PDFBox User Guide

This page will discuss the internals of PDF documents and how those internals map to PDFBox classes. Users should reference the javadoc to see what classes and methods are available. The Adobe PDF Reference can be used to determine detailed information about fields and their meanings.

Examples

A variety of examples can be found in the src/main/java/org/apache/pdfbox/examples folder. This guide will refer to specific examples as needed.

PDF File Format Overview

A PDF document is a stream of basic object types. The low level objects are represented in PDFBox in the *org.apache.pdfbox.cos* package. The basic types in a PDF are:

PDF Type	**Description**	**Example**	**PDFBox class**
Array	An ordered list of items	[1 2 3]	org.apache.pdfbox.cos.COSArray
Boolean	Standard True/False values	true	org.apache.pdfbox.cos.COSBoolean
Dictionary	A map of name value pairs	<< /Type /XObject /Name (Name) /Size 1 >>	org.apache.pdfbox.cos.COSDictionary
Number	Integer and Floating point numbers	1 2.3	org.apache.pdfbox.cos.COSFloat org.apache.pdfbox.cos.COSInteger
Name	A predefined value in a PDF document, typically used as a key in a dictionary	/Type	org.apache.pdfbox.cos.COSName
Object	A wrapper to any of the other objects, this can be used to reference an object multiple times. An object is referenced by using two numbers, an object number and a generation number. Initially the generation number will be zero unless the object got replaced later in the stream.	12 0 obj << /Type /XObject >> endobj	org.apache.pdfbox.cos.COSObject
Stream	A stream of data, typically compressed. This is used for page contents, images and embedded font streams.	12 0 obj << /Type /XObject >> stream 030004040404040404 endstream	org.apache.pdfbox.cos.COSStream
String	A sequence of characters	(This is a string)	org.apache.pdfbox.cos.COSString

A page in a pdf document is represented with a COSDictionary. The entries that are available for a page can be seen in the PDF Reference and an example of a page looks like this:

```
<<
   /Type /Page
   /MediaBox [0 0 612 915]
   /Contents 56 0 R
>>
```

Some Java code to access fields

```
COSDictionary page = ...;
COSArray mediaBox = (COSArray)page.getDictionaryObject( "MediaBox" );
```

```
System.out.println( "Width:" + mediaBox.get( 3 ) );
```

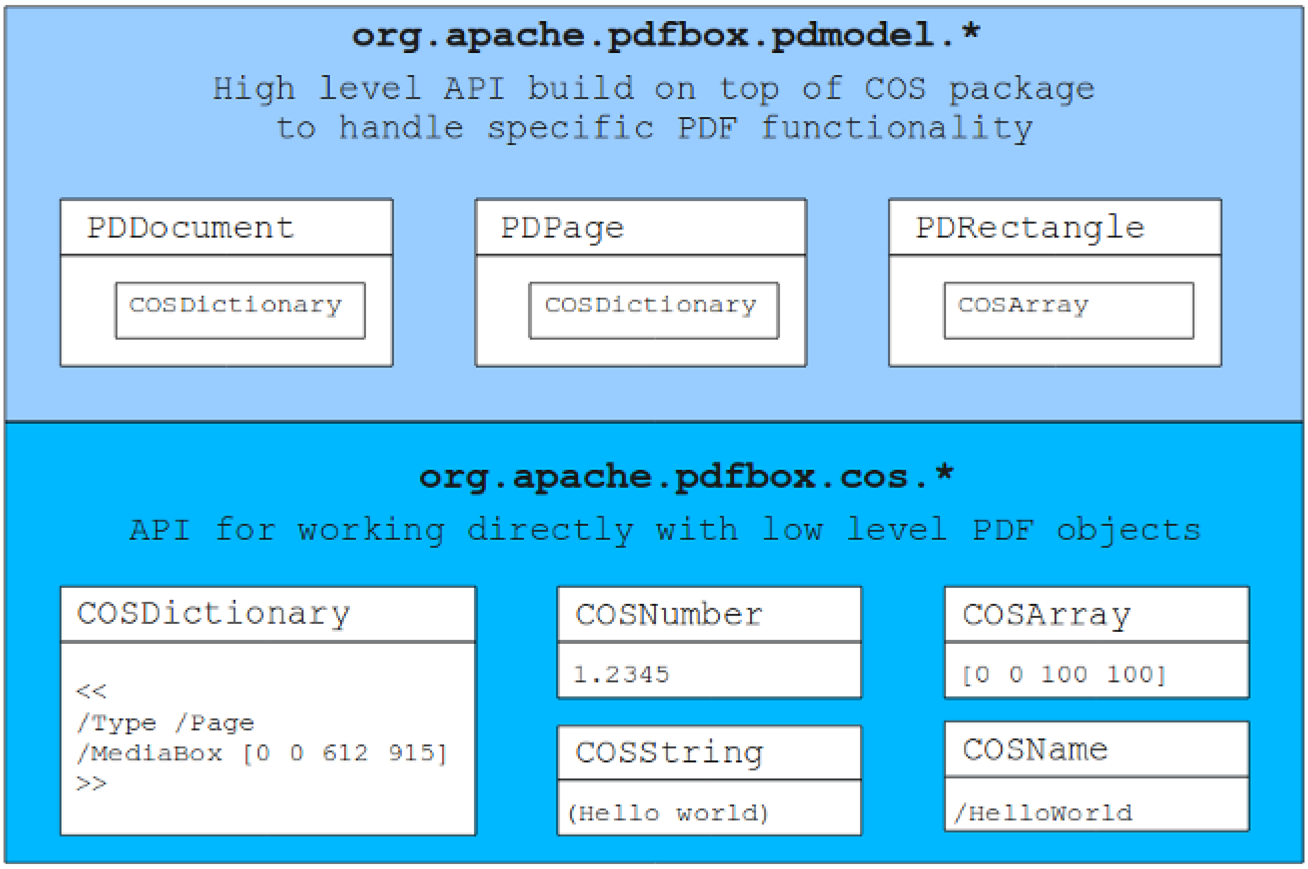
PD Model

The COS Model allows access to all aspects of a PDF document. This type of programming is tedious and error prone though because the user must know all of the names of the parameters and no helper methods are available. The PD Model was created to help alleviate this problem. Each type of object(page, font, image) has a set of defined attributes that can be available in the dictionary. A PD Model class is available for each of these so that strongly typed methods are available to access the attributes. The same code from above to get the page width can be rewritten to use PD Model classes.

```
PDPage page = ...;
PDRectangle mediaBox = page.getMediaBox();
System.out.println( "Width:" + mediaBox.getWidth() );
```

PD Model objects sit on top of COS model. Typically, the classes in the PD Model will only store a COS object and all setter/getter methods will modify data that is stored in the COS object. For example, when you call PDPage.getLastModified() the method will do a lookup in the COSDictionary with the key "LastModified", if it is found the value is then converter to a java.util.Calendar. When PDPage.setLastModified(Calendar) is called then the Calendar is converted to a string in the COSDictionary.

Here is a visual depiction of the COS Model and PD Model design.


org.apache.pdfbox.pdmodel.*
High level API build on top of COS package to handle specific PDF functionality
PDDocument
COSDictionary
PDPage
COSDictionary
PDRectangle
COSArray
org.apache.pdfbox.cos.*
API for working directly with low level PDF objects
COSDictionary
<< /Type /Page /MediaBox [0 0 612 915] >>
COSNumber
1.2345
COSArray
[0 0 100 100]
COSString
(Hello world)
COSName
/HelloWorld

This design presents many advantages and disadvantages.

Advantages:

- Simple, easy to use API.
- Underlying document automatically gets updated when you update the PD Model
- Ability to easily access the COS Model from any PD Model object
- Easily add to and update existing PDF documents

Disadvantages:

- Object caching is not done in the PD Model classes For example, each call to PDPage.getMediaBox() will return a new PDRectangle object, but will contain the same underlying COSArray.

Accessing PDF Bookmarks

See package:org.apache.pdfbox.pdmodel.interactive.documentnavigation.outline
See example:PrintBookmarks

A PDF can contain an outline of a document and jump to pages within a PDF document. An outline is a hierarchical tree structure of nodes that point to pages.

To access the root of the outline you go through the PDDocumentOutline

```
PDDocument doc = PDDocument.load( ... );
PDDocumentOutline root = doc.getDocumentCatalog().getDocumentOutline();
```

Now you can traverse the tree using the getFirstChild() and getNextSibling() functions.

```
PDOutlineItem item = root.getFirstChild();
while( item != null )
{
    System.out.println( "Item:" + item.getTitle() );
    PDOutlineItem child = item.getFirstChild();
    while( child != null )
    {
        System.out.println( "    Child:" + child.getTitle() );
        child = child.getNextSibling();
    }
    item = item.getNextSibling();
}
```

Creating Bookmarks

See example:CreateBookmarks

Creating bookmarks is just as easy. You first need to create the PDDocumentOutline and then add some PDOutlineItem objects to it.

```
//first create the document outline and add it to the page
PDDocumentOutline outline = new PDDocumentOutline();
doc.getDocumentCatalog().setDocumentOutline( outline );

//Create a root element to show in the tree
PDOutlineItem root = new PDOutlineItem();
root.setTitle( "Root of Document" );
outline.appendChild( root)

//Get the page to refer to
PDPage firstPage = (PDPage)doc.getAllPages().get( 0 );

//Create the outline item to refer to the first page.
PDOutlineItem firstPageItem = new PDOutlineItem();
firstPageItem.setTitle( "First Page of document" );
firstPageItem.setDestination( firstPage );
root.appendChild( firstPageItem );
```

NOTE: By default all nodes in the outline tree are closed. You need to call openNode() if you want the node to be open when the document is opened.

Building PDFBox

This page will describe how to build PDFBox. Most users should find the binary releases adequate, but if you are contributing to PDFBox then you will need to know how to properly build the project.

You can obtain the latest source of PDFBox from SVN, see the download page for information about how to connect to SVN. The instructions below should also work for a released/nightly distribution.

Starting with PDFBox 1.0.0, the default build system is based on Maven:

1. Install Maven 2
2. Run "mvn clean install" from the PDFBox root

The old Ant build is still available, and can be used especially for building .NET binaries with IKVM:

1. Install ANT . PDFBox currently uses 1.6.2 but other versions probably work as well.
2. (optional) Setup IKVM, if you want to build the .NET DLL version of PDFBox.
    1. IKVM binaries
    2. In the build.properties, set the ikvm.dir property: ikvm.dir=C:\\javalib\\ikvm-12-07-2004\\ikvm
3. Run "ant" from the root PDFBox directory. This will create the .zip package distribution. See the build file for other ant targets.

NOTE: If you want to run PDFBox from an IDE them you will need to add the 'Resources' directory to the project classpath in your IDE.

Running / Debugging PDFBox in Eclipse

In order to run PDFBox from within Eclipse there are a couple things that need to be setup. Certain pieces of functionality require access to the Resources folder. Here is an example of what you need to do to run the ExtractText command line application.

1. In Eclipse click on "run..." or "debug..."
2. In the tree on the left select "Java Application" and click "New"
3. Name it, select the correct project, and the main class
4. Select the "Arguments" tab and enter the command line arguments to the app
5. Select the "Classpath" tab, the pdfbox project and jars should already be added in the "User Entries"
6. Select the "User Entries" item and click the "Advanced" button
7. The "Add Folder" radio button should be selected, press ok
8. Select the root of the pdfbox project, this will make the "Resources" folder available on the classpath.

Tutorials

The Tutorials for PDFBox provide a quick start into some of the most common uses cases for PDFBox as part of a PDF application. In addition the Cookbook provides a wide range of samples including sample code.

Tutorial	Description
Bookmarks	PDF Bookmarks allow for a quick navigation within documents similar to a Table of Contents. The Tutorial covers how to read bookmarks as well as how to generate them.
File References	A PDF file can contain references to external files as well as files embedded in the PDF itself. The Tutorial covers how handle such references as well as how to embed a file in a PDF document.
Highlighting	PDF allows for text to be highlighted. This can be useful for example to highlight results of a search. There are different ways to approach this covered in the tutorial.
Metadata	PDF documents can have metadata associated with the document or certain objects within the document. As an example metadata can be used to store the author of a document as well as copyright information for an image embedded in the document. The Tutorial covers how to read and generate such metadata.
Text Extraction	One of the main features of PDFBox is its ability to quickly and accurately extract text from a variety of PDF documents. The tutorial covers different approaches to handle that task.

Accessing PDF Bookmarks

See package:org.apache.pdfbox.pdmodel.interactive.documentnavigation.outline
See example:PrintBookmarks

A PDF can contain an outline of a document and jump to pages within a PDF document. An outline is a hierarchical tree structure of nodes that point to pages.

To access the root of the outline you go through the PDDocumentOutline

```
PDDocument doc = PDDocument.load( ... );
PDDocumentOutline root = doc.getDocumentCatalog().getDocumentOutline();
```

Now you can traverse the tree using the getFirstChild() and getNextSibling() functions.

```
PDOutlineItem item = root.getFirstChild();
while( item != null )
{
    System.out.println( "Item:" + item.getTitle() );
    PDOutlineItem child = item.getFirstChild();
    while( child != null )
    {
        System.out.println( "    Child:" + child.getTitle() );
        child = child.getNextSibling();
    }
    item = item.getNextSibling();
}
```

Creating Bookmarks

See example:CreateBookmarks

Creating bookmarks is just as easy. You first need to create the PDDocumentOutline and then add some PDOutlineItem objects to it.

```
//first create the document outline and add it to the page
PDDocumentOutline outline = new PDDocumentOutline();
doc.getDocumentCatalog().setDocumentOutline( outline );

//Create a root element to show in the tree
PDOutlineItem root = new PDOutlineItem();
root.setTitle( "Root of Document" );
outline.appendChild( root)

//Get the page to refer to
PDPage firstPage = (PDPage)doc.getAllPages().get( 0 );

//Create the outline item to refer to the first page.
PDOutlineItem firstPageItem = new PDOutlineItem();
firstPageItem.setTitle( "First Page of document" );
firstPageItem.setDestination( firstPage );
root.appendChild( firstPageItem );
```

NOTE: By default all nodes in the outline tree are closed. You need to call openNode() if you want the node to be open when the document is opened.

PDF File Specification

See package:org.apache.pdfbox.pdmodel.common.filespecification
See example:EmbeddedFiles

A PDF can contain references to external files via the file system or a URL to a remote location. It is also possible to embed a binary file into a PDF document.

There are two classes that can be used when referencing a file. PDSimpleFileSpecification is a simple string reference to a file(e.g. "./movies/BigMovie.avi"). The simple file specification does not allow for any parameters to be set. The PDComplexFileSpecification is more feature rich and allows for advanced settings on the file reference.

It is also possible to embed a file directly into a PDF. Instead of setting the file attribute of the PDComplexFileSpecification, the EmbeddedFile attribute can be used instead.

File Attachments

PDF documents can contain file attachments that are accessed from the Document->File Attachments menu. PDFBox allows attachments to be added to and extracted from PDF documents. Attachments are part of the named tree that is attached to the document catalog.

```
PDEmbeddedFilesNameTreeNode efTree = new PDEmbeddedFilesNameTreeNode();

//first create the file specification, which holds the embedded file
PDComplexFileSpecification fs = new PDComplexFileSpecification();
fs.setFile( "Test.txt" );
InputStream is = ...;
PDEmbeddedFile ef = new PDEmbeddedFile(doc, is );
//set some of the attributes of the embedded file
ef.setSubtype( "test/plain" );
ef.setSize( data.length );
ef.setCreationDate( new GregorianCalendar() );
fs.setEmbeddedFile( ef );

//now add the entry to the embedded file tree and set in the document.
Map efMap = new HashMap();
efMap.put( "My first attachment", fs );
efTree.setNames( efMap );
//attachments are stored as part of the "names" dictionary in the document catalog
PDDocumentNameDictionary names = new PDDocumentNameDictionary( doc.getDocumentCatalo
names.setEmbeddedFiles( efTree );
doc.getDocumentCatalog().setNames( names );
```

Highlighting text in a PDF

There are cases when you might want to highlight text in a PDF document. For example, if the PDF is the result of a search request you might want to highlight the word in the resulting PDF document. There are several ways this can be achieved, each method varying in complexity and flexibility.

1. Use the 'search' open parameter

Acrobat supports passing is various parameters that tell it what to do once the PDF is open. See PDF Open Parameters for documentation on all the open parameters. One of the parameters is the 'search' parameter, this will automatically run the search functionality inside of Acrobat once the PDF is open. For example: http://pdfbox.apache.org/userguide/text_extraction.pdf#search="check"

NOTE: The words must be enclosed in quotes and separated by spaces; for example:#search="pdfbox rocks"

This is a great solution because of its simplicity! It doesn't require PDFBox at all, but it is a potential solution that many developers are not aware of.

2. Generate a highlight XML document

Acrobat also allows you to tell it to highlight specific words in the PDF document. It does this by passing an XML document to Acrobat when opening the PDF. See the PDF Highlight File Format for more detailed documentation.

Basically the document allows you to tell it the characters to highlight in the PDF by using character offsets on a page. As this is just an XML document, there are many ways you could create it but PDFBox does have a utility to make it easier. Take a look at the javadoc for the PDFHighlighter class. This will allow you specify a set of words that you want have highlighted and generate the XML document for you.

PDFBox also ships with a complete web application example of using this class, take a look at the pdfbox.war directory in your PDFBox installation.

You pass the xml to acrobat through a URL (or command line) parameter like this: http://pdfbox.apache.org/userguide/text_extraction.pdf#xml=http://pdfbox.apache.org/highlight.xml

NOTE: The value of the xml parameter must be a full URL to the XML document.
http://pdfbox.apache.org/userguide/text_extraction.pdf#xml=highlight.xml will not work
http://pdfbox.apache.org/userguide/text_extraction.pdf#xml=http://pdfbox.apache.org/highlight.xml is correct!

The one drawback to this solution is that you must parse the PDF and then generate an XML document, which is a time consuming operation.

3. Alter pdf contents to highlight specific text

Using PDFBox it is possible to regenerate the appearance stream to add highlighting to specific areas. While this is possible, it will require recreating a new PDF for every search request. There is nothing prebuilt in PDFBox to do this automatically for you and will require a significant coding effort.

You would need to

1. Find all locations of the text, determine x/y coordinates, width/height
2. Regenerate the PDF appearance stream and draw a highlighted box behind the text. Yellow would be easiest, if you want an inverted black/white, then you would need to change the color of the text to be white and draw a black box.
3. Stream the PDF back to the user

This is the most flexible but is also the most work to implement and is also more resource intensive.

Introduction

PDF documents can contain information describing the document itself or certain objects within the document such as the author of the document or it's creation date. Basic information can be set and retrieved using the PDDocumentInformation object.

In addition to that more metadata can be retrieved using the XML metadata as decribed below.

Getting basic Metadata

To set or retrieve basic information about the document the PDDocumentInformation object provides a high level API to that information:

```
PDDocumentInformation info = document.getDocumentInformation();
System.out.println( "Page Count=" + document.getNumberOfPages() );
System.out.println( "Title=" + info.getTitle() );
System.out.println( "Author=" + info.getAuthor() );
System.out.println( "Subject=" + info.getSubject() );
System.out.println( "Keywords=" + info.getKeywords() );
System.out.println( "Creator=" + info.getCreator() );
System.out.println( "Producer=" + info.getProducer() );
System.out.println( "Creation Date=" + info.getCreationDate() );
System.out.println( "Modification Date=" + info.getModificationDate());
System.out.println( "Trapped=" + info.getTrapped() );
```

Accessing PDF Metadata

See class:org.apache.pdfbox.pdmodel.common.PDMetadata
See example:AddMetadataFromDocInfo
See Adobe Documentation:XMP Specification

PDF documents can have XML metadata associated with certain objects within a PDF document. For example, the following PD Model objects have the ability to contain metadata:

- PDDocumentCatalog
- PDPage
- PDXObject
- PDICCBased
- PDStream

The metadata that is stored in PDF objects conforms to the XMP specification, it is recommended that you review that specification. Currently there is no high level API for managing the XML metadata, PDFBox uses standard java InputStream/OutputStream to retrieve or set the XML metadata. For example:

```
PDDocument doc = PDDocument.load( ... );
PDDocumentCatalog catalog = doc.getDocumentCatalog();
PDMetadata metadata = catalog.getMetadata();

//to read the XML metadata
InputStream xmlInputStream = metadata.createInputStream();

//or to write new XML metadata
InputStream newXMPData = ...;
PDMetadata newMetadata = new PDMetadata(doc, newXMLData, false );
catalog.setMetadata( newMetadata );
```

Extracting Text

See class:org.apache.pdfbox.util.PDFTextStripper
See class:org.apache.pdfbox.searchengine.lucene.LucenePDFDocument
See command line app:ExtractText

One of the main features of PDFBox is its ability to quickly and accurately extract text from a variety of PDF documents. This functionality is encapsulated in the org.apache.pdfbox.util.PDFTextStripper and can be easily executed on the command line with org.apache.pdfbox.ExtractText.

Lucene Integration

Lucene is an open source text search library from the Apache Jakarta Project. In order for Lucene to be able to index a PDF document it must first be converted to text. PDFBox provides a simple approach for adding PDF documents into a Lucene index.

```
Document luceneDocument = LucenePDFDocument.getDocument( ... );
```

Now that you hava a Lucene Document object, you can add it to the Lucene index just like you would if it had been created from a text or HTML file. The LucenePDFDocument automatically extracts a variety of metadata fields from the PDF to be added to the index, the javadoc shows details on those fields. This approach is very simple and should be sufficient for most users, if not then you can use some of the advanced text extraction techniques described in the next section.

Advanced Text Extraction

Some applications will have complex text extraction requiments and neither the command line application nor the LucenePDFDocument will be able to fulfill those requirements. It is possible for users to utilize or extend the PDFTextStripper class to meet some of these requirements.

Limiting The Extracted Text

There are several ways that we can limit the text that is extracted during the extraction process. The simplest is to specify the range of pages that you want to be extracted. For example, to only extract text from the second and third pages of the PDF document you could do this:

```
PDFTextStripper stripper = new PDFTextStripper();
stripper.setStartPage( 2 );
stripper.setEndPage( 3 );
stripper.writeText( ... );
```

NOTE: The startPage and endPage properties of PDFTextStripper are 1 based and inclusive.

If you wanted to start on page 2 and extract to the end of the document then you would just set the startPage property. By default all pages in the pdf document are extracted.

It is also possible to limit the extracted text to be between two bookmarks in the page. If you are not familiar with how to use bookmarks in PDFBox then you should review the Bookmarks page. Similar to the startPage/endPage properties, PDFTextStripper also has startBookmark/endBookmark properties. There are some caveats to be aware of when using this feature of the PDFTextStripper. Not all bookmarks point to a page in the current PDF document. The possible states of a bookmark are:

- null - The property was not set, this is the default.
- Points to page in the PDF - The property was set and points to a valid page in the PDF
- Bookmark does not point to anything - The property was set but the bookmark does not point to any page

- Bookmark points to external action - The property was set, but it points to a page in a different PDF or performs an action when activated

The table below will describe how PDFBox behaves in the various scenarios:

Start Bookmark	**End Bookmark**	**Result**
null	null	This is the default, the properties have no effect on the text extraction.
Points page in the PDF	null	Text extraction will begin on the page that this bookmark points to and go until the end of the document.
null	Points page in the PDF	Text extraction will begin on the first page and stop at the end of the page that this bookmark points to.
Bookmark does not point to anything	null	Because the PDFTextStripper cannot determine a start page based on the bookmark, it will start on the first page and go until the end of the document.
null	Bookmark does not point to anything	Because the PDFTextStripper cannot determine a end page based on the bookmark, it will start on the first page and go until the end of the document.
Bookmark does not point to anything	Bookmark does not point to anything	This is a special case! If the startBookmark and endBookmark are exactly the same then no text will be extracted. If they are different then it is not possible for the PDFTextStripper to determine that pages so it will include the entire document.
Bookmark points to external action	Bookmark points to external action	If either the startBookmark or the endBookmark refer to an external page or execute an action then an OutlineNotLocalException will be thrown to indicate to the user that the bookmark is not valid.

NOTE: PDFTextStripper will check both the startPage/endPage and the startBookmark/endBookmark to determine if text should be extracted from the current page.

External Glyph List

Some PDF files need to map between glyph names and Unicode values during text extraction. PDFBox comes with an Adobe Glyph List , but you may encounter files with glyph names that are not in that map. To use your own glyphlist file, supply the file name to the glyphlist_ext JVM property.

Right to Left Text

Extracting text in languages whose text goes from right to left (such as Arabic and Hebrew) in PDF files can result in text that is backwards. PDFBox can normalize and reverse the text if the ICU4J jar file has been placed on the classpath (it is an optional dependency). Note that you should also enable sorting with either org.apache.pdfbox.util.PDFTextStripper or org.apache.pdfbox.ExtractText to ensure accurate output.

Cookbook

The Cookbook for PDFBox is a collection of source code samples to help using PDFBox. The samples are a growing collection of individual topics covering a wide range of PDF applications. In addition the Tutorials cover some of the most common applications of PDFBox.

Document creation

Sample	Description
CreateBlankPDF	This small sample shows how to create a new PDF document using PDFBox.
HelloWorld	This small sample shows how to create a new document and print the text "Hello World" using one of the PDF base fonts.
HelloWorldTTF	This small sample shows how to create a new document and print the text "Hello World" using a TrueType Font.
HelloWorldType1AfmPfb	This is an example that creates a simple document with a Type 1 font (afm + pfb).
ImageToPDF	This is an example that creates a simple document from an image.
ShowColorBoxes	This is an example that creates a simple document with different boxes.

Working with metadata

Sample	Description
PrintDocumentMetaData	This is an example on how to get a documents metadata information.
AddMetadataFromDocInfo	This is an example on how to add metadata to a document.
ExtractMetadata	This is an example on how to extract metadata from a PDF document.

Dealing with forms

Sample	Description
PrintFields	Shows how to print all the fields from a PDF document
SetField	Shows how to set the value of a form field in a PDF document

Using the PDModel

Sample	Description
AddImageToPDF	This is an example that reads a PDF document and adds an image to it.
AddJavascript	This is an example of how to set some JavaScript in the document.
AddMessageToEachPage	This is an example of how to add a message to every page in a PDF document.
Annotation	This is an example on how to add annotations to pages of a PDF document.
CreateBookmarks	This is an example on how to add bookmarks to a PDF document.
EmbeddedFiles	This is an example that creates a simple document and embeds a file into it..
GoToSecondBookmarkOnOpen	This is an example on how to an action to go to the second page when the PDF is opened.
PrintBookmarks	This is an example on how to access the bookmarks that are part of a pdf document.
PrintURLs	This is an example of how to access a URL in a PDF document.
RemoveFirstPage	This is an example on how to remove pages from a PDF document.
ReplaceString	This is an example that will replace a string in a PDF with a new one.
ReplaceURLs	This is an example of how to replace a URL in a PDF document.
RubberStamp	This is an example on how to add annotations to pages of a PDF document.

RubberStampWithImage	This is an example on how to add a rubber stamp with an image to pages of a PDF document.
UsingTextMatrix	This is an example of how to use a text matrix.

PDFBox persistence features

Sample	**Description**
CopyDoc	This is an example used to copy a documents contents from a source doc to destination doc via an in-memory document representation
PrintFields	Shows how to load a PDF document and write with all streams decoded.

Working with signatures

Sample	**Description**
ShowSignature	This examples will show how to gain access to the PDF signature

Text and Image locations

Sample	**Description**
ExtractTextByArea	This is an example on how to extract text from a specific area on the PDF document.
PrintImageLocations	This is an example on how to get the x/y coordinates of image locations.
PrintTextLocations	This is an example on how to get some x/y coordinates of text.
RemoveAllText	This is an example on how to remove all text from PDF document.

Frequently Asked Questions

General Questions

Text Extraction

General Questions

When will the next version of PDFBox be released?

As fixes are made and integrated into the repository these changes are documented in the release notes. An estimate will be given of when the next version will be released.
Of course, this is only an estimate and could change.

I am getting the below Log4J warning message, how do I remove it?

```
log4j:WARN No appenders could be found for logger (org.apache.pdfbox.util.ResourceL
log4j:WARN Please initialize the log4j system properly.
```

This message means that you need to configure the log4j logging system. See the log4j documentation for more information.

PDFBox comes with a sample log4j configuration file. To use it you set a system property like this

```
java -Dlog4j.configuration=log4j.xml org.apache.pdfbox.ExtractText <PDF-file> <output-text-file>
```

If this is not working for you then you may have to specify the log4j config file using a URL path, like this:

```
log4j.configuration=file:///<path to config file>
```

Please see this forum thread for more information.

Is PDFBox thread safe?

No! Only one thread may access a single document at a time. You can have multiple threads each accessing their own PDDocument object.

Why do I get a "Warning: You did not close the PDF Document"?

You need to call close() on the PDDocument inside the finally block, if you don't then the document will not be closed properly. Also, you must close all PDDocument objects that get created. The following code creates **two** PDDocument objects; one from the "new PDDocument()" and the second by the load method.

```
PDDocument doc = new PDDocument();
try
{
   doc = PDDocument.load( "my.pdf" );
}
finally
{
   if( doc != null )
   {
      doc.close();
   }
}
```

Text Extraction

How come I am not getting any text from the PDF document?

Text extraction from a pdf document is a complicated task and there are many factors involved that effect the possibility and accuracy of text extraction. It would be helpful to the PDFBox team if you could try a couple things.

- Open the PDF in Acrobat and try to extract text from there. If Acrobat can extract text then PDFBox should be able to as well and it is a bug if it cannot. If Acrobat cannot extract text then PDFBox 'probably' cannot either.
- It might really be an image instead of text. Some PDF documents are just images that have been scanned in. You can tell by using the selection tool in Acrobat, if you can't select any text then it is probably an image.

How come I am getting gibberish(G38G43G36G51G5) when extracting text?

This is because the characters in a PDF document can use a custom encoding instead of unicode or ASCII. When you see gibberish text then it probably means that a meaningless internal encoding is being used. The only way to access the text is to use OCR. This may be a future enhancement.

What does "java.io.IOException: Can't handle font width" mean?

This probably means that the "Resources" directory is not in your classpath. The Resources directory is included in the PDFBox jar so this is only a problem if you are building PDFBox yourself and not using the binary.

Why do I get "You do not have permission to extract text" on some documents?

PDF documents have certain security permissions that can be applied to them and two passwords associated with them, a user password and a master password. If the "cannot extract text" permission bit is set then you need to decrypt the document with the master password in order to extract the text.

Can't we just extract the text without parsing the whole document or extract text as it is parsed.

Not really, for a couple reasons.

1. If the document is encrypted then you need to parse at least until the encryption dictionary before you can decrypt.
2. Sometimes the PDFont contains vital information needed for text extraction.
3. Text on a page does not have to be drawn in reading order. For example; if the page said "Hello World", the pdf could have been written such that "World" gets drawn and then the cursor moves to the left and the word "Hello" is drawn.

PDF File Specification

See package:org.apache.pdfbox.pdmodel.common.filespecification
See example:EmbeddedFiles

A PDF can contain references to external files via the file system or a URL to a remote location. It is also possible to embed a binary file into a PDF document.

There are two classes that can be used when referencing a file. PDSimpleFileSpecification is a simple string reference to a file(e.g. "./movies/BigMovie.avi"). The simple file specification does not allow for any parameters to be set. The PDComplexFileSpecification is more feature rich and allows for advanced settings on the file reference.

It is also possible to embed a file directly into a PDF. Instead of setting the file attribute of the PDComplexFileSpecification, the EmbeddedFile attribute can be used instead.

File Attachments

PDF documents can contain file attachments that are accessed from the Document->File Attachments menu. PDFBox allows attachments to be added to and extracted from PDF documents. Attachments are part of the named tree that is attached to the document catalog.

```
PDEmbeddedFilesNameTreeNode efTree = new PDEmbeddedFilesNameTreeNode();

//first create the file specification, which holds the embedded file
PDComplexFileSpecification fs = new PDComplexFileSpecification();
fs.setFile( "Test.txt" );
InputStream is = ...;
PDEmbeddedFile ef = new PDEmbeddedFile(doc, is );
//set some of the attributes of the embedded file
ef.setSubtype( "test/plain" );
ef.setSize( data.length );
ef.setCreationDate( new GregorianCalendar() );
fs.setEmbeddedFile( ef );

//now add the entry to the embedded file tree and set in the document.
Map efMap = new HashMap();
efMap.put( "My first attachment", fs );
efTree.setNames( efMap );
//attachments are stored as part of the "names" dictionary in the document catalog
PDDocumentNameDictionary names = new PDDocumentNameDictionary( doc.getDocumentCatalo
names.setEmbeddedFiles( efTree );
doc.getDocumentCatalog().setNames( names );
```

Standard 14 Fonts

The PDF specification states that a standard set of 14 fonts will always be available when consuming PDF documents. In PDFBox these are defined as constants in the PDType1Font class.

Standard Font
PDType1Font.TIMES_ROMAN
PDType1Font.TIMES_BOLD
PDType1Font.TIMES_ITALIC
PDType1Font.TIMES_BOLD_ITALIC
PDType1Font.HELVETICA
PDType1Font.HELVETICA_BOLD
PDType1Font.HELVETICA_OBLIQUE
PDType1Font.HELVETICA_BOLD_OBLIQUE
PDType1Font.COURIER
PDType1Font.COURIER_BOLD
PDType1Font.COURIER_OBLIQUE
PDType1Font.COURIER_BOLD_OBLIQUE
PDType1Font.SYMBOL
PDType1Font.ZAPF_DINGBATS

TrueType Fonts

Embedding TrueType Fonts

PDFBox supports embedding TrueType fonts. Loading a new font is easy.

```
PDDocument doc = PDDocument.load( ... );
PDFont font = PDTrueTypeFont.loadTTF( doc, new File( "SpecialFont.ttf" ) );
```

External TrueType Fonts

While it is recommended to embed all fonts for greatest portability not all PDF producer applications will do this. When displaying a PDF it is necessary to find an external font to use. PDFBox will look for a mapping file to use when substituting fonts.

PDFBox will load *Resources/PDFBox_External_Fonts.properties* off of the classpath to map font names to TTF font files. The *UNKNOWN_FONT* property in that file will tell PDFBox which font to use when no mapping exists.

Highlighting text in a PDF

There are cases when you might want to highlight text in a PDF document. For example, if the PDF is the result of a search request you might want to highlight the word in the resulting PDF document. There are several ways this can be achieved, each method varying in complexity and flexibility.

1. Use the 'search' open parameter

Acrobat supports passing is various parameters that tell it what to do once the PDF is open. See PDF Open Parameters for documentation on all the open parameters. One of the parameters is the 'search' parameter, this will automatically run the search functionality inside of Acrobat once the PDF is open. For example: http://pdfbox.apache.org/userguide/text_extraction.pdf#search="check"

NOTE: The words must be enclosed in quotes and separated by spaces; for example:#search="pdfbox rocks"

This is a great solution because of its simplicity! It doesn't require PDFBox at all, but it is a potential solution that many developers are not aware of.

2. Generate a highlight XML document

Acrobat also allows you to tell it to highlight specific words in the PDF document. It does this by passing an XML document to Acrobat when opening the PDF. See the PDF Highlight File Format for more detailed documentation.

Basically the document allows you to tell it the characters to highlight in the PDF by using character offsets on a page. As this is just an XML document, there are many ways you could create it but PDFBox does have a utility to make it easier. Take a look at the javadoc for the PDFHighlighter class. This will allow you specify a set of words that you want have highlighted and generate the XML document for you.

PDFBox also ships with a complete web application example of using this class, take a look at the pdfbox.war directory in your PDFBox installation.

You pass the xml to acrobat through a URL (or command line) parameter like this: http://pdfbox.apache.org /userguide/text_extraction.pdf#xml=http://pdfbox.apache.org/highlight.xml

NOTE: The value of the xml parameter must be a full URL to the XML document.
http://pdfbox.apache.org/userguide/text_extraction.pdf#xml=highlight.xml will not work
http://pdfbox.apache.org/userguide/text_extraction.pdf#xml=http://pdfbox.apache.org/highlight.xml is correct!

The one drawback to this solution is that you must parse the PDF and then generate an XML document, which is a time consuming operation.

3. Alter pdf contents to highlight specific text

Using PDFBox it is possible to regenerate the appearance stream to add highlighting to specific areas. While this is possible, it will require recreating a new PDF for every search request. There is nothing prebuilt in PDFBox to do this automatically for you and will require a significant coding effort.

You would need to

1. Find all locations of the text, determine x/y coordinates, width/height
2. Regenerate the PDF appearance stream and draw a highlighted box behind the text. Yellow would be easiest, if you want an inverted black/white, then you would need to change the color of the text to be white and draw a black box.
3. Stream the PDF back to the user

This is the most flexible but is also the most work to implement and is also more resource intensive.

Redistributing PDFBox

PDFBox makes use of several open source libraries. Some are just required for building PDFBox, some are required for running PDFBox. The below table summarizes the licences that are included with PDFBox and when they are required.

Product (with license link)	**Used for**	**Required for PDFBox redistribution**
Adobe AFM	Resource files for extracting font encoding. Bundled inside the PDFBox jar file	Yes
Adobe CMap	Resource files for CJK font mapping. Bundled inside the PDFBox jar file	Yes
Adobe Glyphlist	Mapping for the computation of a Unicode character string from a sequence of glyphs. Bundled inside the PDFBox jar file	Yes
Apache Ant	Tool for building PDFBox	No
bouncycastle	Encryption libraries for encrypting/decrypting PDF documents	Yes
FontBox	Font Library	Yes
JempBox	Library for working with XMP metadata.	Yes
IKVM	Library for .NET version of PDFBox	Only if using .NET version(the DLLs in /bin) of PDFBox
junit	Testing framework used in development	No
Apache Lucene	Text search engine library. PDFBox provides simple integration with Lucene.	Optional, only if using Lucene
ICU4J	Normalizing right to left text.	Optional, only if extracting right to left text

PDFBox Dependencies

PDFBox consists of a three related components and depends on a few external libraries. This page describes what these libraries are and how to include them in your application.

Core components

The three PDFBox components are named pdfbox, fontbox and jempbox. The Maven groupId of all PDFBox components is org.apache.pdfbox.

The fontbox and jempbox components are standalone libraries for handling font information and XMP metadata. These components have no external dependencies and can be used simply by adding the respective jar files to the classpath of your application.

The main PDFBox component, pdfbox, has hard dependencies on the fontbox and jempbox components and the commons-logging library. Commons Logging is a generic wrapper around different logging frameworks, so you'll either need to also use a logging library like log4j or let commons-logging fall back to the standard java.util.logging API included in the Java platform.

To add the pdfbox, fontbox, jempbox and commons-logging jars to your application, the easiest thing is to declare the Maven dependency shown below. This gives you the main pdfbox library directly and the other required jars as transitive dependencies.

```
<dependency>
  <groupId>org.apache.pdfbox</groupId>
  <artifactId>pdfbox</artifactId>
  <version>...</version>
</dependency>
```

Set the version field to the latest stable PDFBox version.

Optional dependencies

Some features in PDFBox depend on optional external libraries. You can enable these features simply by including the required libraries in the classpath of your application.

The most notable such optional feature is support for PDF encryption. Instead of implementing its own encryption algorithms, PDFBox uses libraries from the Legion of the Bouncy Castle. Both the bcprov and bcmail libraries are needed and can be included using the Maven dependencies shown below.

```
<dependency>
  <groupId>org.bouncycastle</groupId>
  <artifactId>bcprov-jdk15</artifactId>
  <version>1.44</version>
</dependency>
<dependency>
  <groupId>org.bouncycastle</groupId>
  <artifactId>bcmail-jdk15</artifactId>
  <version>1.44</version>
</dependency>
```

Another important optional feature is support for bidirectional languages like Arabic. PDFBox uses the ICU4J library from the International Components for Unicode (ICU) project to support such languages in PDF documents. To add the ICU4J jar to your project, use the following Maven dependency.

```
<dependency>
  <groupId>com.ibm.icu</groupId>
  <artifactId>icu4j</artifactId>
  <version>3.8</version>
</dependency>
```

PDFBox also contains extra support for use with the Lucene and Ant projects. Since in these cases PDFBox is just an add-on feature to these projects, you should first set up your application to use Lucene or Ant and then add PDFBox support as described on this page.

Dependencies for Ant builds

The above instructions expect that you're using Maven or another build tool like Ivy that supports Maven dependencies. If you instead use tools like Ant where you need to explicitly include all the required library jars in your application, you'll need to do something different.

The easiest approach is to run "mvn dependency:copy-dependencies" inside the pdfbox directory of the latest PDFBox source release. This will copy all the required and optional libraries discussed above into the pdfbox/target/dependencies directory. You can then simply copy all the libraries you need from this directory to your application.