PDFs mit Java erstellen und bearbeiten

Java-Akrobatik

PDF-Dateien selbst schreiben ist mit freier Software kein Problem. Wer mit Java ein PDF erzeugen möchte, muss nicht mal zu externen Programmen greifen: Das freie Projekt I-Text bietet alles, was dafür notwendig ist. Die neueste Version unterstützt auch XML-Formate. Bernhard Bablok

Der Erfolg des PDF-Formats hat zwei Gründe: Zum einen gab es immer kostenlose Acrobat Reader, zum anderen bietet das Format einige Vorteile: Hyperlinks, eingebettete Bilder, Formularverarbeitung, Metadaten, Verschlüsselung und gute Drucktreue der Dokumente. Das Projekt I-Text [1] des Belgiers Bruno Lowagie stellt ein Java-API für die Erzeugung von PDF-Dateien bereit. Es handelt sich um eine Bibliothek, nicht um ein Satz- oder Layoutsystem.

Damit eignet es sich für die programmgesteuerte Ausgabe von strukturierten Daten, weniger als Allroundwerkzeug für PDF-Dateien. Wer volle Kontrolle über die Ausgabe haben möchte, wird mit I-Text zwar Erfolg haben, aber nicht unbedingt glücklich werden. Dieser Coffee-Shop führt das I-Text-API anhand einfacher Beispiele ein. Wer tiefer einsteigen will, kommt nicht darum herum, das recht gute Tutorial zu studieren. Außerdem gibt es verschiedene Tutorials zu I-Text ([2], [3]), die sich zum Beispiel damit beschäftigen, wie man mit Servlets PDF-Dateien erzeugt.

Download und Installation

Das I-Text-Paket an sich ist relativ kompakt (705 KByte Quellen). Die Bibliothek besteht aus zwei Jar-Dateien (»itext-1.02b.jar« und »iTextHYPH.jar«) mit zusammen 1,1 MByte. Wer die Javadoc-Dokumentation nicht selbst generieren möchte, findet die fertigen HTML-Seiten (1,2 MByte) ebenfalls online. Weiterhin ist ein Paket mit dem Tutorial (91 KByte) und den zugehörigen Beispielen (1,7 MByte) verfügbar. All dies gibt es auf der Homepage [1] des I-Text-Projekts (wobei die eigentlichen Dateien auf Sourceforge liegen). Dort findet sich auch eine experimentelle XML-Unterstützung. Ein praktisches All-in-one-Paket fehlt leider.

Die Installation ist denkbar einfach: Die Jar-Dateien gehören in den »CLASSPATH« der Anwendung. Bei der Installation des Tutorials muss man etwas Acht geben, sonst stimmen die relativen Links nicht:

```
mkdir -p /usr/local/iText/tutorial
mkdir -p /usr/local/iText/examples
tar -xvzf itext-tutorial-0.94.tar.gz ➥
  -C /usr/local/iText/tutorial
tar -xvzf itext-examples.tar.gz ➥
  -C /usr/local/iText/examples
```

Für Selbstkompilierer ist allerdings Handarbeit angesagt, da das Paket weder Makefile noch Ant-Datei enthält. Die erforderlichen Schritte zeigt Listing 1. Alternativ findet sich ein Ant-Buildfile »build.xml« auf der Projekt-Homepage. Damit lässt sich sogar der Code herunterladen:

```
ant download.site
```

Jar-Files erzeugt man mit »ant jar« respektive »ant jarWithXML« für die Version mit XML-Unterstützung. Das I-Text-Paket steht unter einer dualen Lizenz: Zur Wahl stehen die LGPL oder die Mozilla Public License.

Hello World

Den Einstieg in die I-Text-Programmierung verschafft ein erweitertes Hello-World-Beispiel, siehe Listing 2. Das Programm liest von der Standardeingabe und übergibt den daraus erzeugten String an I-Text. Wer zum Beispiel per Pipe das GPL-Dokument ins Programm schreibt, erhält das in Abbildung 1 dargestellte Ergebnis.

Vom notwendigen Drumherum einmal abgesehen passiert das Interessante in den Zeilen 41 bis 45. Etwas ungewöhn-

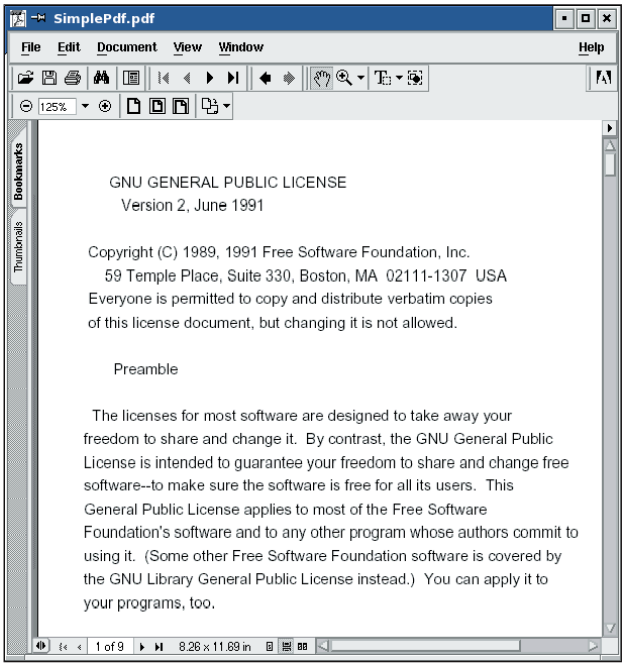

Abbildung 1: Das Programm aus Listing 2 zeigt den GPL-Wortlaut im PDF-Format.

lich und erklärungsbedürftig ist dabei die Zeile 42. Dieser Methodenaufruf verknüpft das Dokument mit einer »PdfWriter«-Instanz. Eine »Document.addWriter()«- oder »PdfWriter.setDocument()«-Methode wäre klarer.

Für die wenigen Codezeilen sieht das Ergebnis schon ganz passabel aus. Wie es sich noch verfeinern lässt, zeigen die folgenden Beispiele.

Metadaten und andere Eigenschaften

Metadaten wie Titel, Thema oder Schlüsselwörter sind zum Beispiel dann nützlich, wenn das Dokument in ein Archiv aufgenommen werden soll und man die entsprechenden Daten für die Suche verwenden möchte. Die »Document«-Klasse besitzt dafür Methoden (siehe Listing 4), die vor dem Öffnen des Dokuments aufzurufen sind. Der Acrobat-Reader zeigt diese Metadaten in den PDF-Eigenschaften an (siehe Abbildung 2).

Funktional wichtiger als Metadaten sind aber andere Eigenschaften des Dokuments, etwa die Papiergröße, die Ausrichtung oder die Ränder.

Neben dem Default-Konstruktor (Listing 2, Zeile 41) von »Document« gibt es noch zwei weitere Konstruktoren, mit denen sich Papiergröße und Ränder definieren lassen. Ersteres erfolgt über ein »Rectangle«-Objekt (eine Klasse in »com.lowagie.text«, nicht zu verwechseln mit der Klasse »java.awt.Rectangle«), Letzteres über absolute Größenangaben der vier Ränder. Gemessen wird jeweils in Points, wobei ein Inch (2,54 cm) 72 Points entspricht.

Um fehleranfälliges und redundantes Rechnen zu verhindern, empfiehlt es sich, die vordefinierten, statischen Objekte aus der »PageSize«-Klasse zu verwenden. So entsteht ein Dokument im A4-Seitenformat mit Rändern der Größe 0,5 Inch:

```
Document doc = new Document(PageSize.A4)
```

»PageSize« hat eine Reihe nützlicher Methoden wie »setBackgroundColor()« oder »rotate()«. Letztere definiert eine Seite im Querformat.

Bildergalerie im PDF

Grundlegend in der I-Text-Architektur ist das »Element«-Interface, das von einfachen Bausteinen eines PDF-Dokuments implementiert wird. Kleinster Textbaustein ist der Chunk, ein String mit einem definierten Font. Mehrere Chunks bilden eine Phrase. Dieses Layoutelement hat als zusätzliches Attribut einen definierten Zeilenabstand. Paragraph ist eine Subklasse von Phrase und bietet zusätzlich Einrückung und Ausrichtung. Anchor ist eine spezielle Phrase und die Basis für Hyperlinks.

Neben den Textbausteinen gibt es weitere Elemente. Das »Rectangle« wurde beim obigen Beispiel schon angespro-

Listing 1: I-Text selbst bauen

```
01 mkdir -p /tmp/iText/src
02 mkdir -p /tmp/iText/build
03 tar -xvzf itext-src-1.02b.tar.gz -C /tmp/iText/src
04 cd /tmp/iText
05 javac -d build `find src -type f -name *.java`
06 cd src
07 find . -name *.afm -exec cp -v --parents {} ../build \;
08 cd ..
09 jar -cf itext.jar -C build .
```

Listing 2: »SimplePdf.java«

```
22 import java.io.*;
23
24 import com.lowagie.text.*;
25 import com.lowagie.text.pdf.*;
26
34 public class SimplePdf {
35
38   public static void main(String[] args) {
39     try {
40       SimplePdf pdf = new SimplePdf();
41       Document document = new Document();
42       PdfWriter.getInstance(document,new
           FileOutputStream("SimplePdf.pdf"));
43       document.open();
44       document.add(new Paragraph(pdf.getLines()));
45       document.close();
46     } catch (Exception e) {
47       e.printStackTrace();
48     }
49   }
50
51   ////////////////////////////////////////////////////
52
57   private String getLines() throws IOException {
58     BufferedReader reader = new BufferedReader(new
           InputStreamReader(System.in));
59     StringBuffer result = new StringBuffer();
60     String line;
61     while((line = reader.readLine()) != null)
62       result.append(line).append("\n");
63     return result.toString();
64   }
65 }
```

Listing 3: »Gallery.java«

```
037 public class Gallery {
038   ...
067   private static final Font iFont = FontFactory
068     .getFont(FontFactory.HELVETICA,20,Font.BOLD,iColor);
069
070   ////////////////////////////////////////////////////
071
072   public static void main(String[] args) {
073     try {
074       // setup environment
075       Gallery gallery = new Gallery();
076       String galleryName = gallery.parseArguments(args);
077       gallery.loadProperties();
078
079       // prepare document and writer
080       Document document  = gallery.getDocument();
081       gallery.getWriter(document,galleryName);
082       gallery.addMetaInfo(document);
083       document.open();
084
085       // process individual images
086       for (int i=0;i<iImageName.length;++i)
087         gallery.addPage(document,iImageName[i]);
088
089       // finish processing
090       gallery.addIndex(document);
091       document.close();
092     } catch (Exception e) {
093       e.printStackTrace();
094     }
095 }
122   private PdfWriter getWriter(Document doc, String file) throws
          Exception {
123     PdfWriter writer =
124       PdfWriter.getInstance(doc,new FileOutputStream(file));
126     writer.setViewerPreferences(PdfWriter.PageModeFullScreen);
127     PdfTransition transition = new PdfTransition
          (PdfTransition.OUTBOX,3);
128     writer.setTransition(transition);
129     return writer;
130   }
131
140   private void addMetaInfo(Document doc) {
141     String prop;
142     if ((prop = iProps.getProperty("gallery.title")) != null)
143       doc.addTitle(prop);
144     if ((prop = iProps.getProperty("gallery.subject")) != null)
145       doc.addSubject(prop);
146     if ((prop = iProps.getProperty("gallery.keywords")) != null)
147       doc.addKeywords(prop);
148     if ((prop = iProps.getProperty("gallery.author")) != null)
149       doc.addAuthor(prop);
150     if ((prop = iProps.getProperty("gallery.creator")) != null)
151       doc.addCreator(prop);
152     // the header is in fact no metainfo, but it fits here
153     if ((prop = iProps.getProperty("gallery.header")) != null) {
154       HeaderFooter header =
155         new HeaderFooter(new Phrase(prop,iFont),false);
156       header.setBorder(Rectangle.BOTTOM);
157       header.setBorderColor(iColor);
158       header.setAlignment(Element.ALIGN_CENTER);
159       doc.setHeader(header);
160     }
161   }
162
172   private void addPage(Document doc,String filename) throws
          Exception {
173     // create image
174     Image img = Image.getInstance(filename);
175     img.setAlignment(Image.MIDDLE);
176     img.scaleToFit(600,400);
177     doc.add(img);
178
180     File file = new File(filename);
181     String imageTitle = iProps.getProperty(file.getName(),
            file.getName());
182     Anchor anchor = new Anchor(imageTitle,iFont);
183     anchor.setName(file.getName());
184     Paragraph title = new Paragraph(anchor);
185     title.setAlignment(Element.ALIGN_CENTER);
186     doc.add(title);
187     doc.newPage();
188   }
189
198   private void addIndex(Document doc) throws Exception {
199     Table indexTable = new Table(2);
200     indexTable.setCellsFitPage(true);
201
202     // iterate over images
203     for (int i=0; i<iImageName.length; ++i) {
204       Image img = Image.getInstance(iImageName[i]);
205       // scale to thumbnail-size
206       float width = img.plainWidth();
207       float height = img.plainHeight();
208       float ratio;
209       if (width > height)
210         ratio = 80/width;
211       else
212         ratio = 80/height;
213       img.setAlignment(Image.MIDDLE);
214       img.scalePercent(ratio*100);
215       indexTable.addCell(new Cell(img));
216
217       // set image description
218
219       File file = new File(iImageName[i]);
220       String imageTitle = iProps.getProperty(file.getName(),
              file.getName());
221       Anchor title = new Anchor(imageTitle,iFont);
222       title.setReference("#"+file.getName());
223       indexTable.addCell(new Cell(title));
224     }
225     doc.add(indexTable);
226   }
282 }
```

chen. Dort kommen noch weitere vor: »HeaderFooter« für Kopf- und Fußzeilen, »Image« für Bilder und »Table« beziehungsweise »Cell« für Tabellen.

Ein komplexeres Anwendungsbeispiel benutzt diese Grundelemente und demonstriert die Gestaltungsmöglichkeiten des I-Text-API: die Umwandlung einer Liste von Bildern in eine PDF-basierte Bildergalerie. Der Vorteil gegenüber den weit verbreiteten Webgalerien ist, dass alles in einer einzigen, wenn auch recht großen Datei steckt.

Das Galerie-Programm erhält die Liste der Bilder als Argument. Metadaten sowie die Titel der Bilder holt es aus einer Properties-Datei. Im Prinzip ließe sich alles (Farben, Fonts und so weiter) aus dieser Properties-Datei konfigurieren, der Übersichtlichkeit halber verzichtet das Beispiel darauf.

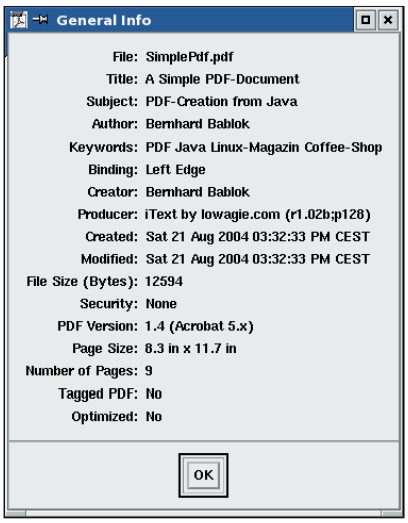

Abbildung 2: Der Acrobat Reader zeigt in seinem Info-Fenster die mit I-Text geschriebenen Metainformationen einer PDF-Datei.

Bilder einbinden

Die Struktur der Datei »Gallery.java« ist einfach, siehe Listing 3, die komplette Datei findet sich unter [7]. Zuerst erzeugt es ein »Document«, einen »Writer«, setzt die Metainformationen und öffnet schließlich das Dokument (Zeilen 80 bis 83). Danach bearbeitet es die Liste der Bilder mit einer Seite pro Bild (ab Zeile 86). Am Schluss erzeugt das Programm einen Index und schließt das Dokument (ab Zeile 90).

Der Acrobat Reader soll die Bilder im Vollbildmodus anzeigen, mit einem automatischen Übergang zwischen den Bildern nach drei Sekunden. Die entsprechenden Attribute setzt man überraschenderweise im »PdfWriter« (Zeilen 122 bis 130) und nicht im »Document« – das I-Text-API ist nicht immer konsistent. Unter Linux mit dem veralteten Acrobat Reader 5 funktioniert der Bildershow-Effekt aber leider nicht.

Die Zeilen 154 bis 159 zeigen, wie aus einem »Rectangle« und dem »HeaderFooter«-Objekt eine Kopfzeile entsteht. Jeder Rand sowie seine Farbe und Dicke sind einzeln konfigurierbar. Bilder in die Galerie aufnehmen ist ein Kinderspiel. Die Methode »addPage()« erzeugt ein »Image«-Objekt, skaliert es und fügt es ins Dokument ein (ab Zeile 174).

I-Text bettet die komplette Bilddatei in das Bild ein, beim Skalieren ändert sich also die Pixeldichte. Abbildung 3 demonstriert das Ergebnis. Der Titel des Bildes ist ein Anchor-Objekt (ab Zeile 180). Er dient als Hypertext-Sprungmarke vom Index aus.

Tabellen und Hyperlinks

Die »addIndex()«-Methode (Zeilen 198 bis 226) ist eigentlich unnötig, da jedes PDF-Dokument auch eine Thumbnail-Sicht bietet – über eine entsprechende Writer-Konfiguration kann man sie sogar automatisch einblenden. Der selbst gemachte Index dient aber zur Demonstration von Tabellen und Hyperlinks.

Tabellen lassen sich entweder mit Reihen- und Spaltenzahl oder nur durch die

Listing 4: Hinzufügen von Metainformationen

```
75 private void addMetaInfo(Document doc) {
76    doc.addTitle("A Simple PDF-Document");
77    doc.addSubject("PDF-Creation from Java");
78    doc.addKeywords("PDF Java Linux-Magazin Coffee-Shop");
79    doc.addAuthor("Bernhard Bablok");
80    doc.addCreator("Bernhard Bablok");
81 }
```

- Anzeige -

Abbildung 3: Eine Beispielseite der Bildershow im Fullscreen-Modus, mit I-Text programmiert.

Spaltenzahl definieren. Im letzteren Fall fügt I-Text bei Bedarf neue Zeilen an die Tabellen an. Zellen können ebenfalls fehlen, sie werden dann aber auch nicht angezeigt.

Der Index des Galerie-Beispiels braucht zwei Spalten. Die erste nimmt die Thumbnails (verkleinerte Vorschaubilder) auf, die zweite Spalte den Bildtitel (oder den Dateinamen, wenn es keinen Titel gibt). Das Programm liest die Bildgröße (Zeilen 206 und 207) und verkleinert das Thumbnail so, dass die längere Bildkante 80 Pixel groß ist. Das entstandene »Image«-Objekt verpackt man anschließend in ein »Cell«-Objekt und fügt es in die Tabelle ein.

Leider hat I-Text die Semantik der Hyperlinks von HTML übernommen. Hier wie dort ist es verwirrend, dass ein Anchor-Objekt als Sprungziel, aber auch als Absprungquelle dient. Andererseits vereinfacht dies aber auch die Verwendung von Links. Das Ziel ist wie in HTML mit vorgestelltem Gartenzaun anzugeben (Zeilen 219 bis 223). Neben diesen internen Links kann PDF natürlich auch externe Dokumente referenzieren, was auch I-Text unterstützt.

Für Fortgeschrittene

Weitere Möglichkeiten von I-Text seien hier nur kurz erwähnt. Natürlich beherrscht es auch die Gliederung in Kapitel und Abschnitte, was die Navigation in großen Dokumenten deutlich erleichtert. Nützlich sind außerdem die Möglichkeiten der so genannten Page-Events. Mit ihnen lässt sich auf vergleichsweise einfache Weise zum Beispiel ein Index bauen. Die erforderlichen Schritte beschreibt das Tutorial.

Für die Feinsteuerung des Layouts steht das »Graphics2D«-Objekt von I-Text zur Verfügung. Es bietet alle Möglichkeiten des Java-2D-API, was zum Beispiel für die Ausgabe von Diagrammen und Funktionskurven nützlich ist. Die in früheren Coffee-Shops vorgestellten Projekte JFreechart [4] und JFreereport [5] nutzen diese Technik von I-Text.

Noch nicht Teil der Kerndistribution von I-Text, aber in Entwicklung ist die Unterstützung von XML. Neben der XML-Ausgabe über einen »XMLWriter« (analog zum »PdfWriter« in den Beispielen) bringt das Paket noch eine interessantere Möglichkeit mit: die Erzeugung von PDF aus XML. Dafür gibt es eine Klasse, die XML-Dokumente gemäß der I-Text-DTD verarbeitet. Sie unterstützt zwar noch nicht alle Konstrukte, ist aber fast komplett. Mit grundlegenden XSLT-Kenntnissen ist der Weg vom XML-Input zum PDF-Ergebnis nicht weit.

Wer mit der recht komplexen XSLT-Syntax auf Kriegsfuß steht, dem hilft I-Text mit einer einfacheren Technologie. Eine simple Tag-Map setzt eigene Tags in I-Text-Tags um. Folgendes Beispiel erzeugt aus dem ursprünglichen Markup »Definition« einen fett gedruckten I-Text-»paragraph«:

```
<tag name="paragraph" alias="Definition">
  <attribute name="style" value="bold" />
</tag>
```

finally{}

I-Text ist ein mächtiges Toolkit für die Textausgabe im PDF-Format. Dank der guten Dokumentation bleibt die Lernkurve flach. Ein einziger Schönheitsfehler ist anzumerken: Weil die I-Text-Klassen genauso heißen wie die Java-Standardklassen (zum Beispiel »Rectangle« und »Font«), ist es erforderlich, im eigenen Programm vollqualifizierte Klassennamen zu verwenden.

Das Lesen von PDFs beherrscht I-Text nur rudimentär. Die Bibliothek bringt dazu ein paar Utilities mit (Concat, Split, Handout und Encrypt), die Dokumentation ist allerdings nicht auf dem aktuellen Stand. Wer bestehende PDFs lesen und weiterverarbeiten will, sollte eher einen Blick auf die Seite des PDF-Box-Projekts [6] werfen. *(ofr)* ■

Infos

[1] I-Text-Homepage: [http://www.lowagie.com/iText/]
[2] Dynamically Creating PDFs in a Web Application: [http://www.onjava.com/pub/a/onjava/2003/06/18/dynamic_files.html]
[3] „Druck gemacht – mit iText dynamisch PDF-Dokumente generieren“: „iX“ 08/04, S. 116
[4] Bernhard Bablok, „Malen nach Zahlen – Mit JFreechart Diagramme zeichnen“: Linux-Magazin 05/04, S. 116
[5] Bernhard Bablok, „Zahlen zeigen – Daten aufbereiten mit JFreereport“: Linux-Magazin 06/04, S. 118
[6] PDF-Box-Projekt: [http://www.pdfbox.org]
[7] Listings dieses Coffee-Shops: [http://www.linux-magazin.de/Service/Listings/2004/11/Coffeeshop]

Der Autor

Bernhard Bablok arbeitet bei der AGIS mbH als Anwendungsentwickler. Wenn er nicht Musik hört, mit dem Radl oder zu Fuß unterwegs ist, beschäftigt er sich mit Themen rund um Objektorientierung. Er ist unter [coffee-shop@bablokb.de] zu erreichen.