TRILOS

Mächtig viel Druck machen

PDF-Dokumente lizenzfrei mit PHP erzeugen

Von Ulrich Hacke

Trilos IT-Dienstleistungen
Am Rathaus 15
30592 Ronnenberg

Stand: 3. Dezember 2003

Inhaltsverzeichnis

1 Einleitung

„Adobe PDF (Portable Document Format) ist der bereits seit über 10 Jahren weltweit erfolgreich eingesetzte De-facto-Standard für den zuverlässigen und sicheren Austausch von elektronischen Dokumenten aller Art.", heißt es auf der Website von Adobe Inc. Ob man sich nun auf die zehn Jahre festlegen möchte oder nicht (mehr dazu kann man in Kapitel 8 nachlesen) - PDF ist im heute einfach nicht mehr wegzudenken. Ganz egal, ob man PDF im „prepress"-Bereich einsetzt (gemeint ist damit die Vorabveröffentlichung von layoutgebundenen Materialien vor dem eigentlichen Druck) oder für layouttreue und druckbare Dokumente im Internet: überall wird man auf das PDF-Format stoßen.

Die nötige Software, um PDF-Dokumente zu erzeugen, liefert Adobe natürlich (kostenpflichtig) gleich mit; zudem sind viele gängige Office-Applikationen in der Lage, Daten im PDF-Format zu speichern. Nimmt man sich aber den Webbereich vor, helfen diese Applikationen nicht mehr weiter – hier sind Webservices oder Scripte gefragt, die aus beliebigen eingehenden Daten PDF-Dokumente „on the fly" erzeugen.

Dieses Tutorial beschreibt, wie man genau das lizenz- und kostenfrei tun kann.

Zielgruppe sind fortgeschrittene und erfahrene PHP-Programmierer, die sich mit einem neuen Gebiet beschäftigen wollen. Ein grundlegendes Verständnis von Klassen und Objekten ist zwingend erforderlich.

Zu Beginn wird die eingesetzte FPDF vorgestellt, ihre grundsätzliche Anwendung erläutert und mit mehreren Beispielen („Case studies") näher beleuchtet.

2 Voraussetzungen

Benötigt werden eine PHP-Umgebung (Windows oder Linux, Webserver), der jeweilige Lieblingseditor, ein PDF-Reader (z.B. der kostenlose Adobe Reader), die frei verfügbare FPDF-Bibliothek und natürlich dieses Tutorial. Die PHP-Umgebung sollte vorhanden sein; die übrige Software lässt sich unter den angegebenen Links (Kapitel 7) downloaden. PHP sollte in der Version ab 4.3 oder höher installiert sein.

FPDF wurde erstmalig 2001 vorgestellt. Die aktuelle Version ist die 1.51.

3 Die FPDF-Bibliothek

Dies Kapitel beschreibt den grundlegenden Aufbau und die Verwendung der FPDF-Bibliothek.

3.1 Warum nicht die pdfLIB verwenden?

Sucht man im PHP-Manual nach „PDF", wird man bei den Funktionen der pdfLIB (geschrieben von Thomas Merz) schnell fündig. Um diese Bibliothek zu verwenden, sind aber noch weitere Zusatzinstallationen nötig (JPEG- und TIFF-Bibliotheken). Lädt man unerschrocken die pdfLIB herunter, wird man bei der Durchsicht der Lizenzbestimmungen recht schnell ernüchtert. Der Einsatz auf einem 1-CPU-System kostet bereits 450,00$ und die Preisskala reicht noch weit. Damit wäre das Hauptargument gegen die zugegebenermaßen höchst performant arbeitende pdfLIB klar: Kosten!

Mit der frei verfügbaren FPDF sind wir aber (fast) genauso gut beraten. Da sie allerdings selbst in PHP geschrieben ist, muss man mit einer Verringerung der Performanz gegenüber der pdfLIB rechnen. Solange wir jedoch eher kürzere Dokumente (<5-10 Seiten) erzeugen und unsere Scripte nicht auf einer stark frequentierten Site einsetzen, sollte dieser Nachteil nicht ins Gewicht fallen. Zudem können wir bei der FPDF in den Quellcode

schauen und das ganze System jederzeit erweitern. Wie das geht, ist in Kapitel 5.1 beschrieben.

3.2 Installation

Die Vorbereitungen für den Einsatz der FPDF sind erfreulich kurz. Das lediglich 180 KB große ZIP-Archiv findet in wenigen Sekunden den Weg auf die heimische Festplatte. Sein Inhalt wird in ein beliebiges Verzeichnis (z.B. /fpdf) unterhalb der DocumentRoot des Webservers extrahiert. Die entstandene Verzeichnisstruktur sieht so aus:

Abb. 1 FPDF-Verzeichnisstruktur

Weitere Softwareinstallationen oder Konfigurationen werden nicht benötigt (Ausnahme: möchte man die erzeugten PDF-Dokumente komprimieren, dann wird die Zlib benötigt. Ist sie nicht vorhanden, wird die Kompression einfach deaktiviert).

4 Grundlagen

4.1 FPDF verwenden

FDPF präsentiert sich als eine einzige große PHP-Klasse. Um sie einzusetzen, müssen wir sie inkludieren und ihr mitteilen, wo sie ihre Zeichensätze (Fonts) finden kann:

```
<?
define("FPDF_FONTPATH","fpdf/font/");
include "fpdf/fpdf.php";
?>
```

Alternativ zur Konstante FPDF_FONTPATH kann man das Font-Directory übrigens auch im INCLUDE_PATH von PHP mit integrieren. Die vom FPDF-Autor vorgeschlagene zusätzliche Alternative, die die Fontfiles in das gleiche Verzeichnis zu legen, in dem das aufzurufende Script läuft, halte ich aus Gründen der Übersichtlichkeit für wenig empfehlenswert.

Damit ist die erste „Hürde" genommen und wir beginnen mit einem tatsächlichen PDF-Dokument.

4.2 Unser erstes Dokument

Traditionellerweise erzeugen wir uns als erstes das weltberühmte „Hallo Welt"-Dokument. Der Quellcode dazu sieht so aus:

```
$pdf = new FPDF("P", "mm", "A4");
$pdf->Open();
$pdf->AddPage();
$pdf->SetFont("Times","",11);
$pdf->Write(5, "Hallo Welt");
$pdf->Output();
```

Abb. 2 „Hallo Welt" als PDF

Mehr Code ist tatsächlich nicht erforderlich. Schauen wir uns die Zeilen etwas genauer an: zu Beginn erzeugen wir uns eine neue Instanz der FPDF-Klasse. Dazu werden drei (optionale) Parameter übergeben, nämlich die Ausrichtung („P" für „paragraph" oder „hochkant" bzw. „L" für „landscape" bzw. „quer"), die Maßeinheit, in der wir rechnen wollen (möglich sind hier Millimeter (mm), Punkt (pt), Zentimeter (cm) oder Inches (in)). Es folgt die Methode open(), welche die PDF-Generierung startet. Da bis hierhin noch keine einzige Seite erstellt wurde, müssen wir AddPage() aufrufen. Jetzt benötigt die Klasse zwingend eine Schriftart, da sie sonst mit einer Fehlermeldung abbricht. Das erledigt die Methode SetFont() mit den drei Parametern „Schriftart", „Format" („b" für „bold", „i" für „italic" oder „" für „normal"). Über Write() geben wir unseren Text aus – der erste Parameter bestimmt die Zeilenhöhe (hier also 5cm).

Abschließend senden wir unser Dokument via Output() an den Browser. Dieser startet den ihm zugeordneten PDF-Reader... et voilá: das erste Dokument ist fertig.

Output() kann das erzeugte Dokument auch in eine Datei speichern; dazu muss lediglich ein Dateiname inklusive Pfad als Parameter übergeben werden und es muss sichergestellt sein, dass PHP im gewünschten Verzeichnis Schreibrechte besitzt.

4.3 „Das Dokument ist eine Scheibe"

Man muss sich klarmachen, dass die Erzeugung eines PDF-Dokumentes sich stark von der Erstellung einer dynamischen Webseite unterscheidet: ein PDF-Dokument präsentiert sich Seite für Seite als eine frei beschreibbare „Malfläche", auf der wir beliebige Inhalte ausgeben können. Indem wir einen „virtuellen Cursor" auf dieser Fläche positionieren, können wir einzelne Bereiche gezielt ansteuern und unsere Elemente exakt positionieren.

Damit kann man eindrucksvolle Effekte erzielen, z.B. lassen sich Texte übereinander legen. Arbeiten wir dagegen mit eher festen Strukturen (wie z.B. einer Tabelle), dann sollten die Inhalte der einzelnen Zellen möglichst nicht über ihre Begrenzungen hinausragen. Übrigens: es macht technischen gesehen keinen Unterschied, ob wir nun Texte oder grafische Elemente (Punkte, Linien, Bilder) ausgeben.

4.4 Textformatierungen

FPDF stellt eine Reihe von Methoden bereit, mit denen wir Textblöcke formatieren können:

Methode	Beschreibung	Parameter
SetDrawColor()	Legt die Zeiechenfarbe für alle Umrandungen fest	Drei Byte-Werte für die GB-Anteile bzw. einen Wert für Graustufen.
SetFillColor()	Legt die Füllfarbe für alle ausgefüllten Bereiche fest	s.o.
SetFont()	Legt die Schriftart für die Zeichenausgabe fest	Schriftart (Name), Style (b/i/u), größe.
SetFontSize()	Ändert die Schriftgröße	Größe
SetTextColor()	Legt die Schriftfarbe für die Zeichenausgabe fest	Siehe SetDrawColor()

In der Standardinstallation werden die folgenden Schriftarten unterstützt:

- Courier (Schrift mit fester Zeichenbreite)
- Helvetica oder Arial (serifenlose Proportionalschrift)
- Times (Proportionalschrift mit Serifen)
- Symbol (Symbole)
- ZapfDingbats (Symbole)

Es können beliebig viele weitere Schriftarten hinzugefügt werden. Der Vorgang hierzu ist in der mitgelieferten Dokumentation ausführlich beschrieben und wird in diesem Tutorial nicht näher beschrieben.

4.5 PDF-Eigenschaften

Einem PDF-Dokument können zusätzliche Informationen zugewiesen werden; auch hierfür gibt es die nötigen Methoden, wie z.B.:

- SetAuthor()
- SetCreator()
- SetKeywords()
- SetSubject()
- SetTitle()

Weitere Informationen zu diesen teilweise selbst erklärenden Methoden kann man der mitgleieferten Dokumention von FPDF entnehmen.

4.6 Vorteile der Klassenstruktur

Dass FPDF in Form einer PHP-Klasse verfügbar ist, bringt gewaltige Vorteile mit sich. Mittels Unterklassen, die von FPDF abgeleitet sind, lässt sich das komplette System im Handumdrehen erweitern, ohne dass man den Original-Quellcode auch nur anschauen muss. Mehr dazu steht in Kapitel 5.1.

Zusätzlich ist – wie wir gesehen haben – der eigene für die PDF-Erstellung Quellcode erfreulich kurz und übersichtlich.

5 Case Studies

5.1 Sprungmarken und Links

FPDF unterstützt sowohl interne Links (Sprungmarken) also auch externe Verweise (Weblinks), die sich mit wenigen Codezeilen realisieren lassen:

```
$link = $pdf->AddLink();
$pdf->SetLink($link,0,2);
$pdf->Write(5, "Seite 1\n");
$pdf->SetTextColor(0,0,200);
$pdf->Write(5, "Gehe zu Seite 2", $link);

$pdf->AddPage();

$pdf->SetTextColor(0,0,0);
$pdf->Write(5,"Seite 2\n");
$pdf->SetTextColor(0,0,200);
$pdf->Write(5,"http://www.trilos.de", "http://www.trilos.de");
```

Abb. 3 Sprungmarken und Links

Um eine Sprungmarke zu setzen, müssen wir sie zunächst via AddLink() instanzieren. SetLink() setzt sie in das Dokument ein; dafür werden bis zu drei Paramter benötigt – die Sprungmarke selbst, vertikale Zielposition und Nummer der Zielseite (die letzten beiden Parameter haben jeweils die Defualtwerte 0 und können auch weggelassen werden.

Für externe Links brauchen wir lediglich in den Ausgabefunktionen Write(), Cell() oder MultiCell() die gewünschte URL als letzten Parameter zu übergeben. Das Beispiel verändert zusätzlich noch die Textfarbe, um die Links/Sprungmarken deutlicher hervorzuheben.

5.2 Grundlage: Die Wrapper-Klasse PDFDOC

Zunächst erstellen wir uns eine eigene Wrapper-Klasse namens PDFDOC, um die FDPD ein wenig zu erweitern und den durch uns zu erstellenden Quellcode möglichst kurz zu halten. Der Wrapper wird für alle folgenden Beispiele benötigt.

```
// Abgeleitete PDFDOC-Klasse
class PDFDoc extends PDF {
 // Übergeordnetes Objekt
 var $pdf;
 // PDF im Konstruktor erstellen
 function PDFDoc() {
       $this->pdf = new PDF("P", "mm", "A4");
       $this->pdf->SetTitle("PDF-Vortrag");
       $this->pdf->AliasNbPages();
       $this->pdf->Open();
       $this->pdf->SetMargins(25,25,20);
       $this->pdf->AddPage();
 }
function Display() {
       header("Content-type: application/pdf");
       $this->pdf->Output();
 }
}
```

Abb. 4 Einfaches Gerüster der Wrapper-Klasse PDFDOC

PDFDOC besitzt lediglich einen Konstruktor, der uns ein PDF mit den gängigen Eigenschaften vorbereitet, sowie eine Methode Display(), die das Erzeugte ausgibt.

5.3 Dokument aus Bausteinen zusammenstellen

Beginnen wir damit, uns ein längeres Dokument aus Bausteinen zusammenzustellen. Der Einfachheit halber definieren wir uns zwei Typen von Bausteinen: Überschriften (groß und fett) sowie Absätze (normal und mehrzeilige Fliesstexte). PDFDOC soll beliebig viele dieser Bausteine entgegennehmen können. Wir erweitern den Wrapper also um zwei Methoden:

```
// Überschrift hinzufügen
function AddHeading($text) {
 // Font stzen
 $this->pdf->SetFont("Times", "B", 14);
 // Überschrift ausgeben
 $this->pdf->Write(5, $text);
 // Abstand
 $this->pdf->Ln(10);
}

// Absatz hinzufügen
function AddParagraph($text) {
 $this->pdf->SetFont("Times", "", 11);
 $this->pdf->Write(5, $text);
 $this->pdf->Ln(10);
}
```

Abb. 5 PDFDOC: Textbausteine entgegennehmen

Eine kurze Demo überprüft das ordnungsgemäße Funktionieren:

```
$d = new PDFDOC();
for($i=1;$i<5;$i++) {
 $d->AddHeading($i.". Kapitel");
 $d->AddParagraph(str_repeat("Gallia est omnia divisa in partes
tres... ", 4+($i*2)));
}
$d->Display();
```

Abb. 6 Textbausteine an PDFDOC übergeben

5.4 Grafik-Ausgabe

Als nächstes betrachten wir die grafischen Fähigkeiten der FPDF; mit Hilfe von Line() und SetDrawColor() lassen sich schöne Effekte erzielen – auch hier wieder als neue Methode innerhalb von PDFDOC realisiert:

```
// Grafische Spielerei
function fancyGraphic() {
 for($x=10;$x<=200;$x++) {
       $y = 60+49*sin($x/20);
       // Malfarbe verändern
       $this->pdf->SetDrawColor(100,200-$x,$x);
       // Linie zeichen
       $this->pdf->Line($x, $y, $y, 100-$y/2);
       $this->pdf->Line($y, (100-$y/2)+100, $y+100, $x);
 }
}
```

Abb. 7 Grafische Spielerei

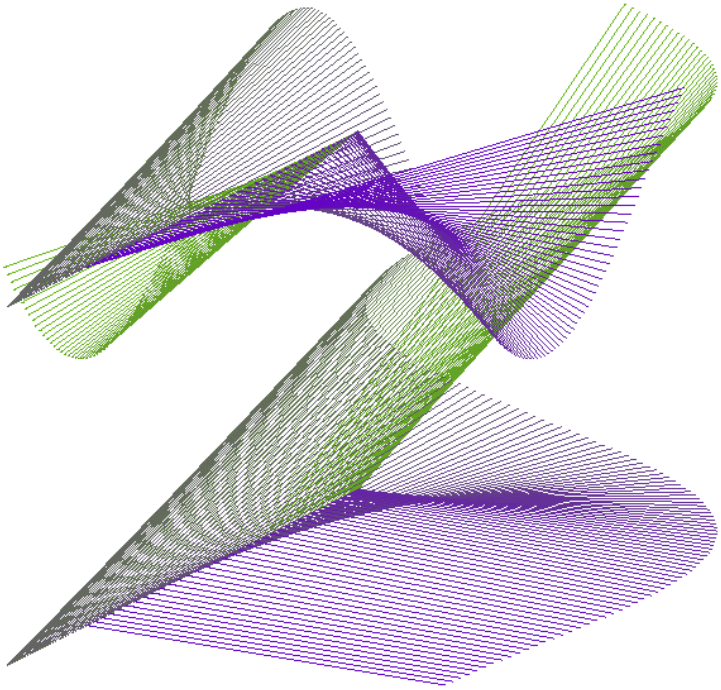

Abb. 8 ...und das ist das grafische Ergebnis

5.5 Einfaches Diagramm

Oftmals steht man vor der Aufgabe, Daten grafisch dazustellen. Mit Hilfe der Gdlib ist das in PHP auch ohne weiteres möglich – und in PDF geht das natürlich auch. Unsere neue Methode simpleDiagram() liefert uns auf komfortable Art ein Balkendiagramm. Als Eingabeparameter erwartet sie eine Überschrift für das Diagramm, die darzustellenden Werte in Form eines einfachen assoziativen Arrays, sowie eine Farbe für die Balken in HTML-konformer Hexadezimal-Notation.

Hier lernen wir eine weitere Methode der FPDF kennen: Cell(), die mit bis zu acht Parametern sehr mächtig ist. Cell() erzeugt uns eine rechteckige Fläche (Zelle) mit einer optionalen Umrandung, einer Hintergrundfarbe und einem Ausgabetext. Die linke obere Ecke der Zelle entsprecht der aktuellen Position des „virtuellen Cursors". Cell() kennt die folgenden Parameter:

1. Breite der Zelle (wenn 0, dann erstreckt sich die Zelle bis zum rechten Rand)
2. Höhe (Standardwert: 0)
3. Ausgabetext (String)
4. Umrandung (0=ohne, 1=mit Rahmen) oder alternativ eine Kombination aus L, T, R und B für left/top/right/bottom
5. Neue Position des Cursors nach der Ausgabe: 0=nach rechts, 1=an den Anfang der nächsten Zeile, 2=darunter
6. Ausrichtung des Ausgabetextes: L (oder leer)=linksbündig, R=rechtsbündig, C=zentriert
7. Füllung: 0=transparent, 1=mit Füllfarbe
8. Link

Hinweis: Für die Ausgabe von längeren Texten (mit Zeilenumbrüchen) ist die Methode MultiCell() vorgesehen. Hier werden die Texte entweder automatisch umgebrochen – d.h. wenn sie den rechten Seitenrand erreichen – oder aber manuell durch das Steuerzeichen „\n". Insgesamt werden so viele Zeilen wie nötig untereinander dargestellt. Weitere Möglichkeiten für die Textausgabe stehen auf der FPDF-Homepage zum Download bereit.

```
// Einfaches Diagramm
function simpleDiagram($title, $data, $color="0000e0") {
 // Font setzen
 $this->pdf->SetFont("Times", "B", 14);
 // Titel ausgeben
 $this->pdf->Write(5, $title);
 $this->pdf->Ln(8);
 $this->pdf->SetFont("Times", "", 11);
 // Malfarbe setzen
 $this->pdf->SetFillColor(hexdec(substr($color,0,2)),
hexdec(substr($color,2,2)), hexdec(substr($color,4,2)));
 // Werte analysieren
 while(list($key, $value) = each($data)) {
       // Breitestes Label
       if($this->pdf->GetStringWidth($key)>$kmax) $kmax = $this->pdf-
>GetStringWidth($key)+5;
       // Größter Datenwert
       if($value>$vmax) $vmax = $value;
 }
 $faktor = (150-$kmax)/$vmax;
 // Daten ausgeben
 reset($data);
 while(list($key, $value) = each($data)) {
       // Label
       $this->pdf->Cell($kmax,5,$key,0,0,"R");
       // Balken
       $this->pdf->Cell($value*$faktor,5," ",0,0,"C",1);
```

```
        // Beschriftung
        $this->pdf->Cell(0,5,number_format($value,0,",","."),0,1,"L");
        $this->pdf->Ln(1);
  }
  // Abstand
  $this->pdf->Ln(5);
}

$d = new PDFDoc();

$d->simpleDiagram("Zusammensetzung des Deutschen Bundestages (2002-
2004)", array("SPD"=>251, "CDU/CSU"=>247, "B90/Die Grünen"=>55,
"FDP"=>47, "fraktionslos"=>3),"ffa07a");

$d->simpleDiagram("Flächen einiger Bundesländer (Quadratkilometer)",
array("Baden-Württemberg"=>35751, "Bayern"=>70554,
"Brandenburg"=>29061, "Hessen"=>21114, "Niedersachsen"=>47344,
"Rheinland-Pfalz"=>19849, "Thüringen"=>16254),"2e8b57");

$d->Display();
```

Abb. 9 Einfaches Balkendiagramm (Quellcode)

Nach dem Aufruf von simpleDiagram() legt die Methode die gewünschten PDF-Eigenschaften fest, gibt den Titel aus und analysiert die übergebenen Werte, damit die Ausgabe auch von der Größe her passt. Die FPDF-Methode Ln() fügt einen frei definierbaren Zeilenabstand ein.

So sieht das Ergebnis des Beispielcodes aus:

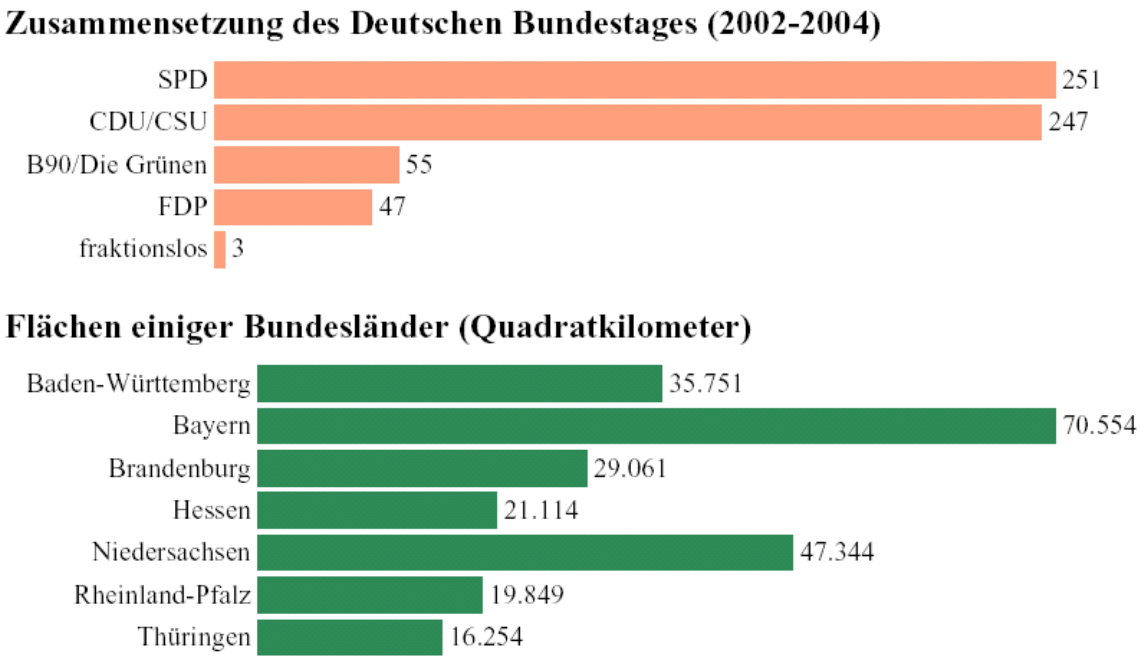

Abb. 10 Beispieldiagramme

Übrigens: weitere Lösungen für die Erzeugung von Diagrammen stehen zum Download auf der FPDF-Homepage bereit.

5.6 mySQL-Operationen

Zum Abschluss der Case studies zeige ich ein Beispiel, um mySQL-Tabellen in ansprechender Form zu dokumentieren. Ziel soll die Auflistung aller wichtigen Eigenschaften einer mySQL-Tabelle sein. Für das Beispiel wird eine beliebige mySQL-Datenbank mit mindestens einer Tabelle beliebiger Struktur benötigt. Im Code-Beispiel stellen wir als erstes eine Verbindung zum Datenbankserver her und rufen unsere neue PDFDOC-Methode mit zwei Parametern auf, nämlich dem Datenbankhandle und dem Namen der zu dokumentierenden Tabelle.

Die Methode erstellt uns eine aus mehreren Cell()-Blöcken bestehende Tabellenkopfreihe und gibt anschließend alle Tabellenfelder mit ihren Eigenschaften untereinander aus. Die Funktionsweise ist in etwa dem bekannten phpMyAdmin vergleichbar.

Es sei noch einmal darauf hingewiesen: unsere „Tabelle" hat mit einer herkömmlichen HTML-Tabelle nicht viel mehr als die Bezeichnung gemein. Da unser PDF-Dokument eine variable Zeichenfläche ist (vergl. Kapitel 4.3), könnten die Zelleninhalte die Umrahmungen (aus der Kopfzeile) jederzeit überschreiben. Entweder nimmt man dieses (unschöne) Verhalten schweigend in Kauf, oder man schreibt sich Routinen, welche die zu erwartenden Stringlängen berücksichtigen und ggf. die Kopfzeilen variabel gestalten – ganz ähnlich wurde ja bereits beim Balkendiagramm aus dem vorherigen Kapitel 5.5) verfahren.

```
// Verbindung zur Datenbank herstellen
$dbh = mysql_connect("localhost", "root", "");
mysql_select_db("demo");

// mySQL-Tabellenstruktur ausgeben
function mySQLTable($dbh, $table) {
 // Felder der Tabelle auslesen
 $query = mysql_query("show fields from ".$table, $dbh);
 if(mysql_affected_rows($dbh)>0) {
       // Tabellenlayot vorbereiten
       $this->pdf->SetFont("Helvetica", "B",14);
       $this->pdf->SetFillColor(200,200,200);
       $this->pdf->Write(5, "Tabellenstruktur von \"$table\"");
       $this->pdf->SetFont("","",11);
       $this->pdf->Ln(8);

       // Kopfzeilen schreiben
       $this->pdf->Cell(55,5,"Field",1,0,"C",1);
       $this->pdf->Cell(30,5,"Type",1,0,"C",1);
       $this->pdf->Cell(20,5,"Null",1,0,"C",1);
       $this->pdf->Cell(35,5,"Extra",1,1,"C",1);

       // Datensätze der Feldzeilen ausgeben
       while($row = mysql_fetch_array($query)) {
             $this->pdf->Cell(55,5,$row["Field"],1,0,"",0);
             $this->pdf->Cell(30,5,$row["Type"],1,0,"",0);
             $this->pdf->Cell(20,5,($row["Null"]=="") ?
"No":"Yes",1,0,"",0);
             $this->pdf->Cell(35,5,$row["Extra"],1,1,"",0);
       }
       // Abstand
       $this->pdf->Ln(10);
 }
}
```

Abb. 11 mySQL-Tabelle dokumentieren

Würde man zuvor via PHP alle Tabellen einer gegebenen Datenbank herausfinden und die PDFDOC-Methode mySQLTable() in einer Schleife aufrufen, wäre das Ergebnis eine ordentliche Datenbankdokumentation, die man sogar einem Kunden aushändigen könnte.

Tabellenstruktur von "mitarbeiter"

Field	Type	Null	Extra
mitarbeiter_id	int(11)	No	auto_increment
name	char(100)	No	
vorname	char(100)	No	
gehalt	decimal(6,2)	No	

Abb. 12 Ausgabe der mySQL-Tabelle

5.7 Kopf- und Fußzeilen

Für mehrseitige PDF-Dokumente bietet es sich an, regelmäßig wiederkehrende Kopf- und Fußzeilen zu implementieren. FPDF bringt dafür bereits zwei Methoden mit; Header() und Footer(), die in der ursprünglichen Mutterklasse jedoch leer sind. Aufgerufen werden sie übrigens automatisch. Um sie einzusetzen, sollte man sie in der eigenen Unterklasse implementieren – in unserem Falle wäre das unsere altgediente Wrapper-Klasse PDFDOC.

In der Kopfzeile wollen wir unser Firmenlogo inklusive Schriftzug unterbringen, während in der Fußzeile die jeweils aktuelle Seitenzahl plus der Gesamtseitenzahl erscheinen soll. Für letzteres bietet FDPD die Methode PageNo(), welche die gerade aktuelle Seitenzahl liefert, sowie einen Alias {nb}, der beim Abschluss des Dokumentes ersetzt wird. Der Alias kann via AliasNbPages() im übrigen auch umdefiniert werden.

```
// Kopfzeile
function Header() {
  // Logo einfügen
  $this->pdf->SetFont(“Arial”,”B”,12);
  $this->pdf->Cell(0,5,"www.trilos.de",0,0,"R");
  $this->pdf->Image(“trilos_logo.png”,10,8,46);
  $this->pdf->Ln(10);
}

function Footer() {
  // Position auf 1.5 cm vom unteren Rand entfernt setzen
  $this->pdf->SetY(-15);
  $this->pdf->SetFont('Arial','I',8);
  // Seitennumer
  $this->Cell(0,10,”Seite „.$this->PageNo().” von {nb}”,0,0,”C”);
}
```

Abb. 13 Automatische Kopf- und Fußzeilen

6 Ausblick

Die FPDF-Bibliothek ist eine leistungsstarke Alternative zur teuren pdfLIB. Bei einigermaßen schnellen Servern mit gut ausgebautem RAM fällt auch der Performanz-Verlust durch den reinen PHP-Einsatz kaum ins Gewicht. Erst wenn man längere Dokumente erzeugt, macht sich die Verzögerung bemerkbar.

Für die dynamische Erzeugung von PDF-Dokumenten existiert ein Fülle von Anwendungsmöglichkeiten. Vor allem auf Websites von Unternehmen werden es die Besucher zu schätzen wissen, wenn sie aktuelle Informationen in diesem druckerfreundlichen Format abrufen können. Dank der überragenden Layoutfähigkeiten von PDF, die von FPDF voll unterstützt werden, kann man statt Textdokumenten aber auch komplexere Ausgaben rendern, wie z.B. Rechnungen, Briefe, Präsentationen, Dokumentationen...

Im Bereich „Scripts" auf der FPDF-Homepage finden sich bereits eine Menge Anregungen. Fazit: auch ohne aufwändige Zusatzsoftware lässt sich in PHP fast alles realisieren.

7 Links

- FPDF-Homepage
  http://www.fpdf.org

- Adobe Reader Downloadpage
  http://www.adobe.de/products/acrobat/readstep2.html

- PDF-Spezifikationen
  http://partners.adobe.com/asn/tech/pdf/specifications.jsp

- PHP pdfLIB
  http://www.pdflib.com

- PDF-Portal
  http://www.planetpdf.com

- Dieses Tutorial
  http://www.hacke.net/php/pdf

8 Die Geschichte von PDF

Bereits in den Kindertagen der ersten PC's – damals tatsächlich noch liebevoll „personal computer" genannt – entstand der Traum vom „papierlosen Büro". Aus dieser Vision ist bis heute nicht viel geworden; zeigt uns die Geschichte doch, dass Computer den Papierberg viel mehr vergrößert haben, anstatt ihn auch nur ansatzweise zu verringern.

Anfang der 90er Jahre schwamm das noch völlig unausgereifte Projekt PDF auf genau der Idee vom „papierlosen Büro"; ausgedacht vom Adobe-Gründer John Warnock. In einem firmeninternen Papier beschrieb, wie toll es doch wäre, wenn man vollständige Dokumente über eMail-gestützte Netzwerke verschicken könnte, die auf jeder Maschine angezeigt und ausgedruckt werden könnten. Man erinnere sich: dies war die Zeit der ASCII-Welt, in der man Texte mit einfachen Editoren schrieb und HTML oder Multimedia ein Unwort waren.

Immerhin verfügte man bei Adobe schon über zwei Technologien, welche die Grundlage für Warnocks „Freizeit-Projekt" bildeten: PostScript war die universelle Grafikprogrammiersprache, über die man plattformunabhängig Druckdaten für Dokumente beschreiben konnte, sowie den Illustrator, der solcher Dokumente auch erzeugen konnte ebenfalls auf mehreren Plattformen lief (um genau zu sein, waren es zwei; nämlich Windows und MacOS – aber damit waren damals mehr als 99% aller Computer abgedeckt). Aus diesen beiden Technologien entwickelte man bei Adobe kurzerhand das neue Dateiformat PDF sowie Applikationen, die das neue Format erzeugen und darstellen konnten.

1991 ging man mit PDF erstmalig in San Jose an die Öffentlichkeit. Ein Jahr später wurde das neue Format auf der Comdex in Las Vegas noch unter dem Namen IPS 1.0 (Interchange PostScript) vorgestellt und prompt mit dem „Best Of Comdex" ausgezeichnet. Die

heute überall bekannte Software „Acrobat" („Distiller" zum Erzeugen und „Reader" zum Anzeigen von PDF-Dateien) lieferte Adobe am 15. Juni 1993 aus. Zu diesem Zeitpunkt kannte PDF interne Links, Bookmarks und eingebettete Fonts. Übrigens: Acrobat sollte zunächst „Camelot" heißen – später taufte man es in „Carousel" um; daraus ergab sich für zumindest auf dem Macintosh der Dateityp „Caro" für PDF-Dokumente.

Anfangs nahm man bei Adobe horrende Preise für die neue Technik; der Distiller kostete zwischen 695,00$ (single) und 2.495,00$ (network edition). Sogar der Reader schlug mit stolzen 50,00$ zu Buche. Diese Preispolitik trug natürlich nicht gerade dazu bei, dass PDF „über Nacht" zum shooting star wurde. Das sah auch Adobe ein und senkte die Preise; der Reader wurde fortan kostenlos verteilt.

Acrobat 2 (PDF 1.1) wurde 1994 released und unterstützte nun auch externe Links, security-Eigenschaften, Anmerkungen und einen unabhängigen Farbraum (zuvor wurde nur RGB unterstützt). Anfangs war Adobe selbst sein bester Kunde; brachte das Format aber dadurch voran, dass alle Adobe-Dokumente für Entwickler nur noch als PDF ausgeliefert wurden. Einer der ersten Großkunden waren die US-Finanzbehörden.

1996 erschien Acrobat 3 (Codename „Amber") mit PDF 1.2, dass neben Formularelementen jetzt auch Halbtonfarben sowie Überlagerungen und vor allem den bei der Druckindustrie so beliebten CMYK-Farbmodus unterstützte. Auf einmal wurde PDF für Grafikdesigner interessant, die mit PDF nun die Möglichkeit erhielten, sogenannte „prepress releases" ihrer Arbeiten zu veröffentlichen. „prepress", also die Vorstufe vor dem Druck, war ideal, um Werke vorab zu publizieren und die Arbeitskosten drastisch zu senken. Gleichzeitig erschien ein Plugin für Netscape, dass PDF-Dokumente im Browser darstellen könnte. Der rasante Boom des Internets katapultierte PDF weit nach vorne. Mit Agfa erhielt Adobe einen Kunden, der ab 1998 damit begann, PDF in hohem Masse für kommerzielle Druckerzeugnisse einzusetzen.

Allmählich wurden aber auch kritische Stimmen laut, denn noch immer waren externe Tools für die Erzeugung von PDF-Dokumente nötig. Hinzu kam, dass man 1991 PDF als einen sehr offenen Standard spezifiziert hatte. Das Ergebnis: es gab einfach zu viele Möglichkeiten, eine tadellos gültige PDF-Datei zu erzeugen, die aber leider nicht verwertbar war, da der Acrobat Reader sie nicht anzeigen konnte.

Um diese Misere zu beseitigen gründeten mehrere Firmen ein Konsortium, das 1998 den neuen Standard PDF/X-1 veröffentlichte. PDF/X-1 basierte auf PDF 1.2, enthielt aber eine höchst detaillierte Beschreibung, wie die Datei später aussehen sollte. Außerdem war sichergestellt, dass alle Fonts eingebettet oder hochaufgelöste Bilder auch tatsächlich vorhanden waren.

Im April 1999 kam Acrobat 4 (Codename „Stout") mit PDF 1.3 auf den Markt, das neben neuen Farbräumen auch eine Technologie namens „smooth shading" unterstützte, welche den weichen Übergang von einer Farbe in die andere ermöglichte. Parallel dazu erweiterte man bei Adobe die Ausgabegröße eine PDF-Dokumentes auf bis zu 5.080mm im Quadrat (vorher waren es nur 1.143mm). Die Version 4.0 besaß bei Auslieferung jedoch viele Fehler, was Adobe veranlasste, ein zunächst kostenpflichtiges Update auf 4.05 nachzuschieben. Die vielen Proteste der User führten aber schnell dazu, dass das Update kostenlos an registrierte Kunden verschickt wurde. Seit Acrobat 4.05 kann man bei PDF von einem „Standard-Format" sprechen; über 100 Millionen Kopien der Reader-Software waren heruntergeladen worden.

Mitte 2000 erschien PDF 1.4, das aber kurioserweise nicht von einer neuen Acrobat-Software begleitet wurde, sondern zunächst ausschließlich vom Adobe Illustrator 9 unterstützt wurde. Erst knapp ein Jahr später war Acrobat 5 (Codename „Brazil") verfügbar und PDF 1.4 konnte sich ausbreiten. Die wichtigsten Neuerungen: Transparenzen, 128-Bit-Vesrchlüsselung, Support für JavaScript und vor allem die sogenannten „tagged PDFs", mit denen man Meta-Informationen wie Definitionen über Titel, Textblöcke etc. in

einer einzigen PDF-Datei publizieren konnte. Das war vor allem für den wachsenden Markt der eBooks interessant, da mit dieser Technologie PDF-Dokumente neu ausgelesen werden konnten um die Darstellung auf den unterschiedlichsten Ausgabegeräten zu optimieren. Ebenfalls (besonders für die Druckindustrie) interessant: Acrobat 5 konnte korrekte Farbüberlagerungen darstellen, d.h. wenn ein Anwender eine gelbe Box teilweise über eine rote Fläche legte, erschien die Überlappungszone folgerichtig in orange.

Der aktuelle Stand liegt derzeit bei PDF 1.5 und Acrobat 6 (Codename „Newport"). Im April 2003 veröffentlicht, sorgte diese Version neben vielen neuen Features aber auch für Verwirrungen. Der Acrobat Reader wurde in Adobe Reader umbenannt und kann neben PDF nun auch Adobe eBooks darstellen. Dadurch wird zwar mehr Funktionalität in einer Applikation gebündelt; dafür ist der Umfang des Softwaredownloads aber auch spürbar angestiegen. Das Wichtigste an PDF 1.5 ist: verbesserte Kompression durch „object streams" und JPEG 2000, Unterstützung von Layern, stark verbesserte Unterstützung von „tagged PDFs".

9 Abbildungsverzeichnis

10 Der Autor

Ulrich Hacke ist Jahrgang 1972 und arbeitet seit Mitte der achtziger Jahre mit Computern. Nach Abitur und Zivildienst absolvierte er das Studium der Religionspädagogik an der Ev. Fachhochschule Hannover, wo er sich in seiner Diplomarbeit intensiv mit den gesellschaftlichen Auswirkungen der Computertechnologie des Informationszeitalters beschäftigte. Es folgte ein halbes Jahr stellvertretende Projektleitung für "kirche online" in Frankfurt am Main.

1997 begann er mit der Ausbildung zum Informatikassistenten (Softwaretechnologie) am b.i.b. Hannover. Anschließend arbeitete er haupt- und freiberuflich im IT-Umfeld und gründete 1999 mit anderen das Unternehmen TRILOS IT-Dienstleistungen, dessen Geschäftsführer er heute ist.

Kontakt:

TRILOS IT-Dienstleistungen
Ulrich Hacke
Am Rathaus 15
30952 Ronnenberg
eMail: hacke@trilos.de