

### [Activiti in Action](#)

By Tijs Rademakers

*Document management is a natural fit with BPM and BPMN 2.0 processes because a lot of processes create documents or at least access documents in workflow tasks to provide context. In this article, based on chapter 13 of [Activiti in Action](#), author Tijs Rademakers discusses integrating documents in a BPMN 2.0 process flow using the Activiti Engine and Explorer.*

[You may also be interested in...](#)

## *Adding Documents to a BPMN 2.0 Process Definition*

A lot of business processes deal with documents for a lot of different reasons, including auditing and traceability, informing customers, and formal communication with business partners. It can be useful for service tasks in these processes to communicate with a document management system like Alfresco. Later on in this section, we'll use Apache POI to retrieve an Excel document stored in the Alfresco repository and look up the right result value in a decision table, and we'll use the iText framework to store a PDF document sent to the loan request applicant for auditing and traceability purposes.

But first, we'll look at how you can attach documents to user tasks and process instances so you can easily view them in the Activiti Explorer or another process application.

### **Working with task and process instance attachments**

When you want to implement a document review and approval process, it's important that you can also automatically attach the to-be-reviewed document to the reviewers' user tasks.

The Activiti task service interface provides service methods to upload attachments and to couple them to user tasks or process instances. Let's implement a unit test that uses the task service interface to upload a PDF document and attach it to a user task.

#### **Listing 1 Adding attachments to a user task with the task service interface**

```
public class AttachmentTest extends AbstractTest {

    @Rule
    public ActivitiRule activitiRule = new ActivitiRule(
        "activiti.cfg.xml"); #1

    @Test
    public void addAttachment() throws Exception {
        activitiRule.getIdentityService()
            .setAuthenticatedUserId("kermit"); #2
        TaskService taskService = activitiRule
            .getTaskService();
        Task task = taskService.newTask();
        task.setName("Task with CMIS attachments");
        task.setAssignee("kermit");
        taskService.saveTask(task);

        InputStream pdfStream = new FileInputStream(
```

For Source Code, Sample Chapters, the Author Forum and other resources, go to  
<http://www.manning.com/rademakers2/>

```

        "src/main/resources/chapter13/cmis-cheatsheet.pdf");
taskService.createAttachment("application/pdf",
    task.getId(), null, "CMISCheatSheet",
    "CMIS cheat sheet", pdfStream);
#3

taskService.createAttachment("url", task.getId(),
    null, "Alfresco site", "A Alfresco site for Activiti",
    "http://localhost:9090/share/page/site/activiti/" +
    "document-details?nodeRef=workspace://SpacesStore/" +
    "007df67f-28a8-4973-a39b-459c835c0712");
#4

List<Attachment> attachmentList = taskService
    .getTaskAttachments(task.getId());
assertEquals(2, attachmentList.size());
}
}

```

**#1 Connects to H2 at localhost**

**#2 Sets user for creating attachments**

**#3 Creates PDF document attachment**

**#4 Creates URL attachment**

Because you want to view the task that's created in this unit test in the Activiti Explorer, the unit test uses the `activiti.cfg.xml` configuration, which points to the H2 database running at localhost (#1 [not the in-memory H2 database]). You set the authenticated user (#2) because an authenticated user is needed to invoke the `createAttachment` methods to show the name of the attachment uploader in the event stream (see the right column in figure 1).

Then you create a task and assign it to Kermit. With the task identifier, you can invoke the `createAttachment` method on the task service interface to upload a PDF document (#3) to the Activiti Engine database and add it as related content to the newly created task. You also create another attachment that points to a URL (#4). In this example, you point it to the uploaded document.

When you run this unit test, a task is created in the Activiti Engine database with two attachments. You can view this task in Kermit's inbox in the Activiti Explorer, as shown in figure 1.

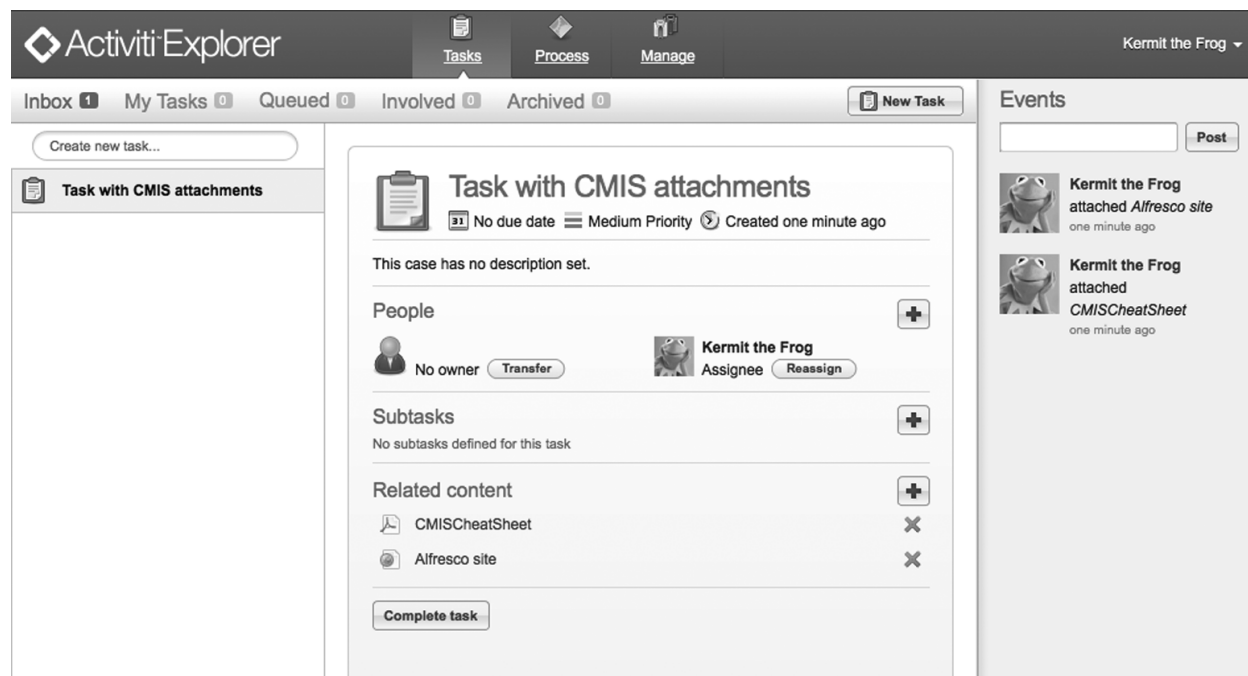


Figure 1 The Activiti Explorer showing the newly created task with two attachments

In the Activiti Explorer, you can see that two related content items are shown in the task details page. When you click on the CMISCheatSheet PDF document, you can download and view it directly. In addition, you can click the Alfresco site link to point your web browser to the uploaded document in Alfresco Share.

There are also two events added in the event stream on the right side of the task page shown in figure 1. This is why you needed to set the authenticated user in listing 1—the event needs a valid user identifier to render the user picture. If you hadn't set the authenticated user, an exception would've been thrown in the Activiti Explorer and the task detail page wouldn't have been shown.

Now let's go ahead and use what you've learned in previous sections. In the next section, you'll implement a process definition that uses CMIS to communicate with the Alfresco Community repository, and you'll add an attachment to the process instance that can be viewed in the Activiti Explorer.

### Implementing a document-aware process definition

A typical business process in a financial organization involves retrieving, creating, or storing a document. In this section, we'll use a loan request example and add logic to retrieve and store Excel and PDF documents.

Figure 2 presents an overview of the process definition and the interaction with the Alfresco repository. The loan request process definition hasn't changed much on a BPMN level. The main difference is that, in the last service task, you create a PDF document that can be sent as a letter to the customer to provide information about the outcome of the loan request approval process. Under the hood, though, there are quite a lot of changes.

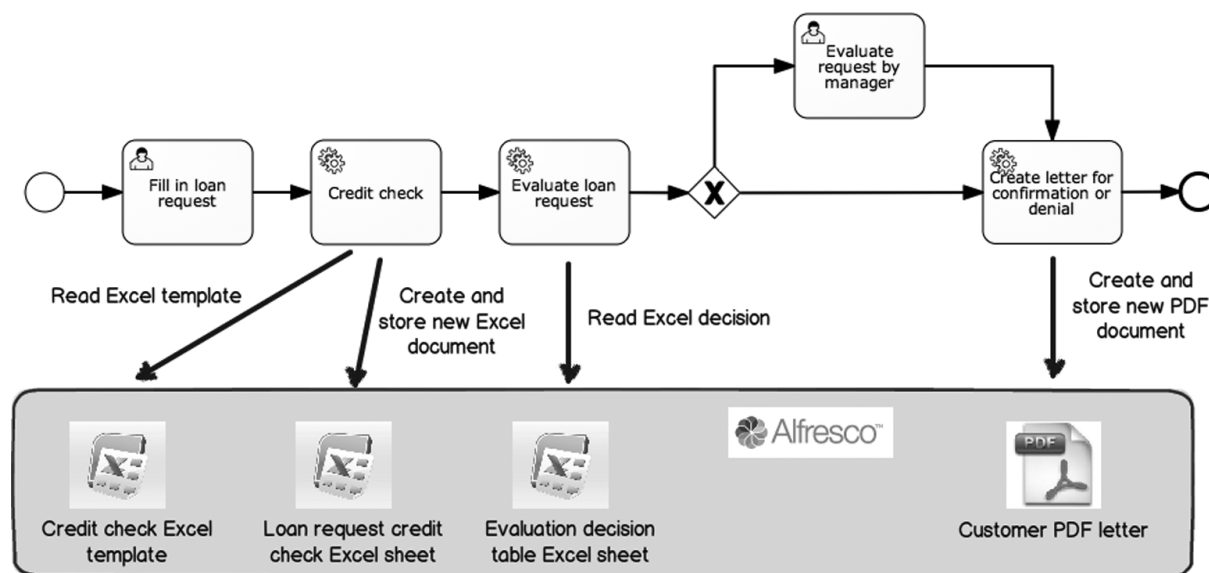


Figure 2 An overview of the loan request process definition we'll be implementing in this section. It shows the interaction with the Alfresco repository to retrieve and store Excel and PDF documents.

The Activiti Explorer showing the newly created task with two attachments that will retrieve an Excel document from the Alfresco repository; the Excel document contains formulas used in the credit check. In addition, a new Excel sheet that contains the loan request details and the outcome of the credit check for that specific loan request is created.

In the second service task, you'll also retrieve an Excel document, but this time it contains a decision table. The "Evaluate loan request" service task will read the decision table and determine the right evaluation value based on the values found. Based on the evaluation value, the "Evaluate request by manager" user task will either be executed or not.

Let's start with implementing the "Credit check" service task that reads an Excel template and then, based on this template, stores a new Excel document containing the loan request details. But, before we do that, let's take a look at the Excel template you'll be using (see figure 3).

|   | A             | B           | C               |
|---|---------------|-------------|-----------------|
| 1 | Customer name |             |                 |
| 2 | Email address |             |                 |
| 3 |               |             |                 |
| 4 | Income        | Loan amount | Loan amount * 2 |
| 5 | 0             | 0           | 0               |
| 6 |               |             |                 |
| 7 | Credit check  | =A5 > C5    |                 |
| 8 |               |             |                 |

Figure 3 The Credit Check Excel template showing the simple formula for the credit check calculation

The Credit Check Excel template shown in figure 3 is simple. In cell C5, the value of the loan amount entered in B5 is doubled. Then, in the credit check formula in cell B7, a check is implemented to see if the income is higher than double the loan amount. The great thing about using Excel templates and a document repository like Alfresco for this kind of logic is that business users can create the formulas themselves without the need for coding. Note that there is a tradeoff when the Excel sheets become large and complex.

In the "Credit check" service task implementation, you'll retrieve the Excel sheet from the Alfresco repository and read it using the Apache POI library, a well-known Apache framework to read and write Microsoft Office documents from Java. In listing 2, the service task implementation of `CreditCheckCMISTask` is shown.

#### Listing 2 Credit check service task implementation using CMIS and POI

```
public class CreditCheckCMISTask implements JavaDelegate {
    private static final String EXCEL_MIMETYPE =
        "application/vnd.openxmlformats-" +
        "officedocument.spreadsheetml.sheet";

    @Override
    public void execute(DelegateExecution execution)
        throws Exception {

        LoanApplication loanApplication = (LoanApplication)
            execution.getVariable("loanApplication");
        POICMISHelper helper = new POICMISHelper();
        helper.openWorkbook("workspace://SpacesStore/" +      #1
            "a5715b04-7422-4e8c-bb8f-def83031103a");

        helper.setCellValue(loanApplication                    #2
            .getApplicant().getName(), 0, 1, true);
        helper.setCellValue(loanApplication
            .getApplicant().getEmailAddress(), 1, 1, true);
        helper.setCellValue(loanApplication
            .getApplicant().getIncome(), 4, 0, false);
        helper.setCellValue(loanApplication                    #3
            .getApplicant().getLoanAmount(), 4, 1, false);

        helper.evaluateFormulaCell(4, 2);
        helper.evaluateFormulaCell(6, 1);                      #4
        loanApplication.getApplicant().setCheckCreditOk(
            helper.getBooleanCellValue(6, 1));

        helper.recalculateSheetAfterOpening();                  #5

        Document document = helper.saveWorkbookToFolder(
            loanApplication.getApplicant().getName(),
            ".xls", EXCEL_MIMETYPE);

        helper.attachDocumentToProcess(                         #6
            execution.getProcessInstanceId(), document, "xls",
            "Credit check sheet for " +
```

For Source Code, Sample Chapters, the Author Forum and other resources, go to  
<http://www.manning.com/rademakers2/>

```

        loanApplication.getApplicant().getName());

    execution.setVariable("loanApplication",
        loanApplication);
    execution.setVariable("documentFolderId",
        helper.documentFolder.getId());
}
}
#1 Gets credit check Excel template
#2 Sets loan application name
#3 Sets loan amount value
#4 Executes Excel formula
#5 Recalculates formula fields when opening
#6 Attaches Excel sheet to process context

```

The service task contains quite a bit of logic, but the core CMIS and POI logic is implemented in the `POICMISHelper` class that you can find in the `bpmn-examples` project of the book's source code. This class has a variable (`ALFRESCO_ADMIN_PASSWORD`) that holds the password of the Alfresco admin user. You should change this password to the password you specified while installing Alfresco.

The first step in the service task retrieves the Excel template shown in figure 2 from the Alfresco repository using the unique identifier (#1). To be able to retrieve the Excel template from the Alfresco repository, you first have to upload it to the Activiti site. Create a folder named `loanapplication` and upload the `creditcheck.xlsx` file from the `src/main/resources/chapter13/cmis` directory (see the `bpmn-examples` project found [here](#)) to that folder. Then, retrieve the object identifier for that object by looking at the document details page or its URI.

Finally, replace the object identifier mentioned in `CreditCheckCMISTask` with the object identifier of the uploaded document in your Alfresco environment. When you retrieve the credit check Excel template from the Alfresco repository, it's parsed by the Apache POI framework in the `POICMISHelper` class. You can now set values like the loan applicant name (#2) and the loan amount (#3) by referencing a cell in the template with a row and column number. For the loan applicant name, you have to create the cell first because the template doesn't yet contain a value. To do this, you set the last parameter of the `setCellValue` method to `true`. For the loan amount, you don't have to create the cell; you can just set the value because there's already a value of 0 in the template. In addition to setting values, you can also execute a formula in the Excel sheet (#4) and get the result value afterwards.

At this point, you have used the credit check logic in the Excel template to get a value of `true` or `false` for the credit check attribute in the loan application process variable. To make sure the formula cells in the Excel sheet contain the right values, you tell the Excel sheet to recalculate all formula cells when the sheet is reopened (#5).

In the last steps of the service task, the Excel document with the loan request information filled in is stored in the Alfresco repository in a subfolder of the loan application folder with the subfolder name that's equal to the value of the name of the loan applicant. Then the new document is coupled to the process instance as a new attachment (#6). You also set the subfolder with the applicant name as a process variable because you need it in the last service task where you'll create the PDF letter.

The credit check service task definition in the `loanrequest.cmis.bpmn20.xml` process definition, which you can find in the `src/main/resources/chapter13/cmis` folder of the `bpmn-examples` project, contains the class attribute configuration pointing to the `CreditCheckCMISTask`, as you'd expect. But there's also an asynchronous attribute, shown in the following code snippet:

```

<serviceTask id="checkCredit"
    activiti:async="true"
    activiti:class="org.bpmnwithactiviti.chapter13.process.task.
        [CA]CreditCheckCMISTask" />

```

By adding the asynchronous attribute, you make sure that the user doesn't have to wait for the whole process instance to finish after completing the first user task. Starting from the credit check service task, the process instance is executed asynchronously, so the user can go on with their work. In this example, the asynchronous execution fits perfectly because the communication with the Alfresco repository, the Excel reading and writing, and the PDF document generation all take some time to complete.

Now that you've got the first service task communicating with the Alfresco repository and reading and writing an Excel document, you can go on with a similar second service task. The loan request evaluation service task

differs a bit because it only reads an Excel document containing a decision table. The service task looks up the evaluation outcome in the decision table (see figure 3) and adds this value to the loan application process variable.

The decision table contains the credit check and loan amount columns for the `if` conditions and the status column for the evaluation result. Let's look at the implementation of the `EvaluationCMISTask` using this decision table. Note that it's a long listing due to the logic needed to process the decision table.

### Listing 3 Implementing a service task using an Excel-based decision table

```
public class EvaluationCMISTask implements JavaDelegate {

    @Override
    public void execute(DelegateExecution execution)
        throws Exception {

        LoanApplication loanApplication = (LoanApplication)
            execution.getVariable("loanApplication");
        POICMISHelper helper = new POICMISHelper();
        helper.openWorkbook("workspace://SpacesStore/" +           #1
            "c70bab92-ce68-444d-8a6c-2f0c43859e0c");

        boolean creditCheck = loanApplication
            .getApplicant().isCheckCreditOk();
        long loanAmount = loanApplication
            .getApplicant().getLoanAmount();
        boolean foundMatch = false;
        boolean reachedEndOfRules = false;
        int rowCounter = 1;                                       #2
        while(foundMatch == false && reachedEndOfRules == false) {
            Cell cell = helper.getCell(rowCounter, 0);
            if(cell == null) {
                reachedEndOfRules = true;
            } else if(creditCheck == helper.getBooleanCellValue(
                rowCounter, 0)) {

                String loanAmountRule = helper.getStringCellValue(   #3
                    rowCounter, 1);
                if("N/A".equalsIgnoreCase(loanAmountRule)) {
                    foundMatch = true;
                } else {
                    int spaceIndex = loanAmountRule.indexOf(" ");
                    String loanAmountRuleCompare =
                        loanAmountRule.substring(0, spaceIndex);     #4
                    String loanAmountRuleValue = loanAmountRule
                        .substring(spaceIndex + 1,                      #5
                            loanAmountRule.length());
                    if("<".equals(loanAmountRuleCompare)) {
                        if(loanAmount < Long.valueOf(
                            loanAmountRuleValue)) {
                            foundMatch = true;
                        }
                    } else if("<=".equals(loanAmountRuleCompare)) {
                        if(loanAmount <= Long.valueOf(
                            loanAmountRuleValue)) {
                            foundMatch = true;
                        }
                    } else if("=".equals(loanAmountRuleCompare)) {
                        if(loanAmount == Long.valueOf(
                            loanAmountRuleValue).longValue()) {
                            foundMatch = true;
                        }
                    } else if(">".equals(loanAmountRuleCompare)) {
                        if(loanAmount > Long.valueOf(                 #6
                            loanAmountRuleValue)) {
                            foundMatch = true;
                        }
                    } else if(">=".equals(loanAmountRuleCompare)) {
                        if(loanAmount > Long.valueOf(
                            loanAmountRuleValue)) {

```

For Source Code, Sample Chapters, the Author Forum and other resources, go to  
<http://www.manning.com/rademakers2/>

```

        foundMatch = true;
    }
}
}
}
if(foundMatch == false) {
    rowCounter++;
}
}
if(foundMatch == false) {
    throw new ActivitiException(
        "No match found in decision table");
}
loanApplication.setStatus(
    helper.getStringCellValue(rowCounter, 2));
execution.setVariable("loanApplication",
    loanApplication);
}
}
}
#1 Gets Excel decision table
#2 Starts at decision table first row
#3 Gets rule value
#4 Gets rule comparator
#5 Gets rule number value
#6 Compares loan amount to rule value
#7 Gets status value

```

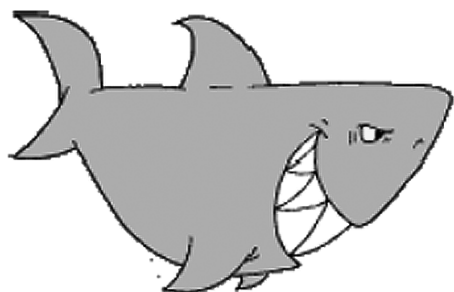
Note that you start with retrieving the Excel decision table document (#1) using the unique document identifier. As in the previous service task, you have to replace the document identifier with the value of your local Alfresco repository after uploading the `evaluation.xlsx` Excel sheet, which can be found in the `src/main/resources/chapter13/cmis` directory.

To process the decision table, you loop through the contents starting with row 1 (#2), which corresponds to row 2 in the Excel sheet shown in figure 3 (remember that Java often starts with 0 instead of 1). You then check if the Excel credit check Boolean value is equal to the process variable value. If it is, you retrieve the loan amount condition from the Excel sheet (#3).

When the value is "N/A", you've found your match in the decision table. For the other cases, you implement logic to validate the comparator (#4) and the loan amount value (#5) used in the condition. For example, when the comparator is `>`, you check whether the loan amount value in the process variable is larger than the one defined in the condition (#6). As you can see, the service task contains this kind of conditional logic for the common comparators.

If a match is found in the decision table, the `while` loop is ended and the value in the status column is retrieved on the same row number as the match (#7). With the loan request evaluation status found, you can update the loan application process variable.

In the exclusive gateway that's connected to the "Evaluate loan request" service task (you can look back at figure 1 for the process definition), the status value is used to determine which sequence flow should be followed. When there's a need for manager approval, a new user task is created. In the other cases, the PDF letter service task is executed. In this service task, you'll generate a PDF document using the process variables to inform the customer about the loan request outcome. An example of this PDF document is shown in figure 4.



Loan Sharks  
4543 1st Street  
Bay City, 38989

E-mail: [info@loansharks.com](mailto:info@loansharks.com)

Dear Mr/Mrs Rademakers,

After analysis regarding your loan request we are happy to inform you that your loan request for \$1000 is approved. Enclosed, you'll find all the details regarding the next steps in the process of your loan request.

With regards,

John Shark  
Manager Loan Sharks

Figure 4 A sample PDF letter generated by the last service task of the loan request process definition. Process variables are used to fill in the values in the PDF document.

To implement the logic needed to generate the PDF document, the iText framework ([www.itextpdf.com](http://www.itextpdf.com)) is used. The iText libraries are listed as Maven dependencies in the pom.xml file of the bpmn-examples project, available in the book's source code.

In the following code snippet, a part of the service task implementation is included.

You can find the full source code in the bpmn-examples project:

```
public class PDFLetterTask implements JavaDelegate {

    public void execute(DelegateExecution execution) throws Exception {
        LoanApplication loanApplication = (LoanApplication)
            execution.getVariable("loanApplication");
        com.itextpdf.text.Document pdf = new com.itextpdf.text.Document();
        pdf.add(new Paragraph("Dear Mr/Mrs " +
            loanApplication.getApplicant().getName() + ","));
        pdf.add(new Paragraph(" "));

        if("approved".equalsIgnoreCase(loanApplication.getStatus()) ||
            "approved by manager".equalsIgnoreCase(
                loanApplication.getStatus())) {
            pdf.add(new Paragraph("After analysis regarding your loan request" +
                " we are happy to inform you that your loan request for $" +
                loanApplication.getApplicant().getLoanAmount() +
                " is approved. Enclosed, you'll find all the details regarding" +
                " the next steps in the process of your loan request."));
        } else {
            pdf.add(new Paragraph("After analysis regarding your loan request" +
                " we regret to inform you that your loan request for $" +
```

For Source Code, Sample Chapters, the Author Forum and other resources, go to  
<http://www.manning.com/rademakers2/>



```

        loanApplication.getApplicant().getLoanAmount() + " is denied.");
    }
}
}

```

The `PDFLetterTask` service task contains more code than is shown here (to include the header and the shark image), but this code snippet shows how you can use `iText` to create a new PDF document without a lot of boilerplate code. To personalize the letter, you make use of the process variables to fill in the customer name and the loan request evaluation outcome. At the end of the service task implementation, the newly created PDF document is uploaded to the same folder in the Alfresco repository where you store the credit check Excel document:

```

POICMISHelper helper = new POICMISHelper();
helper.createCmisSession();
helper.saveDocumentToFolder(outputStream,
    (String) execution.getVariable("documentFolderId"),
    loanApplication.getApplicant().getName(), ".pdf", "application/pdf");

```

The `POICMISHelper` is used to create a new CMIS session, and the PDF document is uploaded to the folder corresponding to the `documentFolderId` process variable.

It's good to take some time to read through the full process definition implementation at this point, starting with the `loanrequest.cmis.bpmn20.xml` file in the `src/main/resources/chapter13/cmis` folder. In the next section, we'll deploy the solution to the Activiti Explorer and start a couple of process instances.

### **Deploying and testing the document-aware process definition**


To test the loan request process definition with the Activiti Explorer, you must deploy the solution to that web application first. Because you're using external libraries like Apache POI and `iText`, you have to copy additional JAR files to the `WEB-INF/lib` directory of the Activiti Explorer web application.

Execute the following steps to get the process definition deployed:




1. Execute the `build.xml` Ant build file in the `src/main/resources/chapter13/cmis` directory by running its default target, `create.cmis`.
2. In the `dist/enginelibs` subdirectory, the `cmis.jar` file is created in step 1 containing the service task and listener classes. In addition, the libraries needed for Apache POI and `iText` are copied there (also by the Ant script of step 1). Copy all of these JAR files to the `WEB-INF/lib` directory of the Activiti Explorer web application in Tomcat.
3. If the Activiti Tomcat server is still running, stop it by executing the command `tomcat.stop` in the setup directory of your Activiti installation. Then, start it up again by running the `tomcat.start` command from that same setup directory.
4. Open the Activiti Explorer in a web browser and log in with Kermit.
5. In the `dist/deploy` subdirectory, a `cmis.bar` file containing the BPMN 2.0 XML process definition is generated. In the deployments tab, upload this BAR file to the Activiti Explorer.

After executing these steps, you're good to go and can start a new process instance. Fill in the required fields of the user task form using an income value of 50,000 and a loan amount value of 20,000. This should lead to creating a new user task with management as the candidate group.

At this point, the credit check and loan request evaluation service tasks are executed, and the credit check Excel document should be attached to the process instance. Figure 5 shows a screenshot of the claimed manager approval user task that contains the credit check Excel document as related content. You can open the Excel document by clicking on it to see if the right values are filled in.



## Evaluate loan request by manager



 No due date
  Medium Priority
  Created moments ago

This case has no description set.

Part of process: 'Process to handle a loan request with CMIS'

---

### People

 No owner [Transfer](#)
 **Kermit the Frog**  
Assignee [Reassign](#)


---

### Subtasks

No subtasks defined for this task

---

### Related content

 John Doe-20111021124533

---

**Fill in the form below and complete the task:**

Customer name

Income of customer

Requested loan amount

Outcome of credit check

Do you approve the request? \*

Motivation

[Complete task](#) [Reset form](#)

Figure 5 The Activiti Explorer showing the claimed manager approval user task with the credit check Excel document as related content

Now approve the request and complete the user task. When you do, the PDF letter will be generated and stored in the Alfresco repository.

Once the process instance has been completed, you can check that the credit check Excel and PDF letter documents are available in the Alfresco repository in the loanapplication folder and the subfolder based on the applicant's name (see figure 6).

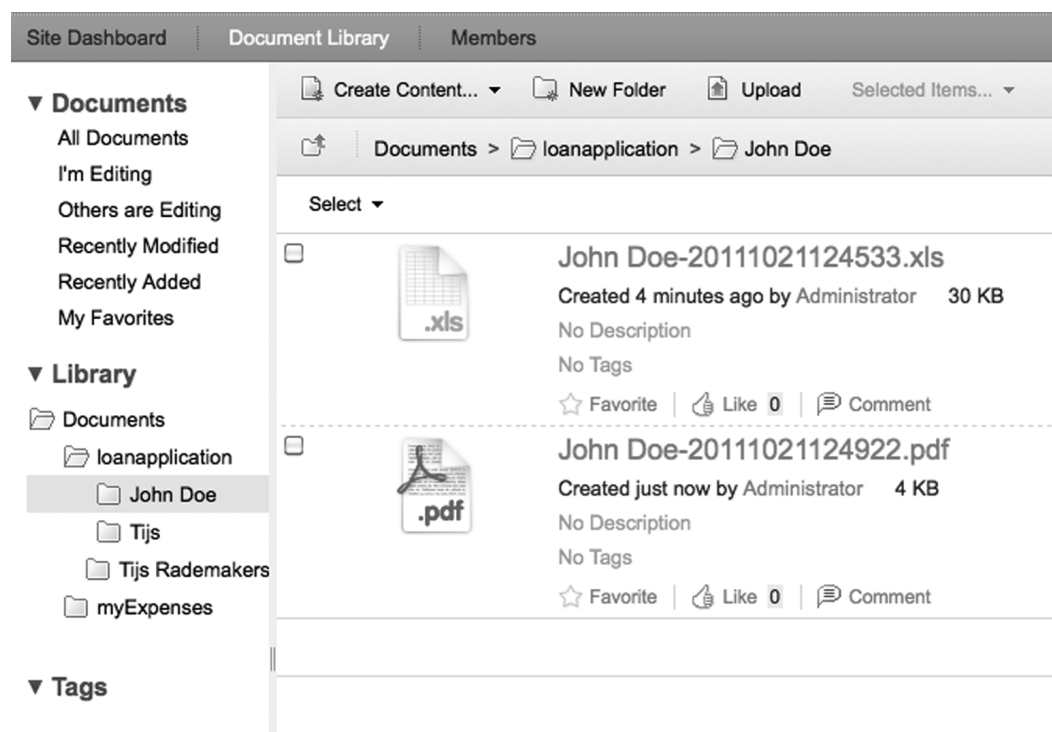


Figure 6 The Alfresco repository showing the Excel and PDF documents generated by the loan request process instance

You can again look at the contents of these files to make sure they are generated correctly. And you can start a couple more process instances to test all cases of the loan request process definition, including, for example, a denial.

## Summary

You implemented a process definition making heavy use of documents and the Alfresco repository. You saw how to utilize Excel documents and their powerful formulas to implement conditional logic. The logic implemented with Drools can easily be implemented using Excel formulas. In addition, you saw that a framework like iText can be used to generate a PDF document with a few lines of code.

**Here are some other Manning titles you might be interested in:**



[Open-Source ESBs in Action](#)

Tijs Rademakers and Jos Dirksen



[Open Source SOA](#)

Jeff Davis



[Mule in Action, Second Edition](#)

David Dossot, John D'Emic, and Victor Romero

Last updated: June 12, 2012

For Source Code, Sample Chapters, the Author Forum and other resources, go to  
<http://www.manning.com/rademakers2/>