Executable business processes in BPMN 2.0

Activiti IN ACTION

Tijs Rademakers

FOREWORDS BY Tom Baeyens
AND Joram Barrez

SAMPLE CHAPTER

MANNING

Activiti in Action

by Tijs Rademakers

Chapter 1

brief contents

Part 1

Introducing BPMN 2.0 and Activiti

This first part of the book provides an introduction to the Activiti framework and the background about the BPMN 2.0 standard. In chapter 1, we'll cover how to set up an Activiti environment, starting with the download of the Activiti framework. In chapter 2, you'll be introduced to the main elements of the BPMN 2.0 standard in order to create process definitions. Chapter 3 offers an overview of the Activiti framework's main components, including the Activiti Designer and Explorer. Finally, in chapter 4, we'll discuss the Activiti API with several short code examples.

1 Introducing the Activiti framework

This chapter covers

- Introduction to Activiti
- Installing the Activiti framework
- Implementing a BPMN 2.0 process

Every day, your actions are part of different processes. For example, when you order a book in an online bookstore, a process is executed to get the book paid for, packaged, and shipped to you. When you need to renew your driver's license, the renewal process often requires a new photograph as input. Activiti provides an open source framework to design, implement, and run processes. Organizations can use Activiti to implement their business processes without the need for expensive software licenses.

This chapter will get you up and running with Activiti in 30 minutes. First, we'll take a look at the different components of the Activiti tool stack, including a Modeler, Designer, and a REST web application. Then, we'll discuss the history of the Activiti framework and compare its functionality with its main competitors, jBPM and BonitaSoft.

Before we dive into code examples in section 1.4, we'll first make sure the Activiti framework is installed correctly. At the end of this chapter, you'll have a running Activiti environment and a deployable example.

First, let's look at Activiti's tool stack and its different components, including the modeling environment, the engine, and the runtime explorer application.

1.1 The Activiti tool stack

The core component of the Activiti framework is the process engine. The process engine provides the core capabilities to execute Business Process Model and Notation (BPMN) 2.0 processes and create new workflow tasks, among other things. You can find the BPMN specification and lots of examples at www.bpmn.org, and we'll go into more detail about BPMN in chapter 2. The Activiti project contains a couple of tools in addition to the Activiti Engine. Figure 1.1 shows an overview of the full Activiti tool stack.

Let's quickly walk through the different components listed in figure 1.1. With the Activiti Modeler, business and information analysts are capable of modeling a BPMN 2.0-compliant business process in a web browser. This means that business processes can easily be shared—no client software is needed before you can start modeling. The Activiti designer is an Eclipse-based plugin, which enables a developer to enhance the modeled business process into a BPMN 2.0 process that can be executed on the Activiti process engine. You can also run unit tests, add Java logic, and create deployment artifacts with the Activiti Designer.

In addition to the design tools, Activiti provides a number of supporting tools. With Activiti Explorer, you can get an overview of deployed processes and even dive into the database tables underneath the Activiti process engine. You can also use Activiti Explorer to interact with the deployed business processes. For example, you can get a list of tasks that are already assigned to you. You can also start a new process instance and look at the status of that newly created process instance in a graphical diagram.

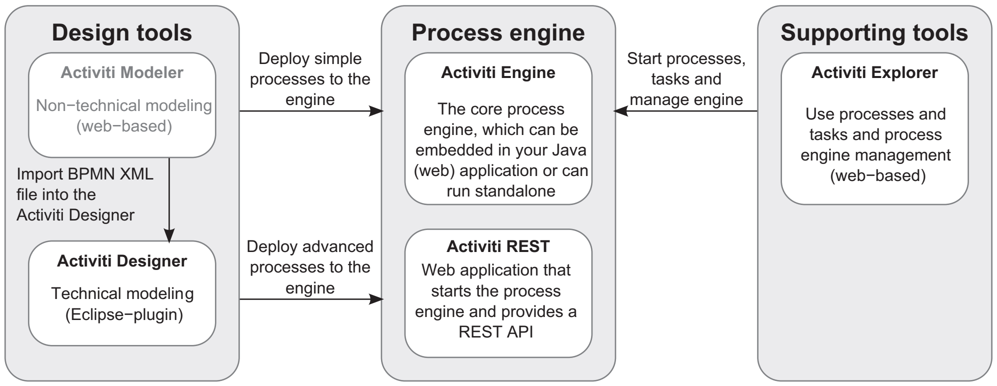

Design tools
Activiti Modeler
Non−technical modeling (web−based)
Import BPMN XML file into the Activiti Designer
Activiti Designer
Technical modeling (Eclipse−plugin)
Deploy simple processes to the engine
Deploy advanced processes to the engine
Process engine
Activiti Engine
The core process engine, which can be embedded in your Java (web) application or can run standalone
Activiti REST
Web application that starts the process engine and provides a REST API
Start processes, tasks and manage engine
Supporting tools
Activiti Explorer
Use processes and tasks and process engine management (web−based)

Figure 1.1 An overview of the Activiti tool stack: in the center, the Activiti process engine, and on the right and left sides, the accompanying modeling, design, and management tools. The grayed-out components are add-ons to the core Activiti framework.

Finally, there's the Activiti REST component, which provides a web application that starts the Activiti process engine when the web application is started. In addition, it offers a REST API that enables you to communicate remotely with the Activiti Engine.

The different components are summarized in table 1.1.

Table 1.1 An overview of the different components of the Activiti tool stack

Component name	Short description
Activiti Engine	The core component of the Activiti tool stack that performs the process engine functions, such as executing BPMN 2.0 business processes and creating workflow tasks.
Activiti Modeler	A web-based modeling environment for creating BPMN 2.0-compliant business process diagrams. This component is donated by Signavio, which also provides a commercial modeling tool, named the Signavio Process Editor.
Activiti Designer	An Eclipse plugin that can be used to design BPMN 2.0-compliant business processes with the addition of Activiti extensions, such as a Java service task and execution listeners. You can also unit test processes, import BPMN 2.0 processes, and create deployment artifacts.
Activiti Explorer	A web application that can be used for a wide range of functions in conjunction with the Activiti Engine. You can, for example, start new process instances and get a list of tasks assigned to you. In addition, you can perform simple process management tasks, like deploying new processes and retrieving the process instance status.
Activiti REST	A web application that provides a REST interface on top of the Activiti Engine. In the default installation (see section 1.1.3), the Activiti REST application is the entry point to the Activiti Engine.

You can't start developing without a clear understanding of the Activiti framework and the architecture that's built around a state machine. Let's take a closer look at the history of the Activiti framework and discuss the Activiti Engine in more detail.

1.2 Getting to know Activiti

When you start working with a new framework, it's always good to know some project background and have an understanding of the main components. In this section, we'll be looking at exactly that.

1.2.1 A little bit of history

The Activiti project was started in 2010 by Tom Baeyens and Joram Barrez, the former founder and the core developer of jBPM (JBoss BPM), respectively. The goal of the Activiti project is to build a rock-solid open source BPMN 2.0 process engine. In the next chapter, we'll talk in detail about the BPMN 2.0 specification, but in this chapter we'll focus on the Activiti framework itself and getting it installed and up and running with simple examples.

Activiti is funded by Alfresco (known for its open source document management system of the same name; see www.alfresco.com and chapter 13 for more details), but Activiti acts as an independent, open source project. Alfresco uses a process engine to support features such as a review and approval process for documents, which means that the document has to be approved by one user or a group of users. For this kind of functionality, Activiti is integrated into the Alfresco system to provide the necessary process and workflow engine capabilities.

> **NOTE** jBPM was used in the past instead of Activiti to provide this process and workflow functionality. jBPM is still included in Alfresco, but it may be deprecated at some point in time.

Besides running the Activiti process engine in Alfresco, Activiti is built to run standalone or embedded in any other system. In this book, we'll focus on running Activiti outside the Alfresco environment, but we'll discuss the integration opportunities between Activiti and Alfresco in detail in chapter 13.

In 2010, the Activiti project started off quickly and succeeded in producing monthly (!) releases of the framework. In December 2010, the first stable and production-ready release (5.0) was made available. The Activiti developer community, including companies like SpringSource, FuseSource, and Mulesoft, has since been able to develop new functionality on a frequent basis. In this book, we'll explore this contributed functionality, such as the Spring integration (chapter 4) and the Mule and Apache Camel integration (chapter 11).

But first things first. What can you do with a process engine? Why should you use the Activiti framework? Let's discuss the core component, the Activiti Engine.

1.2.2 The basics of the Activiti Engine

Activiti is a BPMN 2.0 process-engine framework that implements the BPMN 2.0 specification. It's able to deploy process definitions, start new process instances, execute user tasks, and perform other BPMN 2.0 functions, which we'll discuss throughout this book.

But at its core, the Activiti Engine is a state machine. A BPMN 2.0 process definition consists of elements like events, tasks, and gateways that are wired together via sequence flows (think of arrows). When such a process definition is deployed on the process engine and a new process instance is started, the BPMN 2.0 elements are executed one by one. This process execution is similar to a state machine, where there's an active state and, based on conditions, the state execution progresses to another state via transitions (think again of arrows). Let's look at an abstract figure of a state machine and see how it's implemented in the Activiti Engine (figure 1.2).

In the Activiti Engine, most BPMN 2.0 elements are implemented as a state. They're connected with leaving and arriving transitions, which are called sequence flows in BPMN 2.0. Every state or corresponding BPMN 2.0 element can have attached a piece of logic that will be executed when the process instance enters the state. In figure 1.2, you can also look up the interface and implementing class that are used in the Activiti Engine. As you can see, the logic interface `ActivityBehavior` is implemented by a lot of classes. That's because the logic of a BPMN 2.0 element is implemented there.

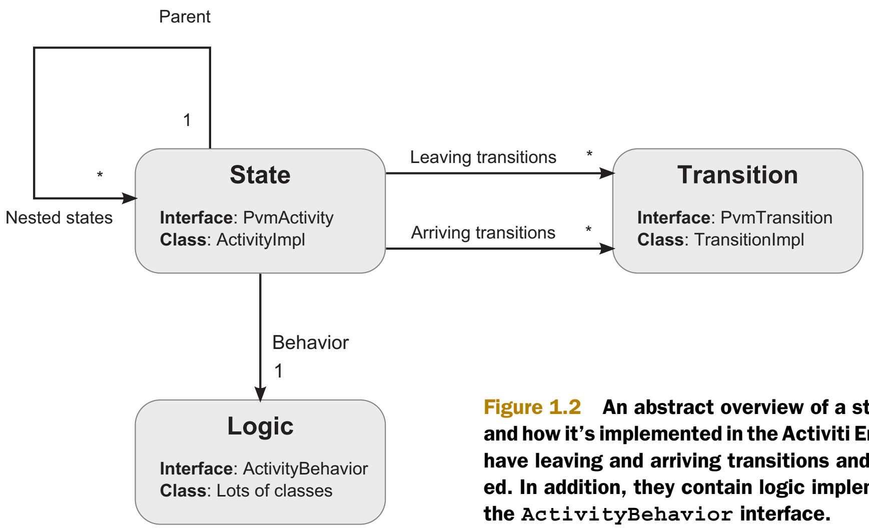


Figure 1.2 An abstract overview of a state machine and how it's implemented in the Activiti Engine. States have leaving and arriving transitions and can be nested. In addition, they contain logic implemented with the `ActivityBehavior` interface.

When you see a complex BPMN 2.0 example later on in the book, remember that, in essence, it's a rather simple state machine. Now let's look at a couple other open source process engines that offer functionality similar to Activiti, and also consider the differences.

1.2.3 Knowing the competitors

When you're interested in an open source process engine like Activiti, it's always good to know a little bit more about the competing open source frameworks. Because the main developers of Activiti were previously involved with the JBoss BPM or jBPM framework, there's also some controversy surrounding this discussion. It's obvious that jBPM and Activiti share a lot of the same architectural principles, but there are also many differences. We'll only discuss the two main open source competitors of Activiti:

- *JBoss BPM or jBPM*—An open source process engine that first supported the custom jPDL process language, but, because version 5.0 supports BPMN 2.0, the jBPM project has merged with the JBoss Drools project (an open source business-rule management framework) and replaced Drools Flow as the rule flow language for the Drools framework.
- *BonitaSoft*—An open source process engine that provides support for the BPMN 2.0 process language. The main differentiators of BonitaSoft are the large set of supported elements and the integrated development environment.

Let's discuss the similarities and differences between Activiti and its two competitors in a bit more detail.

ACTIVITI AND JBPM

Activiti and jBPM have a lot in common: they're both developer-oriented process engine frameworks built around the concept of a state machine (see section 1.2.2).

Because jBPM 5 also implements the BPMN 2.0 specification, a lot of similar functionality can be found. But there are a number of differences that are important to mention; see table 1.2.

Table 1.2 Main differences between Activiti and jBPM

Description	Activiti	jBPM
Community members	Activiti has a base team consisting of Alfresco employees. In addition, companies like SpringSource, FuseSource, and MuleSoft provide resources on specific components. There are also individual open source developers committing to the Activiti project.	jBPM has a base team of JBoss employees. In addition, there are individual committers.
Spring support	Activiti has native Spring support, which makes it easy to use Spring beans in your processes and to use Spring for JPA and transaction management.	jBPM has no native Spring support, but you can use Spring with additional development effort.
Business rules support	Activiti provides a basic integration with the Drools rule engine to support the BPMN 2.0 business rule task.	jBPM and Drools are integrated on a project level, so there's native integration with Drools on various levels.
Additional tools	Activiti provides modeler (Oryx) and designer (Eclipse) tools to model new process definitions. The main differentiator is the Activiti Explorer, which provides an easy-to-use web interface to start new processes, work with tasks and forms, and manage running processes. In addition, it provides ad hoc task support and collaboration functionality.	jBPM also provides a modeler based on the Oryx project and a Eclipse designer. With a web application, you can start new process instances and work with tasks. The form support is limited.
Project	Activiti has a strong developer and user community with a solid release schedule of two months. Its main components are the Engine, Designer, Explorer, and REST application.	jBPM has a strong developer and user community. The release schedule isn't crystal clear, and some releases have been postponed a couple of times. The Designer application is (at the moment of writing) still based on Drools Flow, and the promised new Eclipse plugin keeps getting postponed.

It's always difficult to compare two open source frameworks objectively, and this book is about Activiti. This book by no means presents the only perspective on the differences between the frameworks, but it identifies a number of differences that you can consider when making a choice between them.

Next up is the comparison between Activiti and BonitaSoft.

ACTIVITI AND BONITASOFT

BonitaSoft is the company behind Bonita Open Solution, an open source BPM product. There are a number of differences between Activiti and BonitaSoft:

- Activiti is developer-focused and provides an easy-to-use Java API to communicate with the Activiti Engine. BonitaSoft provides a tool-based solution where you can click and drag your process definition and forms.
- With Activiti, you're in control of every bit of the code you write. With BonitaSoft, the code is often generated from the developer tool.
- BonitaSoft provides a large set of connectivity options to a wide range of third-party products. This means it's easy to configure a task in the developer tool to connect to SAP or query a particular database table. With Activiti, the connectivity options are also very broad (due to the integration with Mule and Camel), but they're more developer focused.

Although both frameworks focus on supporting the BPMN 2.0 specification and offering a process engine, they take different implementation angles. BonitaSoft provides a development tool where you can draw your processes and configure and deploy them without needing to write one line of code. This means that you aren't in control of the process solution you're developing. Activiti provides an easy-to-use Java API that will need some coding, but, in the end, you can easily embed it into an application or run it on every platform you'd like.

As you can see, Activiti is not the only open source process engine capable of running BPMN 2.0 process models, but it's definitely a flexible and powerful option, and one that we'll discuss in detail in this book. Now that you know the different components of Activiti, let's get the framework installed on your development machine.

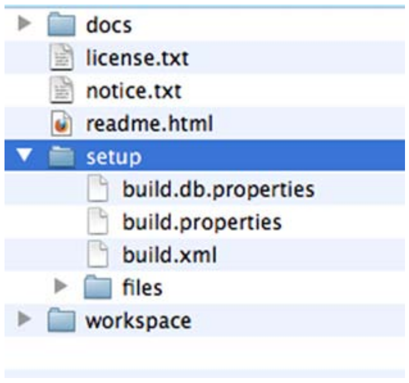
1.3 Installing the Activiti framework

The first thing you have to do is point your web browser to the Activiti website at www.activiti.org. You'll be guided to the latest release of Activiti via the download button. Download the latest version and unpack the distribution to a logical folder, such as

```
C:\activiti (Windows)
/usr/local/activiti (Linux or Mac OS)
```

This isn't the beginning of a long and complex installation procedure—with Activiti, there's a setup directory that contains an Ant build file that installs the Activiti framework. The directory structure of the distribution is shown in figure 1.3.

Before you go further with the installation procedure, make sure that you've installed a Java 5 SDK or higher, pointed the `JAVA_HOME` environment variable to the Java installation directory, and installed a current version (1.8.x or higher) of Ant (http://ant.apache.org). Shortcuts to the Java SDK and the Ant framework are also provided on the Activiti download page.



```
docs
license.txt
notice.txt
readme.html
setup
    build.db.properties
    build.properties
    build.xml
    files
workspace
```

Figure 1.3 The directory structure of the Activiti distribution with the setup directory and the Ant build.xml file as the main parts for the installation procedure.

The last thing to confirm is that you have an internet connection available without a proxy, because the Ant build file will download additional packages. If you're behind a proxy, make sure you've configured the Ant build to use that proxy (more info can be found at http://ant.apache.org/manual/proxy.html).

When you open a terminal or command prompt and go to the setup directory shown in figure 1.3, you only have to run the `ant` command (or `ant demo.start`). This will kick off the Activiti installation process, which will look for a build.xml file in the setup directory. The installation performs the following steps:

1. An H2 database is installed to /apps/h2, and the H2 database is started on port 9092.
2. The Activiti database is created in the running H2 database.
3. Apache Tomcat 6.0.x is downloaded and installed to /apps/apache-tomcat-6.0.x, where x stands for the latest version.
4. Demo data, including users, groups, and business processes, are installed to the H2 database.
5. The Activiti REST and Activiti Explorer WARs are copied to the webapps directory of Tomcat.
6. Tomcat is started, which means that the Activiti Explorer and REST applications are running.
7. Depending on on your OS, a web browser is started by the installation script with the Activiti Explorer URL. On Windows 7, no web browser is started; in other versions of Windows, the web browser is only started if you have Firefox installed.

When the Ant script has finished, you have the Activiti tool stack installed and running. That's not bad for about a minute of installation time. The Ant build file isn't only handy for installing Activiti but also for doing common tasks, like stopping and starting the H2 database (`ant h2.stop`, `ant h2.start`) and the Tomcat server (`ant tomcat.stop`, `ant tomcat.start`) and for re-creating a vanilla database schema (`ant internal.db.drop`, `ant internal.db.create`). It's worth the time to look at the Ant targets in the Ant build file.

The installation of Activiti consists foremost of two web applications being deployed to a Tomcat server and a ready-to-use H2 database being created with example processes, groups, and users already loaded. Figure 1.4 shows the installation result in a schematic overview.

Notice that we haven't yet installed the Activiti Modeler and Designer applications. These components aren't part of the installation script and have to be installed separately. We'll discuss how to do this in chapter 3.

To verify whether the installation has succeeded, the Activiti Explorer, listed in table 1.3, should be available via your favorite web browser. You can use the user `kermit` with password `kermit` to log in. To work with the Activiti REST application, you can use a REST client, such as the REST client Firefox plugin. You can read more about the Activiti REST API in chapter 8.

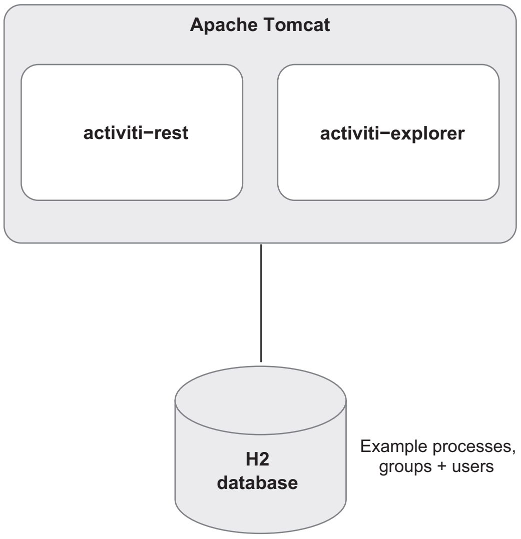


Figure 1.4 An overview of the installation result of the Activiti tool stack, including a running Tomcat server and H2 database with the two Activiti web applications already deployed.

Table 1.3 The URI of the Activiti Explorer and REST web applications available for you after the installation of Activiti

Application name	URI	Short description
Activiti Explorer	http://localhost:8080/activiti-explorer	The Explorer application can be used to work with the deployed processes. This is a good starting point from which to try the example processes.
Activiti REST	http://localhost:8080/activiti-rest/service	The REST application can be used to gain remote access to the Activiti Engine via a REST interface. For all available REST services, you can look in the Activiti user guide that can be found on the Activiti website.

By trying the Activiti Explorer application, you can verify whether the installation was successful. After logging in and clicking on the Process tab, you should get a list of the examples processes that are deployed on the Activiti Engine.

Working with demo processes is fun, but it's even better to try out your own developed business process.

1.4 Implementing your first process in Activiti

Let's try to implement a simplified version of a book order process. We could use the Activiti Modeler to first model the process, and the Activiti Designer to implement and deploy the process, but it's better to start off with a BPMN 2.0 XML document for learning purposes. There won't be any drag-and-drop development, but get ready for some XML hacking.

1.4.1 Say hello to Activiti

We'll keep things simple for now; if you don't understand every construct already, don't be worried—we'll discuss the BPMN 2.0 elements in more detail in chapter 2.

In the following listing, a starter for the BPMN 2.0 XML definition of the book order process is shown with only a start event, an end event, and a sequence flow to connect the two.

Listing 1.1 bookorder.simple.bpmn20.xml document with only a start and end event

```
<?xml version="1.0" encoding="UTF-8"?>
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
             targetNamespace="http://www.bpmnwithactiviti.org">

  <process id="simplebookorder" name="Order book">
    <startEvent id="startevent1" name="Start"/>
    <sequenceFlow id="sequenceflow1"
        sourceRef="startevent1" targetRef="endevent1"/>
    <endEvent id="endevent1" name="End"/>
  </process>
</definitions>
```

A BPMN 2.0 XML definition always starts with a `definitions` element that is identified with a namespace from the OMG BPMN specification. Each process definition must also define a namespace; here, you define a `targetNamespace` with the book's website as its attribute value. Activiti also provides a namespace, which enables you to use Activiti extensions to the BPMN 2.0 specification, as you'll see in chapter 4. You can now run this simple process to test if you've correctly defined the process definition and the environment setup in the right manner.

To test this process, you have to create a Java project in your favorite editor. In this book, we'll use Eclipse for the example description, because the Eclipse Designer is only available as an Eclipse plugin. But it's easier to download the source code from the book's website at Manning (or you can go directly to the Google code repository at http://code.google.com/p/activitiinaction) and import the examples from there.

When you import the `bpmn-examples` project (used in this chapter), the Activiti libraries have to be added to the Java build path. The book's source code uses Maven to retrieve all the necessary dependencies. The sample project's code structure is explained in detail in chapter 4 and appendix A. But, starting from Eclipse Indigo (version 3.7.x), there's good built-in Maven support, so it's easy to get it working. Activate the Maven project capabilities by choosing the Configure–Convert to Maven Project option in the project menu when you right-click on the `bpmn-examples` project in Eclipse. Eclipse will download all the necessary dependencies and configure the classpath for you.

With the dependencies in place, you can look for the `SimpleProcessTest` unit test in the `org.bpmnwithactiviti.chapter1` package of the `bpmn-examples` project. The `SimpleProcessTest` class contains one test method, shown in the following listing.

Listing 1.2 First example of a JUnit test for a Activiti process deployment

```
public class SimpleProcessTest {

  @Test
  public void startBookOrder() {
    ProcessEngine processEngine = ProcessEngineConfiguration
        .createStandaloneInMemProcessEngineConfiguration()
        .buildProcessEngine();

    RuntimeService runtimeService =
        processEngine.getRuntimeService();
    RepositoryService repositoryService =
        processEngine.getRepositoryService();
    repositoryService.createDeployment()
        .addClasspathResource(
            "bookorder.simple.bpmn20.xml")
        .deploy();

    ProcessInstance processInstance =
        runtimeService.startProcessInstanceByKey(
            "simplebookorder");
    assertNotNull(processInstance.getId());
    System.out.println("id " + processInstance.getId() + " " +
        processInstance.getProcessDefinitionId());
  }
}
```

1 **Creates Activiti engine**

2 **Deploys simplebookorder process definition**

3 **Starts bookorder process instance**

In just a few lines of code, you're able to start up the Activiti process engine, deploy the book order process XML file from listing 1.1 to it, and start a process instance for the deployed process definition.

The process engine can be created with the `ProcessEngineConfiguration` 1, which can be used to start the Activiti engine and the H2 database. In this case, the process engine is started with an in-memory H2 database. There are different ways to start up an Activiti engine, and we'll look at the options in detail in chapter 4.

NOTE Activiti can also run on database platforms other than H2, such as Oracle or PostgreSQL.

The next important step in listing 1.2 is the deployment of the bookorder.simple.bpmn20.xml file from listing 1.1. To deploy a process from Java code, you need to access the `RepositoryService` from the `ProcessEngine` instance. Via the `RepositoryService` instance, you can add the book order XML file to the list of classpath resources to deploy it to the process engine 2. The process engine will validate the book order process file and create a new process definition in the H2 database.

It's easy to start a process instance based on the newly deployed process definition by invoking the `startProcessInstanceByKey` method 3 on the `RuntimeService` instance, which is also retrieved from the `ProcessEngine` instance. The key `bookorder`, which is passed as the process key parameter, should be equal to the process `id` attribute from the book order process of listing 1.1. A process instance is stored to the H2 database, and a process instance ID that can be used as a reference to this specific process instance is created. This identifier is very important.

You can now run the unit test and the result should be green. In the console, you should see a message like this:

```
id 4 simplebookorder:1:3
```

This message means that the process instance ID is 4 and the process definition that was used to create the instance was the `simplebookorder` definition with version 1 and the process definition database ID is 3.

Now that we've covered the basics, let's implement a bit more of the book order process; then you can use the Activiti Explorer to claim and finish a user task for your process.

1.4.2 Implementing a simple book order process

It would be a shame to finish chapter 1 with an example that only contains a start and an end event. Let's enhance your simple book order process with a script task and a user task so you can see a bit of action on the Activiti engine. First, the script task will print an ISBN number that will be provided as input to the book order process when it's started in a unit test (like this example) or in the Activiti Explorer. Then, a user task will be used to manually handle the book ordering.

Activiti allows you to use the scripting language you want, but Groovy is supported by default. We'll use a line of Groovy to print the ISBN process variable. The following listing shows a revised version of the book order process.

Listing 1.3 A book order process with a script and user task

```
<definitions xmlns="http://www.omg.org/spec/BPMN/20100524/MODEL"
    targetNamespace="http://www.bpmnwithactiviti.org">

  <process id="bookorder" name="Order book">
    <startEvent id="startevent1" name="Start"/>
    <sequenceFlow id="sequenceflow1" name="Validate order"
        sourceRef="startevent1" targetRef="scripttask1"/>
    <scriptTask id="scripttask1"                              ◁ Defines script task
                name="Validate order"
                scriptFormat="groovy">
      <script>
        out:println "validating order for isbn " + isbn;     ◁ (1) Prints ISBN
      </script>
    </scriptTask>
    <sequenceFlow id="sequenceflow2" name="Sending to sales"
        sourceRef="scripttask1" targetRef="usertask1"/>
    <userTask id="usertask1" name="Work on order">           ◁ (2) Defines user task
      <documentation>book order user task</documentation>
      <potentialOwner>
        <resourceAssignmentExpression>
          <formalExpression>sales</formalExpression>         ◁ Assigns task to sales group
        </resourceAssignmentExpression>
      </potentialOwner>
    </userTask>
    <sequenceFlow id="sequenceflow3" name="Ending process"
        sourceRef="usertask1" targetRef="endevent1"/>
    <endEvent id="endevent1" name="End"/>
  </process>
</definitions>
```

With the two additional tasks added to the process definition, the number of lines in the XML file grows quite a bit. In chapter 3, we'll look at the Activiti Designer, which does the BPMN 2.0 XML generation for you and provides a drag-and-drop type of process development.

The script task contains a `out:println` variable (1), which is a Groovy reserved word within the Activiti script task for printing text to the system console. Also notice that the `isbn` variable can be used directly in the script code without any additional programming.

The user task (2) contains a potential owner definition, which means that the task can be claimed and completed by users that are part of the group sales. When you run this process in a minute, you'll see in the Activiti Explorer that this user task is available in the task list for the user `kermit`, who is part of the `sales` group.

Now that you've added more logic to the process, you also need to change your unit test. One thing you need to add is an `isbn` process variable when starting the process. To test whether the user task is created, you also need to query the Activiti engine database for user tasks that can be claimed by the user `kermit`.

Take a look at the changed unit test in the next code listing. You can again find this unit test class in the `bpmn-examples` project in the `org.bpmnwithactiviti.chapter1` package.

Listing 1.4 A unit test with a process variable and user task query

```
public class BookOrderTest {

  @Test
  public void startBookOrder() {
    ProcessEngine processEngine = ProcessEngineConfiguration
        .createStandaloneProcessEngineConfiguration()
        .buildProcessEngine();

    RepositoryService repositoryService =
        processEngine.getRepositoryService();
    RuntimeService runtimeService =
       processEngine.getRuntimeService();
    IdentityService identityService =
       processEngine.getIdentityService();
    TaskService taskService =
        processEngine.getTaskService();                        <-- (1) Gets TaskService instance
    repositoryService.createDeployment()
        .addClasspathResource("bookorder.bpmn20.xml")
        .deploy();

    Map<String, Object> variableMap =
        new HashMap<String, Object>();
    variableMap.put("isbn", "123456");
    identityService.setAuthenticatedUserId("kermit");
    ProcessInstance processInstance =
        runtimeService.startProcessInstanceByKey(
            "bookorder", variableMap);
    assertNotNull(processInstance.getId());
    List<Task> taskList = taskService.createTaskQuery()
        .taskCandidateUser("kermit")
        .list()
    assertEquals(1, taskList.size());
    System.out.println("found task " +
        taskList.get(0).getName());

    taskService.complete(taskList.get(0).getId());
  }
}
```

Starts process with variable

❷ Sets authenticated user to kermit

Finds tasks available for kermit

The `BookOrderTest` unit test starts a process instance with a `Map` of variables ❷ that contains one variable with a name of `isbn` and a value of `123456`. In addition, when the process instance has been started, a `TaskService` instance ❶ is used to retrieve the tasks available to be claimed by the user `kermit`. Because there's only one process instance running with one user task, you test that the number of tasks retrieved is 1.

Also note that you're not using the in-memory database anymore but have switched (`createStandaloneProcessEngineConfiguration`) to the default standalone H2 database that's installed as part of the Activiti installation procedure. This means that, before running the unit test, the H2 database should be running (`ant h2.start` or `ant demo.start`). Now you can run the unit test to see if your changes work. In the console, you should see a similar output to

```
validating order for isbn 123456
found task Work on order
```

The first line is printed by the Groovy script task in the running process instance. The last line confirms that one user task is available for claim for the user `kermit`. Because a user task is created, you should be able to see this task in the Activiti Explorer. Confirm that Tomcat has been started (`ant tomcat.start` or `ant demo.start`).

Now, point your browser to http://localhost:8080/activiti-explorer and log in with the user `kermit` and the same password. When you click on the link Queued, you should see one task in the group `Sales`. When you click on this `Sales` group, you should see a screen with one user task with the name of Work on Order like the screenshot shown in figure 1.5.

For the sake of completeness, you can claim the user task and see that it becomes available in the Inbox page. There you can complete the task, which triggers the process instance to complete to the end state. But, before you do that, you can click on the process link, Part of process: 'Order Book', to see details about the running process instance, as shown in figure 1.6.

In the process instance overview, you can get the details about the user tasks that aren't yet completed and the process variables of the running instance. The Activiti

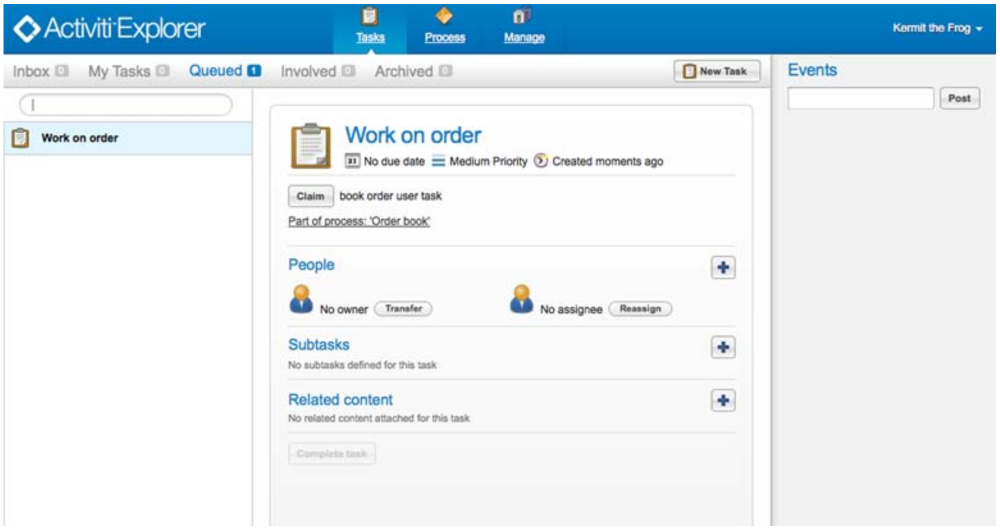
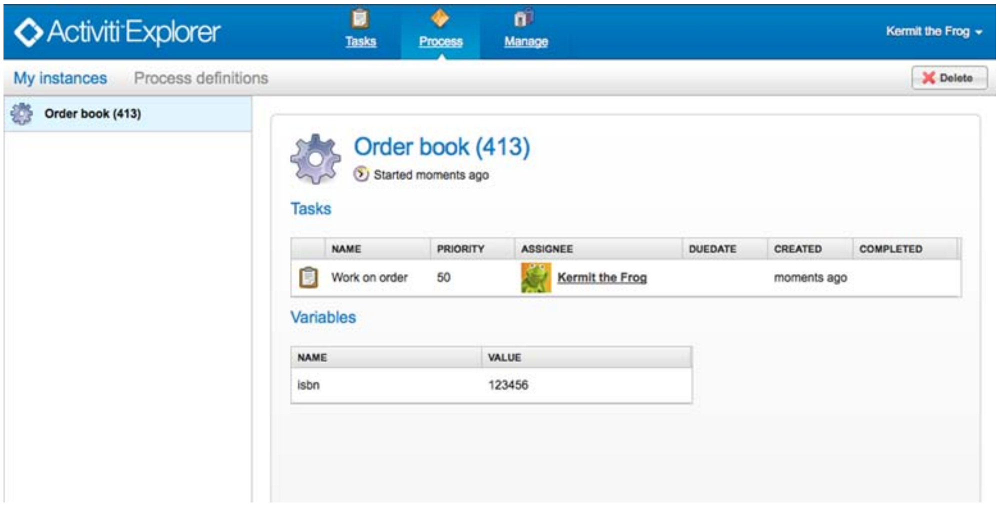


Figure 1.5 A screenshot of the Activiti Explorer showing the user task of the book order process.



NAME	PRIORITY	ASSIGNEE	DUEDATE	CREATED	COMPLETED
Work on order	50	Kermit the Frog		moments ago	

NAME	VALUE
isbn	123456

Figure 1.6 A screenshot of the Activiti Explorer application showing the details of a running process instance with open user tasks and the process instance variables.

Explorer contains a lot more functionality, which we'll discuss throughout the book, starting in chapter 3.

This completes our first journey in the Activiti framework. In the coming chapters, we'll take a more detailed look at the Activiti tool stack and explore how to use Activiti's Java API to, for example, create processes or retrieve management information. But, first, we'll look more closely at BPMN 2.0.

1.5 Summary

In this chapter, we started with an introduction into Activiti, including its history and its competitors. We also got acquainted with the Activiti tool stack and you were able to implement a simple book order process using a script and user task. You also started the Activiti process engine, deployed a book order process, started a process instance, and did some unit testing on it with a couple lines of Java code.

It's obvious that Activiti provides you with a powerful API and tool stack to run your processes. But how can you model and implement these processes? The BPMN 2.0 specification is the foundation for the Activiti Engine, and, to prepare for the examples in the rest of the book, we'll discuss the details of BPMN 2.0 in the next chapter.

METHODOLOGY/JAVA

Free eBook
SEE INSERT

Activiti IN ACTION

Tijs Rademakers

Activiti streamlines the implemention of your business processes: with Activiti Designer you draw your business process using BPMN. Its XML output goes to the Activiti Engine which then creates the web forms and performs the communications that implement your process. It's as simple as that. Activiti is lightweight, integrates seamlessly with standard frameworks, and includes easy-to-use design and management tools.

Activiti in Action introduces developers to business process modeling with Activiti. You'll start by exploring BPMN 2.0 from a developer's perspective. Then, you'll quickly move to examples that show you how to implement processes with Activiti. You'll dive into key areas of process modeling, including workflow, ESB usage, process monitoring, event handling, business rule engines, and document management integration.

What's Inside

- Activiti from the ground up
- Dozens of real-world examples
- Integrate with standard Java tooling

Written for business application developers. Familiarity with Java and BPMN is helpful but not required.

Tijs Rademakers is a senior software engineer specializing in open source BPM, lead developer of Activiti Designer, and member of the core Activiti development team. He's the coauthor of Manning's *Open Source ESBs in Action*.

To download their free eBook in PDF, ePub and Kindle formats, owners of this book should visit manning.com/ActivitiinAction

"A comprehensive overview of the Activiti framework, the Activiti Engine, and BPMN."
—From the Foreword by Tom Baeyens, Founder of jBPM

"A superb book. Best source of knowledge on Activiti and BPMN 2.0. Period."
—From the Foreword by Joram Barrez, Cofounder of Activiti

"The very first book on Actviti ... immediately sets the bar high."
—Roy Prins, CIBER Netherlands

"Just enough theory to let you get right down to coding."
—Gil Goldman
Dalet Digital Media Systems

ISBN 13: 978-1-617290-12-1
ISBN 10: 1-617290-12-2
54999
9 781617 290121

MANNING $49.99 / Can $52.99 [INCLUDING eBOOK]