PDFBox - User Guide

Table of contents

1. Introduction

This page will discuss the internals of PDF documents and those internal 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.

2. 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.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.pdfbox.cos.COSArray
Boolean	Standard True/False values	true	org.pdfbox.cos.COSBoolean
Dictionary	A map of name value pairs	<< /Type /XObject /Name (Name) /Size 1 >>	org.pdfbox.cos.COSDictionary
Number	Integer and Floating point numbers	1 2.3	org.pdfbox.cos.COSFloat org.pdfbox.cos.COSInteger
Name	A predefined value in a PDF document, typically used as a key in a dictionary	/Type	org.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 an a generation number. Initially the generation number will be zero unless the object get replaced later in the stream.	12 0 obj << /Type /XObject >> endobj	org.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.pdfbox.cos.COSStream
String	A sequence of characters	(This is a string)	org.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 ) );
```

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.

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

Note:

For example, each call to PDPage.getMediaBox() will return a new PDRectangle object, but will contain the same underlying COSArray.