

# Developing applications with XPAAJ

XML/PDF Access API for Java (XPAAJ) is a Java API that helps Java developers to build applications that work with PDF documents and forms. It can extract data from a PDF file programmatically and combine XML data with blank PDF forms. You can incorporate XPAAJ into standalone Java applications or deploy it into either a Web container or an application server container.

This tutorial outlines the process to follow when you use XPAAJ to develop applications that support PDF.

## Requirements

To make the most of this article, you need the following software and resources:

### **XPAAJ SDK**

[Download](#)

### **Additional documentation**

[XPAAJ topic page](#)

## Creating a form

An important fact to consider here is to determine the type of form you will be working with. You can create either an Acrobat form or an XML form. An Acrobat form is a PDF document created in Adobe Acrobat while an XML form is a PDF document created in LiveCycle Designer.

For this tutorial, you will be working with an XML form. Follow the Quick Start Tutorial 1 in the LiveCycle online help to create the Corporate Picnic Survey (see Figure 1).

## Corporate Picnic Survey

Help us make it a success

Name  How many will attend?

The park has given us the following dates. Please select the one that works best for you:

June 26  
 July 10  
 July 17

Select the activities that you would like to have at the picnic

Football  
 Volleyball  
 Face Painting  
 Sack Races

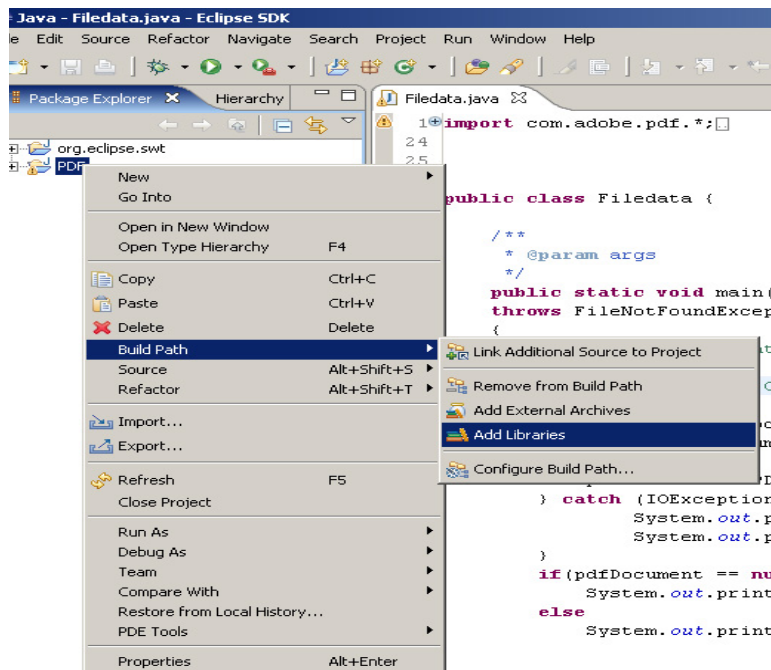
**Figure 1: The Corporate Picnic Survey sample form**

Once the Picnic Survey form is created, save it in the root of the C drive of your computer (C:\\practiceForm.pdf).

## Creating a Java programming environment

Use a suitable IDE such as Eclipse to create your Java application. Since XPAAJ is dependent on Java 2 Standard Edition 1.4.1 or later, you need to [download the J2SE Java Runtime Environment \(JRE\)](#), install it on your computer, and configure it.

Set the CLASSPATH environment variable by updating the environment settings to include the XPAAJ.jar class library, which is included in the XPAAJ SDK download (see Figure 2).



**Figure 2: Adding the XPAAJ.jar class library**

## Importing files and creating a PDFDocument object

Once the XPAAJ.jar file has been included in the library, include the following import statement to your Java project:

```
import com.adobe.pdf.*;
```

Most of the methods to access PDF/XDP files and their XML data are provided through the single public interface `PDFDocument`. Therefore, to access these methods, you first need to create a `PDFDocument` object, which will be used throughout the application.

The following is the sample code used to create a `PDFDocument` object and verify that the input file was successfully opened:

```

public static void main(String[] args)
throws FileNotFoundException, IOException, ParserConfigurationException
{
    // TODO Auto-generated method stub
    //this PDFDocument Object uses a filled pdf form and creates an xml
    file
    //with data tags.
    InputStream myPdfDoc = new FileInputStream("C:\\\\practiceForm.pdf");
    PDFDocument pdfDocument = null;
    try {
        pdfDocument = PDFFactory.openDocument(myPdfDoc);
    } catch (IOException e) {
        System.out.println("Error opening PDF file :" + myPdfDoc);
        System.out.println(e);
    }
}

```

```

}
if(pdfDocument == null)
System.out.println("Cannot open PDF file : " + myPdfDoc);
else
System.out.println( myPdfDoc + " was successfully opened.");

```

**Note:** You can find details regarding creating a `PDFDocument` object in the “Invoking XML/PDF Access API for Java” section of the Developer Guide in the XPAAJ SDK download.

For a list of important import statements that you can use while performing operations using the XPAAJ, refer to the Resources section toward the end of this document.

## Extracting PDF form details

Using the `PDFDocument` object methods, you can extract some details about the saved PDF form (C:\practiceForm.pdf). These details include the type of the form, the version of the form, and the number of pages in the form. You can find detailed instructions on how to extract these details in the Developer Guide. The following is the sample code written to extract the details.

The following code returns the type of the PDF form:

```

// TYPE OF DOCUMENT
FormType docType = pdfDocument.getFormType();
if (docType == FormType.ACROFORM)
System.out.println("This document is based on an ACROFORM.");
else if (docType == FormType.XML_FORM)
System.out.println("This document is based on an XML_FORM.");
else
System.out.println("This document is not a form.");

```

The following code prints the version of the PDF form being used:

```

//DOCUMENT VERSION
String docVersion = pdfDocument.getVersion();
System.out.println("The version of the PDF document is "
+docVersion+".");

```

The following code returns the number of pages in the PDF form:

```

//GETTING THE NUMBER OF PAGES IN THE DOCUMENT.
int numPages = pdfDocument.getNumberOfPages();
System.out.println("The PDF document contains "+ numPages +
pages(s).");

```

## Exporting form data and saving it as an XML file

To extract data from your saved Picnic Survey form and output it in the form of an XML file, first fill in the form and save the form as `practice_Form.pdf`.

In your Java program, do the following to export the data to an XML file for the output:

1. Create a `PDFDocument` object as described earlier in this tutorial and provide `practice_Form` as the input.
2. Create a Java `InputStream` object and populate it by calling the `PDFDocument` object's `exportFormData` method. Pass `FormDataFormat.XFA` as an argument.
3. Follow the steps described in the “Extracting Form Data” section of the Developer Guide in the XPAAJ SDK download.
4. If necessary, use the sample code in the `PDFExtractData.java` file included in the XPAAJ SDK download for help.

Figure 3 below is the sample output in the form of an XML file. The tags contain the information that the form was filled with.

```
<?xml version="1.0" encoding="UTF-8" ?>
- <form1>
- <Name>
  <myName>Shoaib Nursumar</myName>
  <Comments>This Trip sounds fun. i Think Charles would like to come. I hope HE'S not busy with the eBay project.</Comments>
  <attendees>5</attendees>
  <PreferredDate>2</PreferredDate>
  <Football>1</Football>
  <Volleyball>0</Volleyball>
  <FacePainting>0</FacePainting>
  <SackRaces>1</SackRaces>
</Name>
</form1>
```

**Figure 3: Exporting form data as an XML file**

To get a better understanding of this output, read through the Loan Application example in the “Importing and Exporting Form Data” section of the Developer Guide in the XPAAJ SDK download.

## Prepopulating forms

You can use the XPAAJ to prepopulate a form with data to reduce work for the user. There are three different ways to prepopulate a form:

- Converting an XML string to a byte stream
- Referencing an existing XML document
- Dynamically creating an in-memory XML data structure

You could use the same Picnic Survey form as an example, but make sure that the form does not contain any information. Then save a copy of the blank form as `blank_Form.pdf` in your C drive.

The following steps demonstrate how to prepopulate the saved form by converting an XML string to a byte stream.

1. Create a `PDFDocument` object and pass the saved `blank_Form.pdf` as the input data
2. Create a Java String object and assign it a value that resembles an XML schema that corresponds to the form fields you want to prepopulate.
3. Assign the Java String object to a byte array and call its `getBytes` method.
4. Create a Java `InputStream` object by using the `ByteArrayInputStream` constructor. Pass the byte array to the constructor.
5. Call the `PDFDocument` object's `importFormData` method and pass the `InputStream` object.
6. Save the PDF document. For information, see [“Saving PDF documents”](#) on page 27 of the Developer Guide in the XPAAJ SDK download.

Figure 4 below illustrates the code you need to write to prepopulate the form.

```
//Prepopulating a form by converting a string variable to a byte stream
String formData = "<root><myName>Shoaib Nursumar</myName>" +
    "<Comments>Last year we had a lot of fun on this trip. " +
    "I am still looking forward for this one too</Comments>" +
    "<attendees>4</attendees><PreferredDate>2</PreferredDate>" +
    "<Football>1</Football><Volleyball>0</Volleyball>" +
    "<FacePainting>0</FacePainting><SackRaces>1</SackRaces></root>";
    Convert the string variable into a byte array
    byte[] inputData = formData.getBytes("UTF8");
    Create an InputStream by calling the ByteArrayInputStream constructor
    InputStream formInputStream = new ByteArrayInputStream(inputData);
    Call the PDFDocument object's importFormData method
    blankPdf.importFormData(formInputStream);
//saving the data
formInputStream = blankPdf.save();
int numBytes = formInputStream.available();
byte [] PDFBytes = new byte[numBytes];
formInputStream.read(PDFBytes);
File myFile = new File("C:\\Test.pdf");
FileOutputStream myFileW = new FileOutputStream(myFile);
myFileW.write(PDFBytes);
myFileW.close();
```

**Figure 4: Prepopulating forms**

## Resources

The following is a list of important import statements that you can use while writing code to perform operations using the XPAAJ:

```
import com.adobe.pdf.*; //To create PDFDocument objects and PDFDocument
    method calls.
import java.io.InputStream; //To create InputStream objects
import java.io.FileInputStream; //To create a FileInputStream object
import java.io.IOException; //To handle Exceptions in the code
import java.io.FileNotFoundException; //To handle a FileNotFoundException
import java.io.ByteArrayInputStream; //To create a ByteArrayInputStream
    object
```

```
import java.io.FileOutputStream; //To create a FileOutputStream object
```

The following is a list of import statements that you need when building an XML data structure and extracting data from a PDF file:

```
import org.w3c.dom.Document;  
import org.w3c.dom.NodeList;  
import org.w3c.dom.Node;  
import org.w3c.dom.Element;  
import javax.xml.parsers.*;  
import javax.xml.transform.*;  
import javax.xml.transform.dom.DOMSource;  
import javax.xml.transform.stream.StreamResult;
```

Details regarding the above mentioned import statements can be found in the API Reference Guide in the XPAAJ SDK download.