Karaf
Activiti, integrating with 206-211
API (Application Programming Interface) 125
archived menu 80
artifacts
about 34
annotation 34
data object 34
group 34
Assignee property 20, 60
associations 33

B

BAM
about 244
solution, implementing 244
BPM
about 26
Activiti Modeler used 35-38
lifecycle 26
standards 28
BPM, lifecycle
design phase 27
execution phase 27
modeling phase 27
monitoring phase 27
optimization phase 27
BPMN 2.0 Sub-Processes
about 222
embedded Sub-Process 222-229
standalone Sub-Process 229-234
BPMN elements
about 28
artifacts 34
connecting objects 32
flow objects 29
pool 33
swim lane 33
buildProcessEngine() method 123
Business Activity Monitoring. *See* **BAM**
Business Process Modeling. *See* **BPM**
business rule management system (BRMS) 195
business rules
implementing, with Activiti 196-204
integrating, in Activiti 195
integrating, with Activiti 196-203

C

call activity 31
Candidate group comma separated property 61
Candidate user comma separated property 61
CEP 245
complex event processing (CEP) 195
Condition property 62
connecting objects
about 32
associations 33
message flow 33
sequence flow 32
createDeployment method 133
createGroup method 179
Create script task 17

D

database
configuration 99
database tab 93
data object 34
deleteDeployment method 166
deployments tab 94
design phase, BPM 27
Download JDK button 11
Drools Expert 195
Drools Fusion 195
Drools Guvnor 195
Drools Planner 195

E

embedded Sub-Process 221-229
employee productivity 89
Empname property 18
end event 29
Esper
about 245
and Event Processing Language (EPL) 245
with Activiti 245-251
event-based gateway 32
events
about 29
end event 29
intermediate event 29
start event 29
Event Sub-Process 222
exclusive gateway 31
execution listener
about 239
and task listener, implementing 240-243
execution phase, BPM 27
Expression property 62

F

flow objects
about 29
activities 30
events 29
gateways 31
FormService
about 10, 126, 148
unit testing 149-153
Form tab 18

G

gateways
about 31
event-based gateway 32
exclusive gateway 31
inclusive gateway 32
parallel gateway 32
General tab 19
getDeployments() method 164
getEngineInfo method 184
getSpecificTable method 182
group 34
groups tab 95

H

HistoryService
about 10, 126, 146
historical activities, querying 146, 148

I

IdentityService
about 9, 126, 145
users, working with 145, 146
Inbox menu 80
inclusive gateway 32
intermediate event 29
involved menu 80

J

jdbcDriver property 110
jdbcURL property 110
jobs tab 94

L

lane 33
LeaveRequestReceivedEvent class 248
libs directory 11

Liferay Portal
about 188
exploring 188
integrating, with Activiti 188-194
using, with Activiti 187
listeners
about 238
execution listener 239
task listener 239
Log4j-config.xml file 113

M

mailServerDefaultFrom property 113
mailServerHost property 113
mailServerPassword property 113
mailServerPort property 113
mailServerUsername property 113
mailServerUseSSL property 113
mail task 30
Main config tab 18, 20
Management
Activiti Explorer 8
Activiti Rest 8
ManagementService
about 10, 126, 145
using, as REST 180-184
manage tab 75
Manage tab, submenus
database tab 93
deployments tab 94
groups tab 95
jobs tab 94
users tab 95
message flow 33
model
deploying, into Activiti Engine 48, 49
modeling phase, BPM 27
Modelling
Activiti Designer 8
Activiti Kickstart 8
Activiti Modeler 8
monitoring phase, BPM 27
multi-instance processes
about 235
implementing 235-238
mvn install command 207
MyTasks menu 80

N

Name property 19
new end event option 18
Notification task 65

O

optimization phase, BPM 27
org.activiti.engine.impl.cfg.JtaProcessEngine-Configuration 110
org.activiti.engine.impl.cfg.StandaloneIn MemProcessEngineConfiguration 110
org.activiti.engine.impl.cfg.Standalone ProcessEngine Configuration 110
org.activiti.engine.ProcessEngines class 122
org.activiti.spring.SpringProcessEngine Configuration 109
OSGi bundle
Activiti, deploying as 205

P

Palette option 16
parallel gateways
about 32, 213
implementing 214-220
password property 110
pool 33
portal. *See* **Liferay Portal**
process
deploying, Activiti Explorer used 100, 101
Process APIs
using, as REST 168-174
Process_Deployment class 170
Process Engine
FormService 10
HistoryService 10
IdentityService 9
ManagementService 10
RepositoryService 9
RuntimeService 9
TaskService 9
ProcessEngine class 122, 126
ProcessEngineConfiguration.createXXX() method 123

ProcessEngineConfiguration instance 123, 126
processes
managing, ways 84-88
managing, with Activiti Explorer 83
processes tab 75
process instance overview 89
Process Virtual Machine. *See* **PVM**
Properties tab 18
PVM 104, 105

Q

queued menu 80

R

reporting
Activiti Explorer used 89-92
reports tab 75, 88
Repository APIs
as REST 161-167
Repository_Deployment class 165, 166
RepositoryService
about 9, 126, 127
accessing 127
new process instance, starting 133-135
process, deploying 128-133
REST API
about 156
management, working with 180-183
processes, working with 168-175
Repository APIs, working with 161-167
REST service, implementing 156-160
tasks, working with 175-177
users, working with 178-180
REST service
implementing 156-159
Run As | JUnit Test 22
Runtime
Activiti Engine 8
RuntimeService 9, 126, 127

S

Script property 64
script task 30
sequence flow 32
service task 30
SimpleLeaveProces 16
standalone Sub-Process 221, 229-234
start event 17, 29
Sub-Processes
about 30, 220
embedded Sub-Process 221
standalone Sub-Process 221
SuspensionActivation class file 144
swim lane
about 33
lane 33
pool 33

T

task duration report 89
task listener
about 239
and execution listener, implementing 240-243
tasks APIs
using, as REST 175-177
TaskService
about 9, 126, 135
process, suspending 142-144
user tasks, creating 138-141
user tasks, querying for 135-137
tasks tab 74
tasks tab, menus
archived 80
inbox 80
involved 80
MyTasks 80
queued 80
TrainingRequest process 235

U

username property 110
users
working with, REST used 178-180
users tab 95
user task
about 30
working with 80-83

W

web content management (WCM) system 188
workflows
  monitoring, BAM used 244
  monitoring, CEP used 245

[PACKT] open source
community experience distilled
PUBLISHING

Thank you for buying
Activiti 5.x Business Process Management Beginner's Guide

About Packt Publishing

Packt, pronounced 'packed', published its first book "*Mastering phpMyAdmin for Effective MySQL Management*" in April 2004 and subsequently continued to specialize in publishing highly focused books on specific technologies and solutions.

Our books and publications share the experiences of your fellow IT professionals in adapting and customizing today's systems, applications, and frameworks. Our solution based books give you the knowledge and power to customize the software and technologies you're using to get the job done. Packt books are more specific and less general than the IT books you have seen in the past. Our unique business model allows us to bring you more focused information, giving you more of what you need to know, and less of what you don't.

Packt is a modern, yet unique publishing company, which focuses on producing quality, cutting-edge books for communities of developers, administrators, and newbies alike. For more information, please visit our website: `www.packtpub.com`.

About Packt Open Source

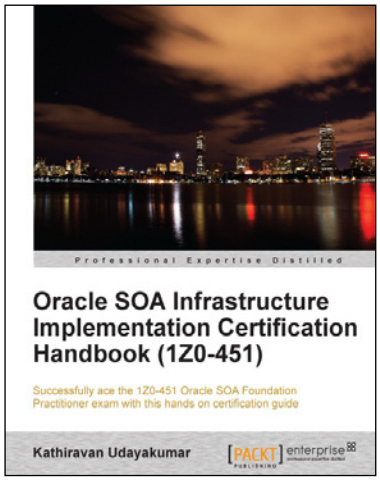
In 2010, Packt launched two new brands, Packt Open Source and Packt Enterprise, in order to continue its focus on specialization. This book is part of the Packt Open Source brand, home to books published on software built around Open Source licences, and offering information to anybody from advanced developers to budding web designers. The Open Source brand also runs Packt's Open Source Royalty Scheme, by which Packt gives a royalty to each Open Source project about whose software a book is sold.

Writing for Packt

We welcome all inquiries from people who are interested in authoring. Book proposals should be sent to `author@packtpub.com`. If your book idea is still at an early stage and you would like to discuss it first before writing a formal book proposal, contact us; one of our commissioning editors will get in touch with you.

We're not just looking for published authors; if you have strong technical skills but no writing experience, our experienced editors can help you develop a writing career, or simply get some additional reward for your expertise.

[PACKT] open source
community experience distilled
PUBLISHING

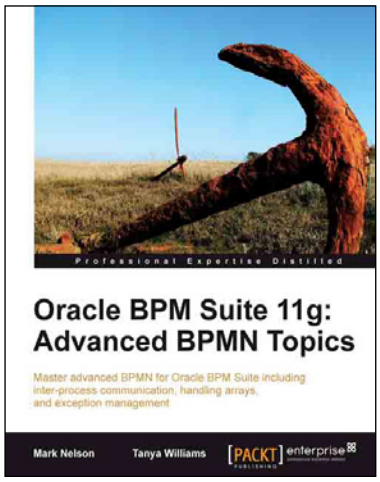


Oracle SOA Infrastructure Implementation Certification Handbook (1Z0-451)

ISBN: 978-1-84968-340-1 Paperback: 372 pages

Successfully ace the 1Z0-451 Oracle SOA Foundation Practitioner exam with this hands on certification guide

1. Successfully clear the first stepping stone towards becoming an Oracle Service Oriented Architecture Infrastructure Implementation Certified Expert.
2. The only book available to guide you through the prescribed syllabus for the 1Z0-451 Oracle SOA Foundation Practitioner exam.
3. Learn from a range of self-test questions to fully equip you with the knowledge to pass this exam.



Oracle BPM Suite 11g: Advanced BPMN Topics

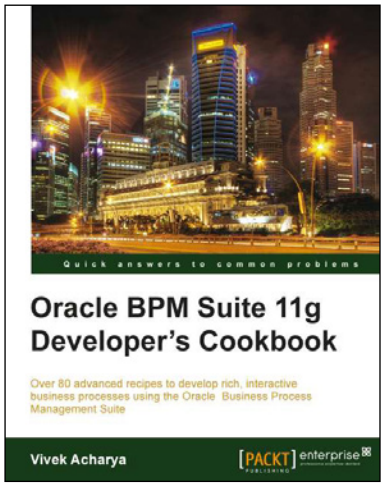
ISBN: 978-1-84968-756-0 Paperback: 114 pages

Master advanced BPMN for Oracle BPM Suite including inter-process communication, handling arrays, and exception management

1. Cover some of the most commonly misunderstood areas of BPMN.
2. Gain the knowledge to write professional BPMN processes.
3. A practical and concise tutorial packed with advanced topics which until now had received little or no documentation for BPM Suite developers and architects.

Please check **www.PacktPub.com** for information on our titles

[PACKT] open source
community experience distilled
PUBLISHING

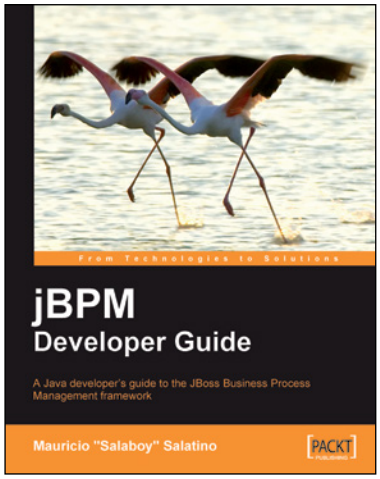


Oracle BPM Suite 11g Developer's Cookbook

ISBN: 978-1-84968-422-4 Paperback: 512 pages

Over 80 advanced recipes to develop rich, interactive business processes using the Oracle Business Process Management Suite

1. Full of illustrations, diagrams, and tips with clear step-by-step instructions and real-time examples to develop Industry Sample BPM Process and BPM interaction with SOA Components.
2. Dive into lessons on Fault, Performance, and Run Time Management.
3. Explore User Interaction, Deployment, and Monitoring.
4. Dive into BPM Process Implementation as process developer while conglomerating BPMN elements.



jBPM Developer Guide

ISBN: 978-1-84719-568-5 Paperback: 372 pages

A Java developer's guide to the JBoss Business Process Management framework

1. Thoroughly understand how the jBPM framework works.
2. Build custom Java Enterprise solutions using the jBPM framework.
3. No experience with jBPM required.
4. Helpful guidance on converting a business analyst's spec into complete, working software.

Please check **www.PacktPub.com** for information on our titles