Mini-Trainingslager auf den MRMCD 2015: Einführung in die Struktur-Elemente des PDF-Dateiformats

Kurt Pfeifle <kurt.pfeifle@mykolab.com>

04. September 2015,
MRMCD 2015, Darmstadt

Geschichtliches

Was viele nicht wissen. . .

Das PDF-Dateiformat ist (fast) von Anfang an offengelegt und (mehr oder weniger gut) dokumentiert gewesen.

Adobe hat (fast) von Anfang an jedermann + und ohne Lizenzkosten freigestellt, das PDF-Format in eigenen (PDF lesenden, schreibenden oder modifizierenden) Produkten einzubauen.

Erste Version der PDF-Spezifikation: 1993 (“PDF-1.0”)

Fun Fact: Adobe versuchte damals zunächst, den Acrobat Reader für 50.- $US pro Benutzer an Firmen zu verkaufen

Versions-Geschichte

- PDF-1.0: 1993
- PDF-1.1: 1994
- PDF-1.2: 1996
- PDF-1.3: 1999
- PDF-1.4: 2001
- PDF-1.5: 2003
- PDF-1.6: 2004
- PDF-1.7: 2006 (Übergabe der Spec an die ISO!)
  PDF-1.7: 2008 veröffentlicht als *ISO 32000-1:2008*
  .....
- PDF-2.0: ca. 2016 (durch ISO!), wird *ISO 32000-2*

Was diesem offenen Standard zu verdanken ist. . .

Für Unix-Welt hat *Ghostscript* bereits sehr früh eine erste (fast FOSS) Implementierung geliefert:

- PDFs generieren und
- PDFs lesen, interpretieren, konvertieren

Viele andere PDF-Tools sind gefolgt.

XPDF, diverse Kommandozeilen-Utilities, LaTeX/pdflatex, Open/LibreOffice

Was ich weglasse. . .

Diverse PDF-Standards (I)

- ISO 32000-xxx (ab 2008)
  PDF-1.7 lt. ISO; seit ca. 5 Jahren hat Adobe die Kontrolle über das PDF-Format in der Version PDF-1.7 (und für alle künftigen Versionen) aufgegeben
- ISO 19005-xxx (ab 2005)
  PDF/A-1b und **PDF/A-1a**; nähere Festlegungen zur elektronischen *Langzeit-Archivierungs-Fähigkeit* von PDFs
  PDF/A-2 (Weiterentwicklung von PDF/A-1)
  PDF/A-3 (Weiterentwicklung von PDF/A...)
- ISO 15930-xxx (ab 2001)
  PDF/X-xxx; nähere Festlegungen, welche automatischen "Blind Exchange" gewährleisten sollen – vollautomatische Verarbeitungs-Workflows in der Druck-Industrie

Diverse PDF-Standards (II)

- ISO 24517 (ab 2008)
  PDF/E; nähere Festlegungen für technische Dokumente aus dem Bereich *Engineering*
- ISO 14289 (ab 2008)
  PDF/UA; nähere Festlegungen zum Aufbau *barrierefreier* PDF-Dokumente (*Universal Accessibility*)
- ISO 16612-xxx (ab 2010)
  PDF/VT-xxx; nähere Festlegungen zum Aufbau von PDFs, die dem hochvolumigen personalisiertem Druck dienen sollen (*Variable Data* und *Transactional* Printing)

Syntax

Grund-Aspekte der PDF-Syntax

Vier verschiedene Bereiche:

- Datei-Struktur
- Dokumenten-Struktur
- Objekte
- “Content Streams”

Datei-Struktur (I)

Grundlegende Gliederung in vier verschiedene Teile:

- **Header** (Einleitung):
  Angabe der PDF-Version
- **Objekte** (Hauptteil):
  jeweils eingeklammert in Schlüsselworte `obj ... endobj`
  (auch als *indirekte Objekte* bezeichnet)
- **`xref`-Tabelle** (Inhaltsverzeichnis):
  Angabe der Adresse jedes im Hauptteil enthaltenen Objekts
- **Trailer** (Schlussteil):
  Angabe der Position der `xref`-Tabelle

DEMO TIME!

```
open pdf-syntax/hello-world.pdf
vim -b pdf-syntax/hello-world.pdf
open pdf-syntax/pdf-structure-physical.png
open pdf-syntax/pdf-structure-tree.png
open pdf-syntax/pdf-structure-physical-reverse.png
```

Datei-Struktur (II)

Datei-Struktur wichtig für den Umgang mit folgenden Themen

- Behandlung mit Objekten innerhalb der PDF-Datei:
  - Wie speichern?
  - Wie lesend darauf zugreifen?
  - Wie modifizieren?

- Unabhängig von der Semantik (Bedeutung) der unterschiedlichen Objekt-Typen
- ("Verschlüsselung" auf Datei-Ebene, um Inhalt der Datei vor unberechtigtem Zugriff zu schützen)

Dokumenten-Struktur

- Verschiedene Inhalts-Typen der PDF-Datei:
  - Seiten
  - Fonts
  - Anmerkungen/Kommentare
  - Bilder
  - Text
  - usw.
- Verschiedene Objekt-Typen für verschiedene Inhalte
- Wie verwendet man die verschiedenen Objekt-Typen?

Objekte

- PDF-Dokument ist eine Daten-Struktur
- Struktur besteht sich aus einigen elementaren Daten-Objekten
- Dafür gelten einige festgelegte Regeln:
  - Lexikalische Konventionen
  - Syntax zur Definition essentieller Eigenschaften elementarer Objekte
  - Syntax zur Definition von “Stream”-Objekten (d.h. dem komplexesten aller Objekt-Typen)

“Content Streams”

- Beschreiben das Erscheinungsbild des Inhalts einer Seite (oder anderer grafisch relevanter Elemente)
- Bestehen aus Abfolge von Einzel-Instruktionen
- Diese Instruktionen lassen sich zwar ebenfalls als “Objekte” beschreiben
- Jedoch unterscheiden sie sich konzeptionell von jenen Objekten, welche die Dokumenten-Struktur festlegen
- Betreffen auch eingebettete Ressourcen der PDF wie Fonts, ICC-Profile, . . .

Details

Lexikalische Konventionen (I)

- PDF als Abfolge von “Bytes”
- Gruppen von Bytes kann man zu “Tokens” zusammenfassen
- Nicht-verschlüsselte PDF prinzipiell ohne “binäre” Bytes darstellbar
  - verwendet dann nur *druckbare* Zeichen aus dem ANSI-Zeichensatz
  - plus beliebige *“Whitespace”*-Zeichen

Lexikalische Konventionen (II)

- Allerdings sind PDFs keineswegs auf ANSI-Zeichen beschränkt
  - beliebige ‘binäre’ Bytes dürfen vorkommen
  - aus ASCII ***müssen*** dann weiterhin bestehen:
    1. Tokens, welche solche Objekte voneinander abtrennen, die die Datei-Struktur beschreiben
    2. alle reservierten PDF-Schlüsselworte und “Namen”
    3. bestimmte Arten von Arrays

Lexikalische Konventionen (III)

- Daten-Werte von Strings und Stream-Objekten kann man schreiben:
  (a) entweder komplett mittels ASCII-Zeichen,
  (b) oder komplett mittels Binär-Bytes
- In der Praxis enthalten PDFs häufig größtenteils Binärdaten:
  (a) lange Content-Streams sind komprimiert (spart Platz)
  (b) manche Daten-Unterstrukturen sind "natürlicherweise" binär:
  . Fonts
  . ICC-Profile
  . Raster-Bilder
  (ebenfalls aus Platzgründen)

Zeichen und Token

Verwendete Zeichen-Klassen in PDFs

Unterteilung in 3 Klassen:

- *Delimiter* (Begrenzer-Zeichen)
- *Whitespace* (Weißraum-Zeichen)
- *Regular Characters* (Normale Zeichen)

Delimiter (Begrenzer-Zeichen)

Delimiter haben eine Sonderstellung

- dienen zur Abgrenzung syntaktischer Einheiten (z.B. Arrays, Namen und Kommentare)
- genau diese Zeichen:

```
( )   [ ]   { }   < >   << >>   /   %
```

Whitespace (Weißraum-Zeichen)

- enthält genau diese Zeichen:
  `NUL`, `HORIZONTAL TAB`, `LINE FEED`,
  `FORM FEED`, `CARRIAGE RETURN`, `SPACE`
- Weißraum dient zur Trennung verschiedener Tokens
- Weißraum (wo erlaubt) darf beliebig groß sein
- darf aus beliebiger Mischung von Weißraum-Zeichen bestehen
- Sonderrolle für *EOL*-Zeichen (End Of Line / Zeilenende)
  - als EOL-Zeichen gelten:
    `LINE FEED` (LF), `CARRIAGE RETURN` (CR) oder `CR+LF`
  - gewisse Stellen erfordern lt. PDF-Syntax unbedingt ein EOL
  - ACHTUNG:
    (1) innerhalb einer PDF darf man alle 3 EOL-Arten beliebig mischen!
    (2) das `CR+LF`-EOL umfaßt 2 Bytes, nicht nur 1 Byte!

Regular Characters (normale Zeichen)

Regular Characters sind der ganze Rest, der in PDFs erlaubt ist:

- alle anderen Zeichen außer Delimiter und Whitespace
- auch solche Zeichen, die außerhalb des ASCII-Zeichensatzes liegen!

Außerdem wichtig:
PDF-Syntax ist *case sensitive*: ‘a’ und ‘A’ sind verschieden!

Objekte

9 verschiedene Objekt-Typen

Es gibt insgesamt 9 verschiedene Objekt-Typen, 6 einfache und 3 zusammengesetzte.

6 einfache (nee, da kommen jetzt ja 7 ?!?):

- *Bools* (Wahrheitswerte)
- *Integers* (Ganzzahlen)
- *Reals* (Fließkommazahlen)
- *Strings* (Text)
- *Names* (Namen)
- das *NULL*-Objekt
- (Zusätzlich, aber nicht als 'Objekt' zu betrachten: Kommentare)

Bools (Wahrheitswerte)

Es gibt genau 2 davon:

- ***true***
- ***false***

(exakt so geschrieben!)

Delimiter : Whitespace oder EOL
(in der Praxis zumeist ein einzelnes Leerzeichen)

Integers (Ganzzahlen)

Werte können positive oder negative Ganzzahlen oder ‘0’ sein

Delimiter : Whitespace oder EOL
(in der Praxis zumeist ein einzelnes Leerzeichen)

Reals (Fließkommazahlen)

Werte können positiv oder negativ sein.

Trennzeichen für die Dezimalstellen: .
(Ein Punkt, kein Komma! Auch nicht bei deutschen PDFs!

Führende Nullen kann man weglassen.
(`0.3456` ist gleichbedeutend mit `.3456`)

Delimiter : Whitespace oder EOL
(in der Praxis zumeist ein einzelnes Leerzeichen)

Strings (Text)

- Delimiter von Strings:
  (...) (bei *literalen* Strings) oder
  < ... > (bei *hexadezimal* dargestellten Strings)
- Beispiele für Strings:
  `(Ein String)` . <= literaler String
  `<45696e20537472696e67>` <= selber String in Hex
  `(\105\151\156\040\123\164\162\151\156\147)` <=
  . . Oktal

Umrechnung von *ASCII* nach *Dec* und *Hex*

Das Kommando **`ascii`** liefert die folgende Tabelle:

Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex		Dec	Hex	
0	00	NUL	16	10	DLE	32	20		48	30	0	64	40	@	80	50	P	96	60	`	112	70	p
1	01	SOH	17	11	DC1	33	21	!	49	31	1	65	41	A	81	51	Q	97	61	a	113	71	q
2	02	STX	18	12	DC2	34	22	"	50	32	2	66	42	B	82	52	R	98	62	b	114	72	r
3	03	ETX	19	13	DC3	35	23	#	51	33	3	67	43	C	83	53	S	99	63	c	115	73	s
4	04	EOT	20	14	DC4	36	24	$	52	34	4	68	44	D	84	54	T	100	64	d	116	74	t
5	05	ENQ	21	15	NAK	37	25	%	53	35	5	69	45	E	85	55	U	101	65	e	117	75	u
6	06	ACK	22	16	SYN	38	26	&	54	36	6	70	46	F	86	56	V	102	66	f	118	76	v
7	07	BEL	23	17	ETB	39	27	'	55	37	7	71	47	G	87	57	W	103	67	g	119	77	w
8	08	BS	24	18	CAN	40	28	(	56	38	8	72	48	H	88	58	X	104	68	h	120	78	x
9	09	HT	25	19	EM	41	29	)	57	39	9	73	49	I	89	59	Y	105	69	i	121	79	y
10	0A	LF	26	1A	SUB	42	2A	*	58	3A	:	74	4A	J	90	5A	Z	106	6A	j	122	7A	z
11	0B	VT	27	1B	ESC	43	2B	+	59	3B	;	75	4B	K	91	5B	[	107	6B	k	123	7B	{
12	0C	FF	28	1C	FS	44	2C	,	60	3C	<	76	4C	L	92	5C	\	108	6C	l	124	7C	|
13	0D	CR	29	1D	GS	45	2D	-	61	3D	=	77	4D	M	93	5D	]	109	6D	m	125	7D	}
14	0E	SO	30	1E	RS	46	2E	.	62	3E	>	78	4E	N	94	5E	^	110	6E	n	126	7E	~
15	0F	SI	31	1F	US	47	2F	/	63	3F	?	79	4F	O	95	5F	_	111	6F	o	127	7F	DEL

Names (Namen)

- Delimiter von Namen: `/...`
- Beispiel für einen Namen: `/Helvetica-Bold`

Dictionaries (I)

Beste dt. Übersetzung ist evtl. “Verzeichnisse”

Dictionary ist Liste von */Keynamen* mit ihren zugehörigen *Werten*.

Oder auch: eine *assoziative Tabelle, die Paare von Objekten enthält*.

Hierbei nennt man. . .

- . . . das erste Wort des Paares **“Key”** (Schlüssel) und
- . . . das zweite Wort **“Value”** (Wert)

Dictionaries (II)

Jedes zusammengehörige Paar eines Dictionaries heisst “Entry” (Eintrag):

- ***Key*** muss immer ein `/Name` sein
  (Unterschied zu PostScript: hier darf der Dictionary Key ein beliebiges Objekt sein).
- ***Value*** darf aus einem beliebigem anderen Objekt bestehen (inklusive eines Dictionarys).

Jeder Schlüssel darf pro Dictionary nur einmal vorkommen.

Dictionaries (III)

Als Wert in einem Entry sind alle anderen Objekt-Typen möglich, also:
Bools, Integers, Reals, Strings, Names, andere Dictionaries, Arrays, Streams, NULL-Objekt

Es kommt ***nicht*** auf die Reihenfolge der Entries innerhalb eines Dictionaries an.

Jede Reihenfolge ist gleichwertig.

Delimiter von Dictionaries: `<< ... >>`

Dictionaries (IV)

Beispiel für ein Dictionary:

```
<<                          % Öffnender Dictionary-Delimiter
   /Type     /XObject       %
   /Subtype  /Form          %
   /Filter   /FlateDecode   %
   /Length   30             %
   ....                     %
>>                          % Schließender Dictionary-Delimiter
```

Arrays (I)

Sequentielle ein-dimensionale Listen von beliebigen Objekt-Typen.

PDF unterstützt direkt nur ein-dimensionale Listen.
(Mehrdimensionale Listen kann man bei Bedarf erzeugen, indem man ein Array von Arrays konstruiert. Diese dürfen in beliebiger Tiefe verschachtelt sein.)

Es kommt ***unbedingt*** auf die Reihenfolge der Bestandteile eines Arrays an!

Andere Reihenfolge ==> andere Bedeutung (z.B. Dimensionen eines Rechtecks)

Delimiter von Arrays: `[ ... ]`

Arrays (II)

Generelles Beispiel für ein Array:

```
[                   % Öffnender Array-Delimiter
  0.1               % Array-Element mit Objekt-Typ *Real*
  33                % Array-Element mit Objekt-Typ *Integer*
  /Name             % Array-Element mit Objekt-Typ *Name*
  (Ein String)      % Array-Element mit Objekt-Typ *String*
  <</VarA /WertB>>  % Array-Element mit Objekt-Typ *Dictionary*
]                   % Schließender Array-Delimiter
```

Spezifisches Beispiel (wie die Seitengröße für A4 per Array definiert ist):

```
[                   % Öffnender Array-Delimiter
  0                 % Array-Element mit Objekt-Typ *Integer*: X-Koordinate linke, untere Ecke
  0                 % Array-Element mit Objekt-Typ *Integer*: Y-Koordinate linke, untere Ecke
  595               % Array-Element mit Objekt-Typ *Integer*: X-Koordinate rechte, obere Ecke
  842               % Array-Element mit Objekt-Typ *Integer*: Y-Koordinate rechte, obere Ecke
]                   % Schließender Array-Delimiter
```

Streams

Delimiter von Streams: die Schlüsselworte `stream ... endstream`

Beispiel für einen Stream:

```
<< ... >>                    % Dictionary
stream                       % Öffnender Stream-Delimiter
... null oder mehr Bytes ... % Stream-Inhalt
endstream                    % Schließender Stream-Delimiter
```

Kommentare

Delimiter von Kommentaren: `%.... <EOL>`

Beispiel für einen Kommentar:

```
% Dies ist ein Kommentar <EOL>
% Dies ist eine weitere Kommentar-Zeile <EOL>
```

Probleme

Das ‘Kreuz’ mit der PDF-Syntax (I)

PDF schön und gut und sehr weit verbreitet und für viele Zwecke einfach Klasse. . .

AAAABER:

- PDF-Syntax erlaubt in vielen Fällen nicht nur genau 1 gültigen Weg!
- Sondern sehr häufig 1, 2, 3, . . . viele Arten, eine bestimmte Sache auszudrücken.

Das öffnet Hacking- und Mißbrauchs-Versuchen Türen und Tore!

Nochmals Strings: lt. Spezifikation

Adobe Acrobat und Adobe Reader sind noch weitaus toleranter gegenüber ‘schlechter’ PDF-Syntax als es die (Spezifikations-)Polizei erlauben sollte!

Tolerieren stillschweigend bestimmte Syntaxfehler. Stellen das Dokument trotzdem ‘richtig’ dar!

Folgenden Hex-Darstellungen eines Strings sind lt. Spezifikation offiziell erlaubt + gleichwertig:

```
<45696e20537472696e67>
                    (ohne Whitespace)
<45 69 6e 20 53 74 72 69 6e 67>
                    (je 1 Space zwischen Hex-Paaren)
<45 69    6e 2053 747269    6e      67>
                    (beliebig viele Spaces dazwischen)
```

Nochmals Strings: per Adobe PDF-Software. . .

Adobe-Software toleriert aber auch dieses:

```
<4   5 6 96e 205 3 747   269      6 e    67>
```

(Spaces auch zwischen *'Nibbles'*. . .)

. . . sogar folgendes darf man Acrobat und Reader zum Fraß vorsetzen und es geht ungestreift durch ihren Verdauungstrakt:

```
<4 5  6
   96e
  2
0

5    3  747
2696e67>
```

Also 'wilde' Whitespace-Variationen (inkl. LINE FEEDs, CARRIAGE RETURNs und HORIZONTAL TABs) innerhalb eines Hex-encoded Strings!

Und nochmals. . .

Diesselbe Toleranz auch bei Verwendung von String-*Literalen* :

Adobe-Software toleriert beliebig viele LINE FEEDs innerhalb von Strings. Beispiel:

```
(
\
\
E\
i\
\
\
\
n\
 S\
tri\
n\
g\
)
```

DEMO TIME !

```
open    pdf-syntax/encode-hex-with-space.pdf  # Öffnen mit Mac OSX Preview.app
gs      pdf-syntax/encode-hex-with-space.pdf  # Öffnen mit Ghostscript
mupdf   pdf-syntax/encode-hex-with-space.pdf  # Öffnen mit MuPDF
xpdf    pdf-syntax/encode-hex-with-space.pdf  # Öffnen mit XPDF
adropen pdf-syntax/encode-hex-with-space.pdf  # Öffnen mit Adobe Reader XI
acropen pdf-syntax/encode-hex-with-space.pdf  # Öffnen mit Acrobat Pro XI
```

(Datei `encode-hex-with-space.pdf` ist BSD-lizenziert
und (c) 2011 *Ange Albertini / Corkami*)

Das Kreuz mit der PDF-Syntax (II)

Viele Möglichkeiten also, dasselbe auszudrücken.

Das hat schwerwiegende Konsequenzen:

- Viele PDF-generierende Applikationen decken nicht alle Möglichkeiten ab. (Eher harmlos...)
- Viele PDF-verarbeitenden Applikationen verstehen nicht alle Möglichkeiten. (Oft sehr ärgerlich...)
- **Mehrdeutigkeit öffnet gezieltem Mißbrauch Tür & Tor!** (Sicherheits-Probleme *èn masse* in Acrobat-/AdobeReader)

It's DEMO TIME again!

=> Google-Suche nach "pocorgtfo spoilers" und nach "corkami"

DEMO TIME !

PoCorGTFO06.pdf ist eine *Matroschka*-PDF:

```
mkdir pocor
cd pocor
wget http://pocorgtfo.thequux.com/pocorgtfo06.pdf

for i in {6..1}; do
    unzip pocorgtfo0${i}.pdf;
done
```

Jetzt hat man aus zuerst ***EINER*** PDF plötzlich ***SECHS*** gemacht!
(man hat alle Ausgaben von v1 bis v6 erhalten)

MORE DEMO TIME...

```
open     PoCorGTFO03.pdf
ln -s    PoCorGTFO03.pdf PoCorGTFO03.jpeg
cp       PoCorGTFO03.pdf PoCorGTFO03.zip
ls -l    PoCorGTFO03.*
md5sum   PoCorGTFO03.*
display  PoCorGTFO03.jpeg
unzip -l PoCorGTFO03.pdf
unzip    PoCorGTFO03.pdf
```

Credits to Ange Albertini (Twitter: @corkami / @angealbertini)

Er nennt solche Dateien “Schimären” und “Polyglots”...

No more thing

THE END

Bitte Feedback an mich per eMail...