Migration to PDFBox 2.0.0

Environment

PDFBox 2.0.0 requires at least Java 6

Packages

There are some significant changes to the package structure of PDFBox:

- Jempbox is no longer supported and was removed in favour of Xmpbox
- the package `org.apache.pdfbox.pdmodel.edit` was removed. The only class contained `PDPageContentStream` was moved to the parent package.
- all examples were moved to the new package “pdfbox-examples”
- all commandline tools were moved to the new package “pdfbox-tools”
- all debugger related stuff was moved to the new package “pdfbox-debugger”
- the new package “debugger-app” provides a standalone pre built binary for the debugger

Dependency Updates

All libraries on which PDFBox depends are updated to their latest stable versions:

- Bouncy Castle 1.53
- Apache Commons Logging 1.2

For test support the libraries are updated to

- JUnit 4.12
- JAI Image Core 1.3.1
- JAI JPEG2000 1.3.0
- Levigo JBIG ImageIO Plugin 1.6.3

For PDFBox Preflight

- Apache Commons IO 2.4

Breaking Changes to the Library

Deprecated API calls

Most deprecated API calls in PDFBox 1.8.x have been removed for PDFBox 2.0.0

API Changes

The API changes are reflected in the Javadoc for PDFBox 2.0.0. The most notable changes are:

- `getCOSDictionary()` is no longer used. Instead `getCOSObject` now returns the matching `COSBase` subtype.
- `PDXObjectForm` was renamed to `PDFormXObject` to be more in line with the PDF specification.
- `PDXObjectImage` was renamed to `PDImageXObject` to be more in line with the PDF specification.
- `PDPage.getContents().createInputStream()`was simplified to `PDPage.getContents()`.

- `PDPageContentStream` was moved to `org.apache.pdfbox.pdmodel`.

General Behaviour

PDFBox 2.0.0 is now parsing PDF files following the Xref information in the PDF. This is similar to the functionality using `PDDocument.loadNonSeq` with PDFBox 1.8.x. Users still using `PDDocument.load` with PDFBox 1.8.x might experience different results when switching to PDFBox 2.0.0.

Font Handling

Font handling now has full Unicode support and supports font subsetting.

TrueType fonts shall now be loaded using

```
PDType0Font.load
```

to leverage that.

`PDAfmPfbFont` has been removed. To load such a font pass the pfb file to `PDType1Font`. Loading the afm file is no longer required.

PDF Resources Handling

The individual calls to add resources such as `PDResources.addFont(PDFont font)` and `PDResources.addXObject(PDXObject xobject, String prefix)` have been replaced with `PDResources.add(resource type)` where `resource type` represents the different resource classes such as `PDFont`, `PDAbstractPattern` and so on. The `add` method now supports all the different type of resources available.

Instead of returning a `Map` like with `PDResources.getFonts()` or `PDResources.getXObjects()` in 2.0 an `Iterable<COSName>` of references shall be retrieved with `PDResources.getFontNames()` or `PDResources.getXObjectNames()`. The individual item can be retrieved with `PDResources.getFont(COSName fontName)` or `PDResources.getXObject(COSName xObjectName)`.

Working with Images

The individual classes `PDJpeg()`, `PDPixelMap()` and `PDCCitt()` to import images have been replaced with `PDImageXObject.createFromFile` which works for JPG, TIFF (only G4 compression), PNG, BMP and GIF.

In addition there are some specialized classes:

- `JPEGFactory.createFromStream` which preserve the JPEG data and embed it in the PDF file without modification. (This is best if you have a JPEG file).
- `CCITTFactory.createFromFile` (for bitonal TIFF images with G4 compression).
- `LosslessFactory.createFromImage` (this is best if you start with a BufferedImage).

Parsing the Page Content

Getting the content for a page has been simplified.

Prior to PDFBox 2.0 parsing the page content was done using

```
PDStream contents = page.getContents();
PDFStreamParser parser = new PDFStreamParser(contents.getStr
eam());
parser.parse();
List<Object> tokens = parser.getTokens();
```

With PDFBox 2.0 the code is reduced to

```
PDFStreamParser parser = new PDFStreamParser(page);
parser.parse();
List<Object> tokens = parser.getTokens();
```

In addition this also works if the page content is defined as an **array of content streams**.

Iterating Pages

With PDFBox 2.0.0 the prefered way to iterate through the pages of a document is

```
for(PDPage page : document.getPages())
{
    ... (do something)
}
```

PDF Rendering

With PDFBox 2.0.0 `PDPage.convertToImage` and `PDFImageWriter` have been removed. Instead the new `PDFRenderer` class shall be used.

```
PDDocument document = PDDocument.load(new File(pdfFilename
));
PDFRenderer pdfRenderer = new PDFRenderer(document);
int pageCounter = 0;
for (PDPage page : document.getPages())
{
    // note that the page number parameter is zero based
    BufferedImage bim = pdfRenderer.renderImageWithDPI(pageC
ounter, 300, ImageType.RGB);

    // suffix in filename will be used as the file format
    ImageIOUtil.writeImage(bim, pdfFilename + "-" + (pageCou
nter++) + ".png", 300);
}
document.close();
```

`ImageIOUtil` has been moved into the `org.apache.pdfbox.tools.imageio` package. This is in the `pdfbox-tools` download. If you are using maven, the `artifactId` has the same name.

Important notice when using PDFBox with Java 8

Due to the change of the java color management module towards “LittleCMS”, users can experience slow performance in color operations. Solution: disable LittleCMS in favour of the old KCMS (Kodak Color Management System):

- start with `-Dsun.java2d.cmm=sun.java2d.cmm.kcms.KcmsServiceProvider` or call
- `System.setProperty("sun.java2d.cmm", "sun.java2d.cmm.kcms.KcmsServiceProvider");`

Sources:
http://www.subshell.com/en/subshell/blog/Wrong-Colors-in-Images-with-Java8-100.html
https://bugs.openjdk.java.net/browse/JDK-8041125

> Since PDFBox 2.0.4

PDFBox 2.0.4 introduced a new command line setting

`-Dorg.apache.pdfbox.rendering.UsePureJavaCMYKConversion=true`

which may improve the performance of rendering PDFs on some systems especially if there are a lot of images on a page.

PDF Printing

With PDFBox 2.0.0 `PDFPrinter` has been removed.

Users of `PDFPrinter.silentPrint()` should now use this code:

```
PrinterJob job = PrinterJob.getPrinterJob();
job.setPageable(new PDFPageable(document));
job.print();
```

While users of `PDFPrinter.print()` should now use this code:

```
PrinterJob job = PrinterJob.getPrinterJob();
job.setPageable(new PDFPageable(document));
if (job.printDialog()) {
    job.print();
}
```

Advanced use case examples can be found in th examples package under
org/apache/pdfbox/examples/printing/Printing.java

Text Extraction

In 1.8, to get the text colors, one method was to pass an expanded .properties file to the PDFStripper constructor. To achieve the same in PDFBox 2.0 you can extend `PDFTextStripper`and add the following `Operators` to the constructor:

```
addOperator(new SetStrokingColorSpace());
addOperator(new SetNonStrokingColorSpace());
addOperator(new SetStrokingDeviceCMYKColor());
addOperator(new SetNonStrokingDeviceCMYKColor());
addOperator(new SetNonStrokingDeviceRGBColor());
addOperator(new SetStrokingDeviceRGBColor());
addOperator(new SetNonStrokingDeviceGrayColor());
addOperator(new SetStrokingDeviceGrayColor());
addOperator(new SetStrokingColor());
addOperator(new SetStrokingColorN());
addOperator(new SetNonStrokingColor());
addOperator(new SetNonStrokingColorN());
```

Interactive Forms

Large parts of the support for interactive forms (AcroForms) have been rewritten. The most notable change from 1.8.x is that there is a clear distinction between fields and the annotations representing them visually. Intermediate nodes in a field tree are now represented by the `PDNonTerminalField` class.

With PDFBox 2.0.0 the prefered way to iterate through the fields is now

```
PDAcroForm form;
...
for (PDField field : form.getFieldTree())
{
    ... (do something)
}
```

Most `PDField` subclasses now accept Java generic types such as `String` as parameters instead of the former `COSBase` subclasses.

PDField.getWidget() removed

As form fields do support multiple annotations `PDField.getWidget()` has been removed in favour of `PDField.getWidgets()`which returns all annotations associated with a field.

PDUnknownField removed

The `PDUnknownField` class has been removed, such fields are treated as `null` see PDFBOX-2885 (https://issues.apache.org/jira/browse/PDFBOX-2885).

Document Outline

The method `PDOutlineNode.appendChild()` has been renamed to `PDOutlineNode.addLast()`. There is now also a complementary method `PDOutlineNode.addFirst()`.

Why was the ReplaceText example removed?

The ReplaceText example has been removed as it gave the incorrect illusion that text can be replaced easily. Words are often split, as seen by this excerpt of a content stream:

```
[ (Do) -29 (c) -1 (umen) 30 (tation) ] TJ
```

Other problems will appear with font subsets: for example, if only the glyphs for a, b and c are used, these would be encoded as hex 0, 1 and 2, so you won't find "abc". Additionally, you can't replace "c" with "d" because it isn't part of the subset.

You could also have problems with ligatures, e.g. "ff", "fl", "fi", "ffi", "ffl", which can be represented by a single code in many fonts. To understand this yourself, view any file with PDFDebugger and have a look at the "Contents" entry of a page.

See also https://stackoverflow.com/questions/35420609/pdfbox-2-0-rc3-find-and-replace-text