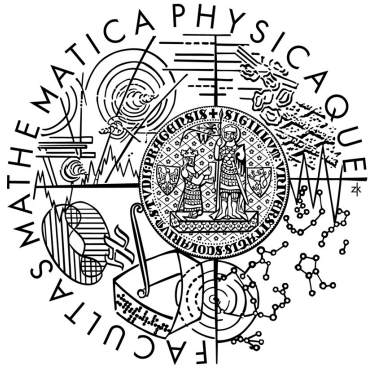
Univerzita Karlova v Praze
Matematicko-fyzikální fakulta

BAKALÁŘSKÁ PRÁCE



Jonáš Bujok

Nástroj pro převod PDF na text

Ústav formální a aplikované lingvistiky (207. • 32-UFAL)

Vedoucí bakalářské práce: Mgr. Jan Raab

Studijní program: Informatika
Studijní obor: obecná informatika (IOI)

Praha 2011

Chtěl bych poděkovat mému vedoucímu bakalářské práce, panu Mgr. Janovi Raabovi, za ochotu, rady a vstřícnost, které mi projevoval v průběhu psaní programu a této práce.

Dále bych chtěl poděkovat mému kolegovi Mgr. Jánovi Dupéjovi za trpělivost s mými neustálými dotazy ohledně MFC, C++, Unicode, Visual Studia a mnoho dalších témat a pomoc při testování výsledného programu.

A v neposlední řadě jsem také vděčný kolegovi Peterovi Miniarovi za pomoc s testováním a přenosem programu do linuxového prostředí.

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně a výhradně s použitím citovaných pramenů, literatury a dalších odborných zdrojů.

Beru na vědomí, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorského zákona v platném znění, zejména skutečnost, že Univerzita Karlova v Praze má právo na uzavření licenční smlouvy o užití této práce jako školního díla podle § 60 odst. 1 autorského zákona.

V Praze dne ..... podpis

Název práce: Nástroj pro převod PDF na text

Autor: Jonáš Bujok

Katedra / Ústav: Ústav formální a aplikované lingvistiky (32-UFAL)

Vedoucí bakalářské práce: Mgr. Jan Raab, Ústav formální a aplikované lingvistiky (32-UFAL)

Abstrakt: V této práci je podrobně rozebrán postup extrakce textových informací z PDF (Portable Document Format) souborů a navrhnut, popsán a implementován program pro tento účel. Práce se zaměřuje hlavně na středoevropské jazyky. Kromě programu a jeho popisu jsou zde pak informace o objektové struktuře, syntaxi a logice PDF formátu nutné pro správné pochopení principu hledání textu v PDF souboru. Dále jsou zde rozebrány filtry, fonty a všechny další PDF objekty, které takový program musí umět zpracovat. Také se tato práce zabývá metodami a možnostmi vylepšení funkčnosti, rychlosti, paměťové náročnosti, spolehlivosti a univerzálnosti použití programu.

Klíčová slova: PDF, text, převodník

Title: A Tool for Transformation of PDF to Text

Author: Jonáš Bujok

Department: Institute of Formal and Applied Linguistics (32-UFAL)

Supervisor: Mgr. Jan Raab, Institute of Formal and Applied Linguistics (32-UFAL)

Abstract: In this thesis we described an extraction procedure of text information from PDF (Portable Document Format) files. Thesis is focused mainly on middle-Europe languages. We designed, described and implemented program for this purpose. Besides the program and it's description the thesis contains information about PDF format object structure, it's syntax and logic necessary for proper understanding of text searching principles in PDF file. We also discussed filters, fonts and all other PDF Objects that the program need to process. This thesis also deals with methods and possibilities of improving program's functionality, speed, memory usage, reliability an universality of usage.

Keywords: PDF, text, convertor

Obsah

Předmluva

Vážení čtenáři této bakalářské práce, rád bych se s Vámi v tomto krátkém úvodu podělil o mé myšlenky a motivace pro vytvoření práce na tak zdánlivě jednoduché téma, jako je získání textu z PDF souboru. PDF soubory jsou v dnešní době velmi rozšířené a používají se hojně v počítačích, mobilních zařízeních i tiskárnách. Ovšem programů a knihoven, které s PDF pracují, je relativně málo na to, jak je tento formát populární. Sice se jejich počet zvětšuje, ale v době, kdy vznikala tato práce ještě ve formě ročníkového projektu, se jen těžko hledal program, který by dobře zvládl takový základní úkon, jako je získat z PDF souboru text. Programy, které byly k dispozici, měly různé problémy a často nevygenerovaly správně většinu speciálních znaků neanglických jazyků včetně českých. Dokonce i Acrobat Reader společnosti Adobe měl v dřívějších verzích s tím problém. Jako hlavní důvod bych viděl velikou složitost tohoto formátu. Což lze ostatně poznat už z toho, že jeho dokumentace (PDF verze 1.7 z roku 2006) má přes 1300 stran. (Podrobněji se na téma složitosti vyjadřuji v první kapitole této práce.) Dnes se situace o něco změnila a existují již celkem spolehlivé programy pro extrakci textu ze souborů tohoto formátu. Některé budou pravděpodobně i mnohem lepší, než program, který jsem napsal já, avšak tato práce může někomu dobře posloužit (a taky doufám, že poslouží) jako návod a shrnutí toho, co je k tomuto účelu potřeba vědět a naprogramovat.

1. PDF soubor

V tomto oddíle jsem se podrobněji zaměřil na PDF soubor a jeho formát. Snažil jsem se o shrnutí všech potřebných informací pro získávání textu a nastínění struktury i syntaxe PDF souboru, aby čitatel nemusel pro tento účel číst anglickou specifikaci, je ale zřejmé, že v některých případech tento kratší popis plnohodnotnou specifikaci zcela nezastoupí.

1.1. O PDF

Vznik PDF souboru se datuje někde kolem roku 1990. Autorem tohoto formátu je společnost Adobe Systems Inc. a vznikl jako náhrada jejich dřívějšího PostScript popisovacího jazyka. PDF prakticky na tomto formátu staví a přidává nové možnosti jako jsou např. interaktivní navigace či struktura dokumentu. Hlavním přínosem je nezávislost na hardwaru i softwaru, která zajišťuje, že libovolný dokument vypadá stejně na všech zařízeních.

PDF patří do skupiny PDL (Page Description Language) jazyků což jsou, jak již samotný název napovídá, jazyky, které popisují, jak má stránka vypadat. Pro porovnání – například markup jazyky ze skupiny DDL (Data Description Language), jako je HTML, RTF a další, jsou jazyky, které popisují význam textu (nebo obecněji - nějakých dat). Textová (či datová) informace je zde klíčová a není až tak důležité, jak je prezentována. Kdežto v PDF samotný význam textu a jiných objektů (např. obrázků) není podstatný, hlavní je, jak se má text či objekt zobrazit. To je taky důvod, proč není získání textové informace z PDF souboru až tak triviální úkol. Mohou existovat dokonce textové PDF soubory vytvořené tak, že nebude možné programaticky zjistit text jinou metodou než použitím OCR (Optical Character Recognition).

Vzhledem velkému rozšíření PDF souborů jak v soukromém, tak i ve vládním sektoru, se 29. ledna roku 2007 společnost Adobe Systems Inc. rozhodla vydat úplnou specifikaci PDF pro účel standardizace a publikace formátu organizací ISO (International Organization for Standardization). O rok později v lednu 2008 byla dokumentace schválena jako mezinárodní standard ISO 32000-1. PDF je tedy dnes již standardizovaný formát a tato specifikace je volně dostupná ke stažení přímo ze stránek Adobe.

1.2. Syntaxe

Tato kapitola nám pomůže pochopit, jaké jsou základní pravidla a konvence v PDF souboru a co jsou jeho objekty.

1.2.1. Lexikální konvence

PDF je sice binární soubor, ale často binární jsou pouze Stream objekty (viz. 1.2.2.7 Stream Object) a zbytek souboru je textový a používá běžně známou ASCII sadu znaků. Není to ale pravidlo, výjimkou je např. zašifrovaný soubor, který je binární téměř celý. Za PDF objekt se považuje posloupnost bajtů (znaků) specifikovaná pravidly syntaxe v následující kapitole o objektech.

Objekty a části objektů jsou odděleny bílými znaky. PDF může obsahovat bílé znaky vypsané v tabulce níže (neplatí pro streamy).

Tabulka 1.1: Seznam bílých znaků

Desítkové	**Hexadecimální**	**Název**
0	00	Null (NUL)
9	09	Horizontal Tab (HT)
10	0A	Line Feed (LF)
12	0C	Form Feed (FF)
13	0D	Carriage Return (CR)
32	20	Space (SP)

Znaky CR ($0D_h$) a LF ($0A_h$) jsou považovány za značky konce řádku. Navíc kombinace znaku CR, bezprostředně následovaný znakem LF, se považuje jako jedna značka konce řádku. Značky konce řádku zkráceně nazýváme EOL (end of line).

Kromě bílých znaků a EOL značek PDF ještě definuje skupinu oddělovačů, jako znaky, které syntakticky oddělují objekty. Mezi oddělovače patří znaky `(`, `)`, `<`, `>`, `[`, `]`, `{`, `}`, `/` a `%`. Tyto znaky ukončují objekt před ním, ale k objektu se nepočítají (opět neplatí pro streamy).

PDF soubor může obsahovat také komentáře. Jsou to části souboru, které mohou být klientskou aplikací ignorovány. Komentáře jsou uvozeny znakem `%` a končí značkou EOL - tedy komentář je vždy od výskytu znaku `%` až do konce řádky. Komentáře se nemohou vyskytovat v řetězci (1.2.2.3 Řetězec) a ve streamu (1.2.2.7 Stream Object). Tam je znak `%` chápán jako součást řetězce či streamu.

Všude jinde by aplikace měla považovat celý komentář jako jeden bílý znak. Kromě komentářů `%PDF-x.x` a `%%EOF` (viz. 1.3 Struktura souboru) nemají žádný sémantický význam.

1.2.2. PDF Objekty

PDF dokument se skládá z objektů a v dalších podkapitolách si rozebereme jak tyto objekty vypadají, jakou mají sémantiku a syntaxi.

1.2.2.1. Booleovský objekt

(Anglicky Boolean Object) Jednoduchý objekt reprezentující booleovskou hodnotu. Může to být buď `true` nebo `false`. Slouží pro různé účely - např. jako návratová hodnota PostScriptových funkcí. Pro náš účel však nemá příliš význam, jen je třeba s jeho možným výskytem počítat při parsování různých jiných objektů.

1.2.2.2. Číselný Objekt

(Anglicky Number Object) Mohou být dvojího druhu – celočíselné a reálné (resp. aproximace reálných čísel plovoucí řadovou čárkou). Celá čísla jsou v souboru uložena jako jedna, nebo více desítkových číslic za sebou, volitelně mohou být se znaménkem před. Příklady:

```
15 42133 +123 -65 0
```

Pokud číslo přesáhne rozsah běžné integer hodnoty v konkrétní implementaci, tak je převedeno na reálné. Reálná čísla mohou mít navíc oproti celým ještě desetinnou tečku (ne čárku). Tečka může být na začátku, uprostřed nebo na konci čísla. Příklady:

```
26.5 -12.66 +78.456 .45 5. 0.0 -.58
```

Není ale podporován PostScriptový zápis typu `16#FFFE`, ani exponenciální typu `32.5E10`. Pro další účely této práce není třeba rozlišovat tyto dva (reálné a celočíselné) druhy číselných objektů. Můžeme vše považovat za reálný číselný objekt.

1.2.2.3. Řetězec

(Anglicky String Object) Řetězec je chápán jako posloupnost bajtů (hodnot od 0 do 255). Význam samotného řetězce určuje až jeho umístění a použití, případně kódování. Jsou li řetězce v Textovém Objektu, pak jednotlivé bajty řetězce představují kódy znaků v aktuálním Fontu (viz. 1.4.4 Textový objekt). Délka řetězce

v Obsahovém Streamu (1.4.3 Obsahový Stream a Zdroje) je omezena na 32767B, v ostatních částech souboru není omezena. Řetězce v PDF souboru jsou také dvojího druhu: Písemné Řetězce a Hexadecimální Řetězce.

1.2.2.3.1 Písemný řetězec

Je to posloupnost znaků uzavřena mezi znaky '(' a ')'. Jakékoliv znaky se mohou vyskytovat v řetězci, kromě nevyvážených závorek a zpětného lomítka, které vyžadují speciální zacházení. Příklady platných řetězců:

```
(Toto je retezec)
(Retezec muze obsahovat
vice radku)
(retezec muze obsahovat vyvazene zavorky () a
specialni znaky (*!$&% atd.). )
(Nasleduje prazdny retezec)
()
```

Zpětné lomítko se používá pro vložení dalších speciálních znaků jako jsou například netisknutelné ASCII znaky, značky EOL, nevyvážených závorek nebo samotného zpětného lomítka. Znak, který následuje přímo za lomítkem, určuje jeho význam. Přehled možných znaků je v následující tabulce:

Tabulka 1.2: Přehled funkcí zpětného lomítka v řetězci

Znaky	**Význam**
`\n`	Line Feed
`\r`	Carriage Return
`\t`	Horizontal Tab
`\b`	Backspace
`\f`	Form Feed
`\(`	Levá závorka
`\)`	Pravá závorka
`\\`	Zpětné lomítko
`\`*`ddd`*	Kód znaku ddd (osmičková soustava)

Je li zpětné lomítko na konci řádku před EOL, pak lomítko a EOL nejsou považovány jako součást řetězce a řetězec pokračuje na dalším řádku. Jinak EOL patří do řetězce a není rozdíl, jestli je v řetězci například přímo Line Feed, nebo kombinace `\n`. Příklady:

```
(Tyto dva \
retezce jsou \
stejne)
(Tyto dva retezce jsou stejne)
```

```
(A tyto dalsi dva taky.
)
(A tyto dalsi dva taky.\n)
```

Kombinace `\ddd` se používá pro reprezentaci znaků mimo ASCII znakovou sadu. Číslo `ddd` je v osmičkové soustavě a určuje kód znaku (podle aktuálního kódování – viz. 2 Extrakce textu), který se má použít. Může být jedno, dvou anebo tři ciferné, takže např. `\5`, `\05` a `\005` označují stejné znaky. Pokud je ale bezprostředně za kódem nějaký numerický znak, pak jsou úvodní nuly nutné pro rozpoznání správného kódu. Čili `\053` není to samé jako `\0053`. V prvním případě se jedná o znak s kódem 53 a ve druhém o znak s kódem 5 následovaný číslovkou '3'.

1.2.2.3.2 Hexadecimální řetězec

Hexadecimální řetězec je uzavřen mezi znaky '<' a '>'. Je to posloupnost hexadecimálních číslic (znaků 0-9 a buď A-F nebo a-f). Řetězec často reprezentuje nějaká binární data. Pro náš účel nejčastěji bude reprezentovat kódy znaků. Každý pár hexadecimálních čísel určuje jeden bajt řetězce. Bíle znaky v řetězci jsou ignorovány. Pokud se řetězec skládá z lichého počtu číslic, takže v posledním páru jedna číslice chybí, pak je jako poslední číslice považována nula. Příklad:

```
<8236F4A>
```

Řetězec v příkladu má 4B a obsahuje bajty 82h, 36h, F4h a A0h.

1.2.2.4. Název

(Anglicky Name Object) Název je objekt, který reprezentuje určitou identifikaci v souboru. Je to posloupnost znaků uvozena znakem '/'. Znak '/' se do názvu nepočítá – pouze jej uvozuje. Název v PDF se používá k mnoha účelům, jako jsou např.: pojmenování fontů, slovníků fontu, kódování atd... Je unikátní v celém souboru a atomický. Unikátní znamená, že dva objekty vytvořené ze stejných znaků znamenají tentýž název. Atomický znamená, že (i přesto, že se skládá z posloupnosti znaků,) nemá žádnou vnitřní strukturu. Může obsahovat jakékoli znaky kromě oddělovačů a bílých znaků (viz. 1.2.1 Lexikální konvence). V názvu záleží na velikosti písmen, takže např. názvy `/b` a `/B` jsou různé. V názvu mohou být speciální znaky, bílé znaky a oddělovače v následujícím formátu. Speciální znaky se zapisují do názvu jako dvouciferné hexadecimální číslo, před kterým je znak '#' - např. `#33` je znak '!', `#23` je znak '#', atd. Pokud je potřeba určit délku názvu, pak je nutno vědět, že znak zadaný pomocí hexadecimální číslice se počítá jako jeden a ne tři znaky.

Proto název `/A#20B` („A B“) má délku tři a ne pět. Názvy slouží pro interní identifikaci entit v souboru a ne pro prezentaci člověku, proto většinou není nutné vědět, jaké kódování je použito pro znaky. Pokud ale v nějakém případě je třeba vědět, jak název interpretovat, pak je doporučeno použít UTF-8 kódování.

1.2.2.5. Pole

(Anglicky Array Object) Pole v PDF je posloupnost PDF objektů uzavřených mezi znaky '[' a ']'. Pole je heterogenní, jedno pole může tedy obsahovat objekty různých typů (řetězce, čísla, slovníky, další pole atd.). Pole může být také prázdné. Příklad pole:

```
[5.12 /Gustav 2 [a b c] (ma rad vdolky)]
```

1.2.2.6. Slovník

(Anglicky Dictionary Object) Slovník je struktura pro přiřazení PDF objektů ke jménům. Je to posloupnost dvojic objektů uzavřená mezi << a >>. První z dvojce se nazývá klíč a je to vždy PDF objekt typu Název. Druhý objekt nazýváme hodnota a může to být jakýkoli PDF objekt, včetně dalšího slovníku. Slovník má tedy následující tvar:

```
<<
  klíč1 hodnota1
  klíč2 hodnota2
    …
  klíčn hodnotan
>>
```

Konkrétní příklad může vypadat např. takto:

```
<<
/Typ /Priklad
/Podtyp /PrikladSlovniku
/Verze 0.01
/CiselnaHodnota 12
/Retezec (ten nejlepsi priklad)
/Podslovník <<    /Polozka1 0.4
                  /Polozka2 true
                  /Posledni polozka (ne!)
                  /UplnePosledniPolozka (OK)>>
>>
```

1.2.2.7. Stream Object

Zde jsem si dovolil nechat objekt bez překladu, protože žádný český ekvivalent anglického stream mi nepřipadá vhodný. Stream objekt patří ke složitějším objektům v PDF, ale neobejdeme se bez něj. Jedná se, stejně jako v

řetězci, o posloupnost bajtů s tím rozdílem, že na stream mohou být aplikovány různé transformace (např. kompresní metody) a není omezena jeho délka. Stream objekt se skládá ze slovníku, za kterým je mezi klíčovými slovy `stream` a `endstream` posloupnost bajtů. Stream se používá většinou pro potenciálně velká data (obrázky, text, popisy stránek aj.) a sám o sobě nemá význam. Význam streamu určuje jeho kontext, ve kterém je použit. Všechny streamy musí být vždy nepřímé objekty (viz. 1.2.2.8 Nepřímý Objekt) a slovník ve streamu musí být vždy přímý objekt. Za klíčovým slovem `stream` může být ještě CR+LF nebo LF, ale ne CR samostatně. Před klíčovým slovem `endstream` se může také nacházet EOL značka. Slovník streamu obsahuje délku dat mezi slovy `stream` a `endstream`. Do této délky se nepočítají EOL značky před a za daty.

Od verze PDF 1.2 mohou být data i v externím souboru. Je-li tomu tak, poznáme ze slovníku streamu (viz Tabulka 1.3). Jsou v něm také informace potřebné k získání dat. Data mezi klíčovými slovy `stream` a `endstream` mohou být v takovém případě ignorována.

Transformace dat, o kterých jsem psal výše, se v PDF specifikaci nazývají Filters, proto dále v textu považujte transformace streamu a filtr streamu za totéž. Podrobněji konkrétním filtrům se věnuji v sekci 1.2.2.7.1 Filtry. Následující tabulka obsahuje seznam položek slovníku, které jsou společné pro stream objekty. Slovníky některých konkrétních stream objektů mohou mít i jiné položky, tyto však může mít každý stream objekt.

Tabulka 1.3: Položky slovníku Stream Objektu

Klíč	**Hodnota**	**Popis**
Length	Celé číslo	(Povinné) Délka dat mezi klíčovými slovy `stream` a `endstream` bez EOL značek před a za daty. Je-li stream v externím souboru pak je tato položka zbytečná, ale stále povinná a musí obsahovat délku dat mezi kl. slovy i když to bude nejčastěji 0.
Filter	Název nebo Pole	(Nepovinné) Název filtru nebo pole s názvy filtrů, které se mají použít na data pro získání dekódovaných originálních dat. V případě pole jsou filtry v pořadí v jakém se mají aplikovat.

Klíč	Hodnota	Popis
DecodeParams	Slovník nebo Pole	(Nepovinné) Slovník s parametry pro dekódování pro použitý filtr nebo pole takových slovníků pro každý filtr, je-li jich použito více. Pokud všechny filtry mají své standardní nastavení, nebo nejsou použity žádné filtry, pak může být položka vynechána.
F	Specifikace souboru	(Nepovinné) Externí soubor, který obsahuje data.
FFilter	Název nebo Pole	(Nepovinné) Filtr nebo filtry pro data v externím souboru. Platí stejná pravidla jako pro položku **Filter**.
FDecodeParams	Slovník nebo Pole	(Nepovinné) Parametry pro filtry pro data v externím souboru. Platí stejná pravidla jako pro položku **DecodeParams**.
DL	Celé číslo	(Nepovinné) Až od verze PDF 1.5. Nenulové celé číslo určující délku dat po aplikaci všech filtrů. Toto číslo by mělo být bráno čistě jako orientační. Není zaručena jeho přesnost.

1.2.2.7.1 Filtry

PDF formát podporuje 10 různých filtrů pro Stream Objekty (přesný seznam viz. Tabulka 1.4). Tyto filtry mají různé vlastnosti a smysl, avšak samotný formát nebrání použití jakýchkoli těchto filtrů na jakýkoliv stream. Tím chci říct, že i přes to, že se v naprosté většině případů jako filtr pro náš účel používá FlateDecode, neznamená, že nebude třeba pro některá PDF pro stejný účel použít i ostatní filtry. Čistě teoreticky – PDF nebrání ani použití komprese pro obrázky na text i když v praxi by to nemělo smysl. Které filtry a s jakými parametry použít na stream objekt nám určuje slovník v něm. V položce Filter (nebo FFilter pro externí data) máme jeden nebo více filtrů v poli v pořadí, jak se mají aplikovat. Například máme-li ve slovníku takovou položku:

```
/Filter [/ASCIIHexDecode /LZWDecode]
```

Pak je třeba data ve streamu nejdříve dekódovat pomocí ASCIIHexDecode a následně pomocí LZWDecode, abychom dostali originální data.

Tabulka 1.4: Seznam filtrů Stream objektu

Název	Parametry	Popis
ASCIIHexDecode	Ne	Dekóduje data v ASCII Hexadecimální reprezentaci. Používá se pro binární data.

Název	Parametry	Popis
ASCII85Decode	Ne	Dekóduje data v ASCII base-85 reprezentaci. Používá se pro binární data.
LZWDecode	Ano	Dekomprimuje data zakódovaná LZW (Lampel-Ziv-Welch) adaptivní kompresní metodou. Používá se pro text i binární data.
FlateDecode	Ano	Dekomprimuje data zakódovaná zlib/deflate kompresní metodou. Používá se pro text i binární data.
RunLengthDecode	Ne	Dekomprimuje data zakódovaná run-length kompresní metodou. Používá se pro text i binární data. (Typicky pro monochromatické obrázky nebo jiná data s dlouhými výskyty stejných bajtů.)
CCITTFaxDecode	Ano	Dekomprimuje data zakódovaná CCITT (Huffman) kompresní metodou. Používá se pro binární data. (Typicky pro monochromatické obrázky.)
JBIG2Decode	Ano	Dekomprimuje data zakódovaná JBIG2 kompresní metodou. Používá se pro monochromatické obrázky (1 bit na pixel) nebo pro aproximaci těchto dat.
DCTDecode	Ano	Dekomprimuje data zakódovaná DCT (Diskrétní Kosinusovou transformační) technikou založenou na JPEG standardu. Používá se pro obrázky, reprodukuje aproximaci originálních dat.
JPXDecode	Ne	Dekomprimuje data zakódovaná waveletovou kompresní metodou založenou na JPEG2000 standardu. Používá se pro obrázky.
Crypt	Ano	Dešifruje data. (problém šifrování je v této práci zcela vynechán)

Nemám prostor v této práci se každou kompresní metodou zabývat zvlášť a navíc pro účel extrakce textu je postačující implementace FlateDecode a v ne příliš častých případech LZWDecode. Pro podrobností ohledně Flate decode může čitatel sáhnout do standardů Internet RFC 1950 a 1951. Pro ostatní filtry například do PDF Reference nebo PDF standardu (viz. Seznam použité literatury).

1.2.2.8. Nepřímý Objekt

(Anglicky Indirect Object) Nepřímé objekty v PDF souboru jsou takové PDF objekty, které mají vlastní identifikátor a pomocí něj mohou být odkazovány z různých jiných objektů. Tedy nenacházejí se přímo na místě, kde mají být použity,

ale na daném místě se nachází pouze odkaz na ně. Samotný objekt pak může být kdekoliv jinde v těle souboru a najít jej lze díky referenční tabulce (viz. 1.3.3 Referenční tabulka (Cross-reference Table)). Jako unikátní identifikátor nepřímého objektu, který se pak používá pro odkazování na něj, slouží dvojce čísel – číslo objektu a generační číslo.

Číslo objektu je kladné celé číslo. Toto číslo je často přiřazováno sekvenčně v PDF souboru, ale není to nutné. Mohou být také v jakémkoli pořadí.

Generační číslo je nezáporné celé číslo. V nově vytvořeném souboru je u všech objektů nulové. Kladná generační čísla vznikají až s úpravami souboru. (viz. 1.3.5 Úpravy souboru)

Samotný nepřímý objekt vypadá následovně. Na začátku je identifikační dvojce čísel oddělena bílým znakem a za ní klíčové slovo `obj`, pak hodnota, tedy nějaký PDF objekt a za ním klíčové slovo `endobj`. Hodnota uvnitř nepřímého objektu může být PDF objekt jakéhokoli typu kromě zase nepřímého objektu. Příklad nepřímého objektu:

```
15 0 obj
(abc)
endobj
```

Má-li někde být použita hodnota nepřímého objektu, pak je na daném místě odkaz na daný objekt. Odkazy na nepřímý objekt mají formát: číslo objektu, generační číslo a písmeno 'R' oddělené bílými znaky. Na nepřímý objekt z příkladu výše by tedy šlo odkazovat kdekoli z dokumentu takto: `15 0 R`.

1.2.2.9. Nulový objekt

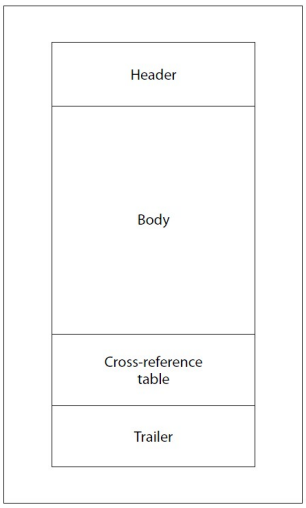
Nulový objekt je jednoduchý objekt, který není ekvivalentní s žádným jiným objektem. Je to prázdný objekt bez významu. Jedná se o klíčové slovo `null`. Odkaz na neexistující nepřímý objekt v souboru se taky považuje za nulový objekt. Je-li použit nulový objekt jako hodnota ve slovníku, pak je to stejné jako by daný klíč, odkazující na `null`, vůbec ve slovníku nebyl.

1.3. Struktura souboru

Pro správné nalezení potřebných objektů je třeba znát strukturu PDF souboru. Soubor je rozdělen do čtyř hlavních částí. (viz. Ilustrace 1)

- Hlavička (ang. Header)
- Tělo (ang. Body)

- Referenční tabulka (ang. Cross-reference Table)
- Patička (ang. Trailer)



Ilustrace 1: Struktura PDF Souboru (Zdroj: PDF Reference)

Nyní bych se rád na tyto čtyři sekce zaměřil podrobněji. Jsou to základy nutné pro práci s PDF souborem.

1.3.1. Hlavička (Header)

Hlavička je nejjednodušší částí souboru. Je to první řádek souboru s označením verze PDF. Má formát `%PDF-x.x`, kde x.x je verze souboru (např. `%PDF-1.5`). V některých souborech bývá za hlavičkou v komentáři několik (čtyři a více) binárních znaků (- tedy znaků s kódy většími než 127) a to kvůli aplikacím, které podle začátku souboru rozpoznávají, zda se jedná o binární nebo textový soubor.

Od PDF verze 1.4 je možné, aby bylo označení verze souboru také v Katalogu dokumentu (viz. 1.4.1 Katalog dokumentu) v položce Version. Je-li tomu tak, pak má označení v katalogu větší váhu, a označení v hlavičce by mělo být ignorováno.

1.3.2. Tělo (Body)

Hlavní část souboru je tělo, které obsahuje všechny nepřímé objekty reprezentující obsah a strukturu dokumentu, stránky, obrázky, text, fonty a všechny další objekty potřebné pro zobrazení PDF, od verze 1.5 pak i Objektové streamy, kterými se v této práci nebudeme zabývat.

1.3.3. Referenční tabulka (Cross-reference Table)

Jedna z nejdůležitějších součástí PDF souboru. Jedná se o tabulku, která odkazuje na všechny použité i nepoužité (ty jsou však odlišeny) nepřímé PDF objekty (1.2.2.8 Nepřímý Objekt). Tabulka obsahuje pro každý (nepřímý) objekt jeden jednořádkový záznam, který se skládá z identifikace, pozice v souboru a příznak je li používán (- podrobněji vysvětleno níže) . Klientská aplikace (a tedy i náš převaděč) musí tuto tabulku načíst, aby mohla dále se souborem a s jeho objekty pracovat. Jak už jsem zmínil výše, v PDF souboru mohou být i nepoužívané objekty (např. v případech, kdy soubor byl upravován). Také proto je nutné tuto tabulku správně načíst a získat z ní seznam použitých nepřímých objektů a jejich pozic v souboru.

V předchozím odstavci jsem sice mluvil o tabulce v jednotném čísle, v praxi ale tato referenční tabulka v PDF souboru může být rozdělena do několika sekcí (, každá se může nacházet jinde v souboru) a jednotlivé sekce mohou být rozděleny do podsekcí a až v nich máme ony podstatné záznamy o PDF objektech v souboru. Běžně na začátku (po vytvoření) mají PDF soubory pouze jednu sekci (případně dvě pokud jsou linearizované – tím se nebudeme a ani nemusíme zabývat). Další sekce přibývají až s úpravami dokumentu (viz. 1.3.5 Úpravy souboru). Nutno ještě říci, že celá tabulka je textová, tedy není binární a čísla v ní jsou „textové“ a v desítkové soustavě.

Každá sekce začíná klíčovým slovem `xref` a za ním následuje jedna nebo více podsekcí. Běžně v PDF, který nebyl nikdy upravován, má každá sekce jen jednu podsekci. Pokud má jich víc, jsou postupně všechny za sebou, takže není problém je načíst všechny. Z předchozích kapitol víme, že každý nepřímý objekt má jako svou identifikaci dva čísla. První je číslo objektu a druhé je generační číslo (viz. 1.2.2.8 Nepřímý Objekt). Tabulka (, konkrétněji všechny její podsekce,) nám tedy dává pro správnou identifikaci obě tato čísla. Každá podsekce má na prvním řádku dva čísla, kde první číslo je číslo prvního objektu (patřícímu prvnímu záznamu) v podsekci a druhé číslo je počet objektů v dané podsekci. Např. pokud první řádek podsekce vypadá takto

```
32 10
```

pak podsekce obsahuje 10 záznamů a to pro objekty s čísly 32 až 42.

Nyní přejděme k samotným záznamům v tabulce. Každý záznam je na jednom řádku a má velikost 20B. Záznamy jsou dvojího druhu: pro objekty, které

jsou použity a pro nepoužité objekty.

- Formát záznamů pro použité objekty je následující

  `nnnnnnnnnn ggggg` **`n`** `eol`

  kde `nnnnnnnnnn` je 10-ti místné (desítkové) číslo vyjádřující pozici objektu v souboru v bajtech od jeho začátku, `ggggg` je generační číslo objektu, **`n`** označuje, že je objekt používán a nakonec je značka konce řádky.

- Formát záznamů pro nepoužívané (tzv. volné) objekty je následující

  `nnnnnnnnnn ggggg` **`f`** `eol`

  kde `nnnnnnnnnn` je 10-ti místné číslo dalšího volného objektu, `ggggg` je generační číslo, **`f`** označuje, že objekt není používán a nakonec je opět značka konce řádky. Objekt s číslem 0 je vždy volný objekt a všechny záznamy volných objektů jsou čísly `nnnnnnnnnn` propojeny do uzavřeného spojového seznamu. Tedy poslední záznam s volným objektem v tabulce odkazuje zpět na záznam objektu 0.

Pokud je použit dvoubajtový `eol` (CR+LF) pak už před ním není mezera (space), aby celý záznam měl, jak již bylo výše psáno, přesně 20B. Volné objekty nejsou pro náš účel podstatné, proto se jimi už nebudeme dále zabývat. Příklad referenční tabulky (konkrétněji jedné sekce se dvěmi pod-sekcemi):

```
xref
0 6
0000000002 65535 f
0000000121 00000 n
0000000005 00002 f
0000000239 00001 n
0000001023 00000 n
0000000000 00001 f
15 2
0000005960 00000 n
0000006231 00000 n
```

Od verze 1.5 PDF souboru mohou být některé, nebo všechny, informace referenční tabulky také uloženy v referenčním streamu (ang. Cross-Reference Stream). Jeho syntaxe je stejná jako syntaxe Stream Objektu (1.2.2.7 Stream Object). Stream obsahuje informace ekvivalentní k informacím referenční tabulky, avšak v trochu jiném formátu. Referenčním streamem se nebudeme dále v této práci zabývat. Pro zájemce je podrobně popsán v anglické dokumentaci (viz. Seznam použité literatury).

1.3.4. Patička (Trailer)

Patička (jak jsem to volně přeložil) je část na konci souboru, která umožňuje rychle najít referenční tabulku a další klíčové objekty a informace. Aplikace by proto měla začít číst PDF soubor od konce. Poslední řádek v souboru obsahuje značku konce souboru (`%%EOF`), a předchozí dva řádky obsahují klíčové slovo `startxref` na prvním řádku a pozici (v bajtech od začátku souboru) poslední sekce referenční tabulky v souboru (- pozici klíčového slova `xref`) na druhém. Před řádkem s klíčovým slovem `startxref` se nachází slovník (viz. 1.2.2.6 Slovník) uvozený klíčovým slovem `trailer`. Patička PDF souboru vypadá tedy kupříkladu takto:

```
trailer
<<
  klíč1 hodnota1
  klíč2 hodnota2
    …
  klíčn hodnotan
>>
startxref
pozice poslední sekce ref. tabulky
%%EOF
```

V následující tabulce (Tabulka 1.2) jsou popsány některé hodnoty a klíče, které se mohou nacházet v slovníku patičky a jsou pro náš účel něčím významné. Pro přehled všech klíčů, které může slovník v patičce obsahovat, je možno nahlédnout do PDF standardu ISO 32000-1.

Tabulka 1.5: Položky slovníku v patičce

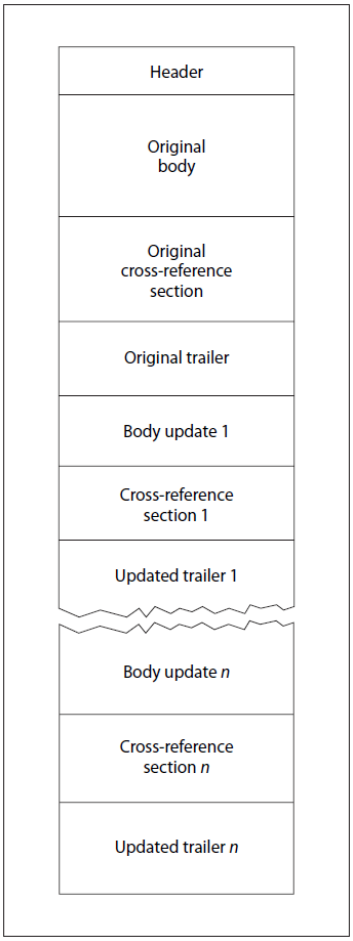
Klíč	**Hodnota**	**Popis**
Size	Celé číslo	(Povinné, musí být přímý objekt) Počet všech záznamů ve všech sekcích referenční tabulky. Může posloužit pro kontrolu. Toto číslo by mělo být také o jedna větší než nejvyšší číslo objektu v PDF souboru.
Prev	Celé číslo	(Přítomné pouze, je-li v souboru více sekcí referenční tabulky, musí být přímý objekt) Pozice předchozí sekce referenční tabulky v bajtech od začátku souboru.
Root	Slovník	(Povinné, musí být nepřímý objekt) Slovník katalogu dokumentu (1.4.1 Katalog dokumentu)

1.3.5. Úpravy souboru

Sice u většiny lidí panuje chybný názor, že PDF soubor se nedá upravovat, opak je ale pravdou. PDF soubor může být upravován a jeho formát i struktura je na to připravená. Existují ale určitá pravidla, jak se úpravy provádí a pokud je soubor

podepsán, pak po provedení úprav nepovoleným způsobem, bude podpis zneplatněn.

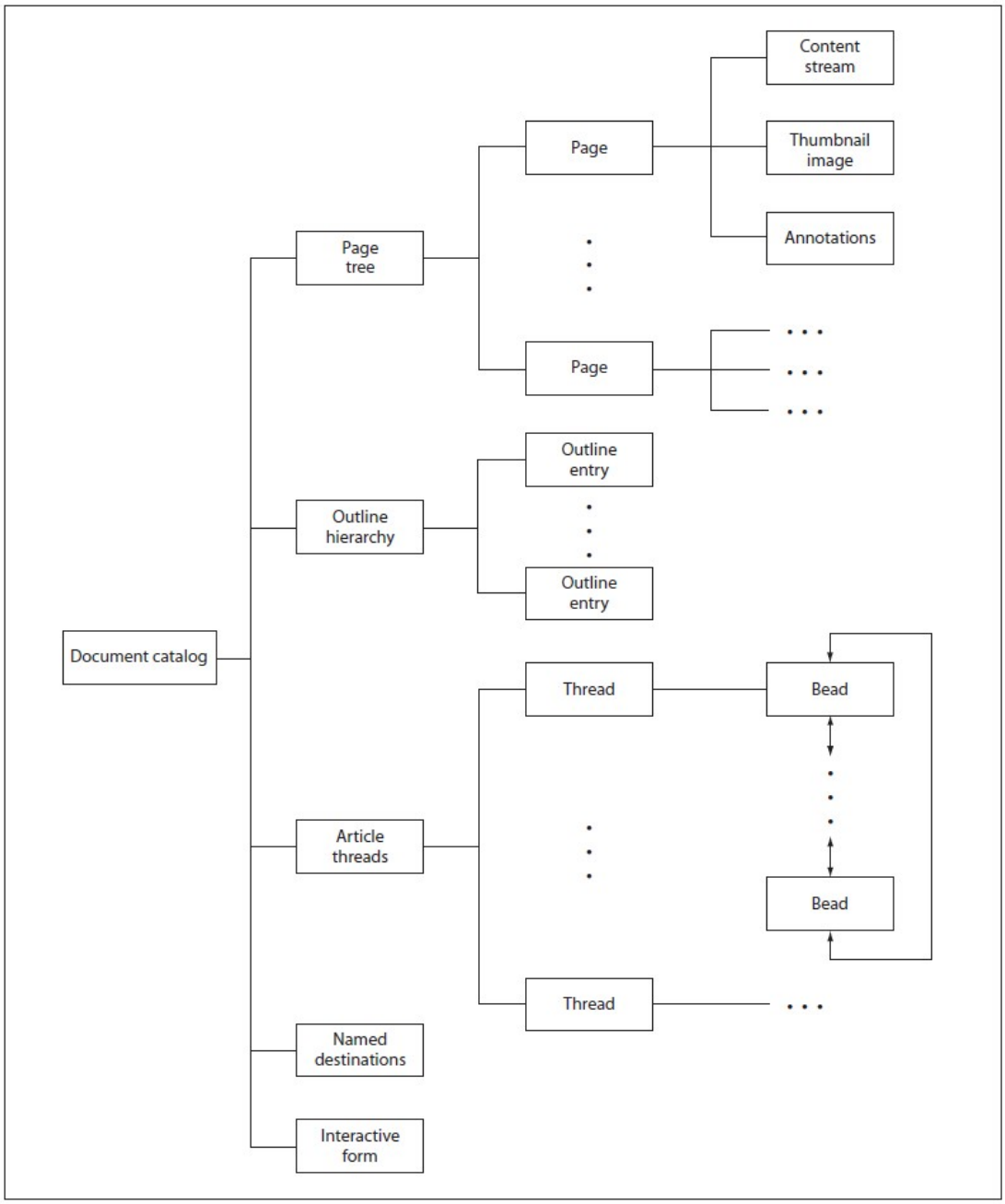
Pro náš účel ale nepotřebujeme vědět, jak se soubor upravuje a zabezpečuje. Musíme pouze vědět, jak se mění struktura souboru a hlavně referenční tabulka upravovaného PDF dokumentu. Úpravy dokumentu jsou navrženy tak, aby se nemuselo zasahovat do originálního dokumentu, ale změny se pouze připojují za něj – tedy za značku konce souboru (`%%EOF`). Přidávají se tam nové a upravené nepřímé objekty (doplnění těla), nová sekce referenční tabulky a nová patička souboru. Nové objekty, které mají nahradit některé stávající, se vytvoří s vyšším generačním číslem, nahradí se odkazy a původní objekt se označí jako volný. Upravovaný dokument má tedy několik částí těl, sekcí referenční tabulky a patiček (viz. Ilustrace 2). U takto upraveného souboru je nutné, aby slovník v patičce obsahoval položku **`Prev`**, která odkazuje na předchozí sekci referenční tabulky a za níž se nachází předchozí patička. Ta může mít zase odkaz na dřívější sekci, patička za ní zase na dřívější... atd. Tímto způsobem můžeme vyhledat všechny sekce referenční tabulky a její záznamy v upravovaném souboru.



Ilustrace 2: Struktura souboru po úpravách (Zdroj: PDF Reference)

1.4. Struktura Dokumentu

Struktura PDF dokumentu je hierarchická a její kořen je Katalog dokumentu podrobněji popsán v sekci 1.4.1. Většina objektů ve struktuře jsou slovníky, které jsou hierarchicky propojeny. Ilustrace 3 ukazuje jak taková struktura vypadá. Pro náš účel nás bude zajímat pouze horní větev – tedy Katalog dokumentu (Document Catalog), Strom Stránek (Page Tree), Stránka (Page) a její obsah.



Ilustrace 3: Struktura dokumentu (Zdroj: PDF Reference)

1.4.1. Katalog dokumentu

Je to kořenová položka v hierarchii PDF dokumentu. Odkaz na něj je přímo z Patičky souboru (viz. 1.3.4 Patička (Trailer)). Jedná se o slovník, který obsahuje různé odkazy na objekty definující dokument a jeho vlastnosti. Obsahuje také informace o tom, jak se má dokument zobrazit, která stránka má být zobrazena, jestli mají být miniatury stránek zapnuty a mnoho dalších informací pro nás nepodstatných. Slovník má proto mnoho položek (cca 30) a nemá smysl je vypisovat zde všechny. Následující tabulka popisuje některé položky katalogu, které by nás mohly nebo měly zajímat.

Tabulka 1.6: Položky Katalogu Dokumentu

Klíč	Hodnota	Popis
Type	Název	(Povinné) Pro katalog dokumentu je hodnota vždy `/Catalog`. Podle této položky poznáme, že se jedná právě o Katalog dokumentu.
Version	Název	(Nepovinné) Tento Název označuje aktuální verzi souboru. Má větší váhu než verze zapsána v hlavičce. (viz 1.3.1 Hlavička (Header))
Pages	Slovník	(Povinné, musí být nepřímý objekt) Kořenový uzel stromu stránek. (viz. 1.4.2 Strom Stránek)

Ostatní položky Katalogu si může čitatel v případě potřeby pročíst v PDF Reference (viz. Seznam použité literatury).

1.4.2. Strom Stránek

Strom stránek se skládá z uzlů stromu stránek (viz. 1.4.2.1 Uzel stromu stránek) a stránek samotných (viz. 1.4.2.2 Stránka). Stránky musí být vždy listy stromu a uzly stromu stránek nesmí být listy stromu. Jak strom vypadá záleží čistě na aplikaci, která PDF vytváří. Může to být klidně kořen a pod ním přímo všech n stránek, může to třeba také být vyvážený binární strom, nebo jakýkoliv obecný strom.

Strom stránek díky své struktuře umožňuje rychlejší procházení a vyhledávání stránek a také i dědičnost některých nastavení či zdrojů (podrobněji viz. 1.4.2.2 Stránka).

1.4.2.1. Uzel stromu stránek

Jsou to vnitřní uzly stromu stránek. Jedná se o slovník a položky, které nutně musí uzel stromu obsahovat, jsou popsány níže v tabulce. Uzel může obsahovat i

některé položky ze Stránky, které jsou dědičné. (viz. 1.4.2.2 Stránka).

Tabulka 1.7: Položky Uzlu stromu stránek

Klíč	Hodnota	Popis
Type	Název	(Povinné) Pro uzel stromu stránek je hodnota vždy `/Pages`. Podle této položky poznáme, že se jedná právě o uzel stromu stránek.
Parent	Slovník	(Povinné v ne-kořenových uzlech, Zakázáno v kořenovém uzlu, musí být nepřímý objekt) Rodičovský uzel ve stromu.
Kids	Pole	(Povinné) Pole Nepřímých objektů, které jsou přímými potomky tohoto uzlu. Potomky mohou být pouze Stránky nebo další Uzly stromu stránek.
Count	Celé číslo	(Povinné) Počet listů (Stránek), které se nacházejí (přímo i nepřímo) pod tímto uzlem. Tedy suma všech stránek v tomto uzlu a všech pod-uzlech.

Příklad, jak může takový uzel vypadat:

```
3 0 obj
<<
      /Type /Pages
      /Parent 2 0 R
      /Kids
      [
            4 0 R
            6 0 R
            10 0 R
      ]
      /Count 12
>>
endobj
```

1.4.2.2. Stránka

Stránka (Anglicky Page Object) je list Stromu stránek. Je to také slovník, jako většina prvků struktury. Podobně, jako Katalog Dokumentu, i tento slovník obsahuje veliké množství položek a nemá smysl je zde všechny vypisovat. Proto jsem do následující tabulky vybral pouze ty podstatné, které se používají při extrakci textu.

Tabulka 1.8: Položky Stránky

Klíč	Hodnota	Popis
Type	Název	(Povinné) Pro Stránku je hodnota vždy `/Page`. Podle této položky poznáme, že se jedná právě o Stránku.
Parent	Slovník	(Povinné, Musí být nepřímý objekt) Rodičovský uzel ve stromu. (viz. 1.4.2.1 Uzel stromu stránek)

Klíč	Hodnota	Popis
Resources	Slovník	(Povinné, dědičné) Obsahuje slovník se zdroji pro stránku (viz. 1.4.3 Obsahový Stream a Zdroje) Pokud stránka žádné zdroje nepotřebuje, pak je zde prázdný slovník. Pokud zde není žádný slovník určen, znamená to, že zdroje mají být zděděny z rodičovského uzlu.
Contents	Stream nebo Pole	(Nepovinné) Obsahový stream (viz. 1.4.3 Obsahový Stream a Zdroje), který popisuje celý obsah stránky nebo pole takovýchto streamů. Pokud není zadán, pak stránka je prázdná. Je-li zadáno pole, je třeba ho považovat jako by to byl jeden stream spojený ze všech obsahových streamu v poli (ve stejném pořadí). Pozor: Streamy mohou být rozděleny kdekoliv – třeba i mezi operandem a operátorem.

1.4.3. Obsahový Stream a Zdroje

1.4.3.1. Obsahový Stream

(Anglicky Content Stream) Obsahový stream je Stream Object (viz. 1.2.2.7 Stream Object), který svými (textovými) daty popisuje s použitím zdrojů, jak má celá stránka dokumentu vypadat.

Poznámka: V dalším textu se budeme zabývat jakýmisi operacemi složených z operandů a operátorů. Zde musím upozornit, že se nejedná o klasickou operaci, která odpovídá přesně matematické definici operace, jako postup, který, na základě vstupních hodnot, vygeneruje výstupní hodnotu. Spíše je třeba tyto operace chápat jako příkaz, který něco provede na základě argumentů. A nyní zpět k tématu.

Obsahový stream je posloupnost operandů a operátorů. Notace je postfixová, takže nejdříve jsou operandy a z nimi operátor, který je zpracuje. Nejedná se ale o zásobníkovou strukturu, protože operace nevrací žádnou hodnotu. Operandy jsou vždy nějaké PDF Objekty (viz. 1.2.2 PDF Objekty) kromě Stream Objektů, nepřímých objektů a odkazů na ně. Operátory jsou určitá klíčová slova, která určují, jaká akce se má provést s operandy. Rozdíl mezi operátorem a Názvem je, že operátor nemá uvozující značku '/'. Operátory mají význam pouze uvnitř Obsahového streamu. Je jich mnoho a opět nemá smysl zde rozebírat všechny. Jen namátkou pro představu vyjmenuji např. w, J, j, M, q, Q W, W*, BI, EI, ', BMC, EMC, S, s ... a další. Nás ale budou v Obsahovém streamu zajímat tyto operátory: BT, ET, Tf, TD, Td, T*, Tj, TJ, ', ".

První dva operátory BT a ET jsou nulární (nepotřebují operandy) a označují

začátek a konec textového objektu. Textový objekt je popsán níže.

Operátor Tf slouží k výběru aktuálního fontu a nastavení velikosti. Tf potřebuje dva operandy. První je název fontového slovníku ze zdrojů stránky a druhý je velikost. Velikost fontu nás nemusí zajímat. Důležité pro nás je, že operátor nastaví aktuální font a všechny následující řetězce se musí vypsat pomocí tohoto fontu, dokud není font znova změněn. Příklad použití: `/F1 12.0 Tf`.

Ostatní operátory výše vyjmenované jsou probrány v sekci 1.4.4 Textový objekt, protože se vyskytují pouze v něm.

1.4.3.2. Zdroje

Opět se jedná o slovník s mnoha položkami, pro nás ale je podstatná pouze jedna a to položka **Font**. Font ve zdrojích je slovník, který mapuje názvy fontů, které používá operátor Tf (viz. 1.4.3.1Obsahový Stream), na jejich Fontové slovníky. Fontové slovníky jsou pak nutné pro správné získání textu z řetězce (podrobněji viz. 1.4.5 Fonty). Příklad, jak může takový zdroj vypadat:

```
<<
/ProcSet [/PDF /ImageB]
/Font <<
      /F3 22 0 R
      /F4 28 0 R
      /F5 35 0 R
      >>
/XObject <</Im1 13 0 R>>
>>
```

1.4.4. Textový objekt

Jak již bylo zmíněno v Obsahovém Streamu, textový objekt je uzavřen mezi operátory BT a ET. A taky platí, že použití operátoru Tf pro nastavení fontu je možné i uvnitř textového objektu a ne jen v obsahovém streamu (viz. 1.4.3.1 Obsahový Stream). Textový objekt je stále posloupnost operací (operandů a operátorů), protože je součástí Obsahového streamu, pouze používá jiné operace. Nyní se podíváme na jednotlivé operace (TD, Td, T*, Tj, TJ, ', ") podrobněji.

T*, TD a Td patří k pozičním operátorům a nastavují aktuální pozici, na které se bude vypisovat text. Některé funkce, které operátory provádějí záměrně nebudu popisovat, abych zbytečně nepopisoval věci, jež se netýkají extrakce textu. Některé operátory, totiž provádějí více úkonů a ne všechny jsou pro nás podstatné. T* nepotřebuje žádné operandy a pouze přesune aktuální pozici na začátek dalšího řádku. Td a TD jsou binární operace (mají dva operandy) a také posouvají aktuální

pozici na nový řádek s tím rozdílem, že podle parametrů nastaví odsazení řádku. Parametry jsou vlastně x a y souřadnice v textovém prostoru o které je třeba aktuální pozici posunout vůči začátku minulého řádku. Jak se toto odsazení přesně počítá nebudu popisovat, protože by to zabralo několik stran a textový výstup, kterého chceme docílit, nemusí mít formátování. Můžeme tedy parametry ignorovat a považovat operátory TD a Td za totožné s T*.

Tj, TJ, ', " patří k zobrazovacím operátorům, které vykreslují řetězce aktuálním fontem. Následující tabulka popisuje jejich funkce.

Tabulka 1.9: Zobrazovací operátory textového objektu

Operandy	**Operátor**	**Popis**
Řetězec	Tj	Unární operace. Zobrazí řetězec na aktuální pozici.

Řetězec	'	Unární operace. Přesune se na další řádek a zobrazí řetězec. Má stejnou funkci jako posloupnost operací:
		`T*`
		`Řetězec Tj`

A B Řetězec	"	Ternární operace. Má stejnou funkci jako operace ' s tím rozdílem, že nastaví podle operandů (čísel) A a B mezery mezi slovy a mezery mezi znaky – což můžeme pro textový export ignorovat a považovat operátory ' a " za totožné.
Pole	TJ	Unární operace. Zobrazuje jeden nebo více řetězců. Objekty v poli mohou být řetězce a čísla. Pokud je objekt řetězec, pak je zobrazen, je li objekt číslo operátor pomocí daného čísla upravuje pozici. Opět můžeme se soustředit pouze na řetězce a čísla nebrat v úvahu. Příklad použití takovéto operace: `[ (Re) 100 (te) 120 (zec) ] TJ`

To je pouze přehled několika málo operací, které mohou být v textovém objektu. Nám ale tato množina stačí a pokud čitatel chce vědět celý seznam možných operací (a nejen v textovém objektu) může jej najít (pro verzi PDF 1.7) v ISO standardu PDF (viz Seznam použité literatury).

Zde je nutno zdůraznit, že řetězec, který se vykresluje nemusí být (a často také není) přímo text v nějakém fixním kódování, který je možno přímo poslat do výstupního souboru při extrakci textu. Bajty řetězce určují, jaké glyfy (grafické reprezentace písma) z Fontu se mají vykreslit. Dokonce ani nevíme, kolik bajtů řetězce patří jednomu výslednému znaku. Na to vše potřebujeme více informací o Fontech.

1.4.5. Fonty

O fontech pojednává ve Specifikaci mnoho kapitol, ale pro jednoduchost zkusím zde shrnout jen několik podstatných informací, které je třeba znát pro pochopení dalšího textu. PDF rozděluje fonty na dvě hlavní skupiny: jednoduché fonty (Simple fonts) a kompozitní fonty (Composite fonts). Každý použitý font v PDF je reprezentován fontovým slovníkem, který získáme ze zdrojů stránek. Fontový slovník poznáme podle položky `Type` ve slovníku, kde je vždy hodnota `Font`. Samotný font pro náš účel nepotřebujeme a stačí nám informace, které poskytuje jeho slovník. Font totiž popisuje hlavně zobrazování glyfů. Asi nejdůležitější položka ve fontovém slovníku je pro nás `ToUnicode`, která sice není v každém fontovém slovníku, ale pokud tam je, tak nám dává přímo speciální mapu pro získávání Unicode znaků (viz. 2.2.2.1 ToUnicode mapa). Tato mapa se může vyskytovat jak v jednoduchém, tak i v kompozitním fontu.

Kompozitními fonty se hlouběji zabývat nemusíme, protože pokud tuto mapu neobsahují tak další extrakci textu provádět nebudeme. Podrobněji je problém vysvětlen v kapitole 2.2.2 Převod kódů znaků na Unicode kódy. Kompozitní font poznáme ze slovníku tak, že má v položce `SubType` vždy hodnotu `Type0`.

K jednoduchým fontům si však musíme vyjasnit, co je rozdílové pole a kde je najdeme. Jednoduchý font poznáme podle toho, že slovníková položka `SubType` má hodnotu `Type1`, `TrueType` nebo `Type3`. Tento slovník může volitelně mít položku `Encoding`, která běžně odkazuje na jedno z následujících předdefinovaných kódování: StandardEncoding, MacRomanEncoding, MacExpertEncoding nebo WinAnsiEncoding. V některých případech, kdy je třeba trochu odlišné kódování, může položka Encoding obsahovat také kódovací slovník, který může mít položky popsané v následující tabulce.

Tabulka 1.10: Položky kódovacího slovníku

Klíč	**Hodnota**	**Popis**
Type	Název	(Nepovinné) Pokud položka existuje, pak má hodnotu **`Encoding`**.
BaseEncoding	Název	(Nepovinné) Název předdefinovaného kódování jehož rozdíly popisuje položka **`Differences`**. Tedy MacRomanEncoding, MacExpertEncoding nebo WinAnsiEncoding. Jestliže položka chybí, pak se použije základní vestavěné kódování fontu pokud je vložen v PDF souboru. Jinak se použije StandardEncoding.

Klíč	Hodnota	Popis
Differences	Pole	(Nepovinné) Pole popisující rozdíly oproti základnímu kódování v položce **`BaseEncoding`** nebo pokud položka chybí, pak rozdíly oproti implicitnímu základnímu kódování. Přesnější popis pole následuje pod tabulkou.

Ještě než vysvětlím, jak rozdílové pole použít, musím zdůraznit, že toto pole (stejně jako všechny předdefinované kódování) mapuje kódy znaků na jejich názvy. Nezískáme tedy nějakou Unicode hodnotu, nebo nějaký další kód, ale název znaku, který je třeba pak dále mapovat na Unicode hodnotu (viz. 2.2.2 Převod kódů znaků na Unicode kódy).

Rozdílové pole má tento formát:

$kód_1$ $název_{1.1}$ $název_{1.2}$ …
$kód_2$ $název_{2.1}$ $název_{2.2}$ …
…
$kód_n$ $název_{n.1}$ $název_{n.2}$ …

Každý kód má za sebou jeden nebo více názvů a tyto názvy mají od daného kódu postupně vždy o jedna vyšší kód. První název má tedy kód stejný jako uvozující kód, další název má kód o jedna větší atd. až do konce pole nebo do dalšího kódu, který pak určuje kódy pro názvy za ním. Příklad kódovacího slovníku:

```
25 0 obj
<<
/Type /Encoding
/Differences
[ 39 /quotesingle
128 /Adieresis /Aring /trademark
]
>>
endobj
```

V tomto slovníku podle rozdílového pole se kód 39 bude mapovat na jedoduchou uvozovku ('), 128 na Adieresis (Ä), 129 na Aring (Å) a 130 na trademark(™).

2. Extrakce textu

Nyní, když už máme hrubý základní přehled o PDF souboru (z kapitoly 1 PDF soubor) a jeho částech týkajících se zobrazování textu, můžeme se začít zabývat tím, jak z něj text (ideálně v nějakém Unicode kódování) získat.

2.1. Naivní přístup

Na začátku tohoto projektu, jako úplný nováček v tomto tématu, jsem brouzdal internetem a hledal užitečný kus kódu v C++ pro extrakci textu z PDF, který bych mohl použít jako inspiraci pro můj převaděč. Našel jsem jeden program neznámého autora, který ale varoval, že kód je velmi hrubý a neručí za něj. Avšak já na varování nedbal. Celý program čítal asi 250 řádků včetně komentářů a s přiloženým PDF souborem to vypadalo, že funguje. Zdál se mi jako dobrý začátek, proto jsem jej začal testovat, trochu upravovat a spolu s testováním se seznamovat s PDF formátem pomocí PDF Reference. Záhy jsem ale zjistil, že na jiných PDF souborech se do výsledného textového souboru nedostalo nic, nebo prázdné řádky. Dlouho jsem hledal soubor, na kterém by tento program vyprodukoval nějaký, alespoň trochu správný, výsledek.

Tento program fungoval přibližně takhle: Nahrál celý soubor do paměti. Procházel soubor postupně od začátku až do konce a vyhledával klíčová slova `stream` a `endstream`. Pokud našel, tak zkusil na data mezi klíčovými slovy použít zlib/deflate dekompresi. Pokud se „povedlo“, pak ve výsledku vyhledával klíčová slova `BT` a `ET` a pokud našel, tak v mezi nimi vyhledával vyvážené závorky '(' a ')' jejichž obsah vypisoval do výsledného textového souboru.

Každý teď, po přečtení předchozích kapitol, si nejspíš uvědomil, proč tento způsob nefungoval dobře. Pro ujasnění pár základních nedostatků vypíši. Program vyhledával všechny streamy, některé ale mohly už být nepoužité, nebo starší verze pokud byl soubor upravován. A bez načtení referenční tabulky to ani nemohl ze streamu nijak zjistit. Program mohl narazit na klíčové slovo `stream` i jinde než ve stream objektu – např. v komentáři, řetězci nebo shodou okolností v nějakých datech. Na všechny streamy pouštěl FlateDecode a ignoroval informace, které o transformaci streamu poskytuje jeho slovník. To pak způsobilo, že mohly být dekodovány nějaké binární data (např. Obrázek) a z nich pak extrahoval nesmyslné výsledky. Úplně ignoroval hexadecimální řetězce, které se pro reprezentaci textu velmi často

používají a v neposlední řadě, nenačetl strukturu dokumentu a nezjišťoval Fonty a kódování použité pro řetězce.

Sečteno a podtrženo, takový program fungoval správně pro cca 1% (ne-li méně) dostupných PDF souborů. Takže, když jsem si postupně tyto věci uvědomil, celou dosavadní práci jsem zahodil a začal programovat, jak se říká, na zelené louce - úplně od začátku a pořádně. Následující kapitola popisuje, jak má takový proces získávání textu vypadat a tuto kapitolu jsem napsal spíše pro to, aby si čitatel uvědomil jaké „nástrahy“ extrakce textu v sobě jímá a že tento úkol nelze vyřešit zcela jednoduše.

2.2. Celý postup extrakce

Unicode standard definuje systém číslování všech běžných znaků používaných ve velkém množství jazyků (viz. Unicode Standard - Seznam použité literatury). Je proto vhodným prostředkem pro reprezentaci jakéhokoli textu. Problém ale je, že řetězce používané v PDF nejsou řetězce v Unicode, ale jsou to kódy znaků, které určují glyf (grafickou reprezentaci znaku) pro vykreslení a proto převaděč do textu musí nějakým způsobem převést kódy znaků na Unicode kódy. Na tento postup se podíváme v následujících podkapitolách, nejdříve je ale třeba popsat, jak se dostat k řetězcům, abychom je mohli následně převádět.

2.2.1. Získání řetězců

V dřívějších kapitolách jsem popisoval strukturu dokumentu a z ní už postup získání řetězců, více méně, vyplývá, ale přece jen jej popíši, aby byl shrnut na jednom místě. Začnu od konce, od samého řetězce. Řetězce jsou, jak již víme, v textovém objektu (viz. 1.4.4 Textový objekt), a textové objekty jsou zase v obsahovém streamu (viz. 1.4.3.1 Obsahový Stream). Obsahové streamy najdeme ve stránce (viz. 1.4.2.2 Stránka), stránku ve stromu stránek (viz. 1.4.2 Strom Stránek), strom (nebo spíše jeho kořen) v katalogu dokumentu (viz. 1.4.1 Katalog dokumentu) a ke katalogu se dostaneme z patičky souboru (viz. 1.3.4 Patička (Trailer)). Struktura až k Obsahovému streamu je dobře vidět na znázornění Ilustrace 3. K nalezení katalogu dokumentu a všech dalších objektů ve struktuře je potřeba, aby aplikace načetla celou referenční tabulku podle popsaných pravidel v sekci 1.3.3 Referenční tabulka (Cross-reference Table). Z patičky je třeba z položky `Root` zjistit, který objekt je aktuální katalog dokumentu. Pak, s pomocí referenční tabulky, najít katalog

a v něm z položky **Pages** najít (zase s užitím referenční tabulky) kořen stromu stránek. Strom stránek pak je třeba projít pomocí položek **Kids** a **Parent** až ke stránkám, které poznáme podle toho, že položka **Type** ve slovníku má hodnotu **Page**. (Tento postup je také dobře vidět v nákresu Ilustrace 4) Stránka pak obsahuje položku **Contents**, která odkazuje už na popisovací Obsahový stream (a může jich být i více na jednu stránku). Ten je potřeba (v naprosté většině případů) dekomprimovat podle pravidel popsaných v stream objektu (viz. 1.2.2.7 Stream Object), jeho obsah postupně procházet, vyhledávat textové objekty uzavřené mezi klíčová slova BT a ET a v nich řetězce uzavřené do závorek '(' a ')' nebo '<' a '>'. Toto je postup jak se dostat k řetězcům, ale nyní následuje druhý nelehký úkol, a to převedení řetězce do Unicode.

2.2.2. Převod kódů znaků na Unicode kódy

Celé převádění je přímo závislé na Fontovém slovníku, který je nastaven jako aktuální ve chvíli, kdy se vykresluje řetězec. Proto, abychom věděli o jaký slovník se jedná, je třeba si pamatovat při procházení obsahového streamu, který název fontu byl naposledy operátorem Tf v Obsahovém streamu, nebo Textovém objektu, nastaven. Tento název pak je třeba vyhledat ve zdrojích stránky (viz. 1.4.3.2 Zdroje a 1.4.2.2 Stránka) ve slovníku **Fonts** a v něm najdeme Fontový slovník pro daný název. Nyní se pokusím popsat tři způsoby mapování kódů znaků na Unicode hodnoty v pořadí podle priority.

Jako první, co by měla aplikace udělat, pokud chce znát text řetězce, je zjistit, jestli v slovníku aktuálního fontu je položka **ToUnicode**. Pokud ano, tak obsahuje speciální objekt ToUnicode CMap a ten je třeba použít pro mapování kódů znaků na Unicode kódy. Je to speciální mapa, která slouží právě pro tento účel. Jak vypadá a jak se používá je popsáno v podkapitole 2.2.2.1 ToUnicode mapa. (Jak se dostat k ToUnicode mapě je také dobře znázorněno v nákresu Ilustrace 5)

Jestliže fontový slovník nemá ToUnicode mapu pro získání znaků, tak by aplikace měla zjistit, jestli se nejedná o jednoduchý font, který používá jedno z předdefinovaných kódování **MacRomanEncoding**, **MacExpertEncoding** nebo **WinAnsiEncoding**. Nebo jestli používá kódování, jehož rozdílové pole (viz. 1.4.5 Fonty) obsahuje pouze názvy znaků z Adobe Standard Latin množiny nebo množiny symbolů definovaných v PDF Standardu, Annex D, tabulka D.5 a D.6 (viz. Seznam použité literatury). Platí-li výše vyjmenovaná podmínka, pak Unicode

hodnotu získáme následujícím dvojitým mapováním:

1. Je třeba namapovat **kód znaku** na **název znaku** pomocí tabulky D.1 v Apendix D Adobe PDF Reference nebo identické tabulky D.2 v PDF standardu ISO 32000-1:2008 Annex D (viz. Seznam použité literatury) a rozdílového pole fontu (viz. 1.4.5 Fonty).
2. Následně je třeba získaný název znaku namapovat na Unicode hodnotu použitím seznamu Adobe Glyph List (viz. Seznam použité literatury). Je to seznam, který obsahuje názvy znaků a k nim příslušné Unicode hodnoty a je volně stažitelný ze stránek společnosti Adobe.

Třetí způsob, jak získat Unicode hodnoty, je nejsložitější a popíši jej pouze velmi zběžně, protože se stejně používá většinou pro znaky asijských jazyků, jako jsou Čínština, Japonština, Korejština aj. a ty nejsou cílem této práce. Proto ani nebudu příliš vysvětlovat dílčí pojmy, pouze tento způsob vypíši (resp. přeložím do češtiny z anglické specifikace PDF) pro informaci, aby byl postup kompletní.

Jestliže je font kompozitní font (viz. 1.4.5 Fonty), který používá předdefinovanou mapu CMap uvedenou v tabulce 118 (str. 273) PDF standardu ISO 32000-1:2008 (kromě Identity-H a Identity-V) nebo jeho pod-font CIDFont používá Adobe-GB1, Adobe-CNS1, Adobe-Japan či Adobe-Korea1 znakovou sadu, pak:

1. Namapujeme kód znaku na znakový identifikátor (CID) použitím CMap objektu ve fontu. (CMap je obecnější mapa, která mapuje znakové kódy na jakési identifikátory v CID fontu)
2. Získáme registr (`Registry`) a řazení (`Ordering`) znakové sady použité mapou CMap (např. Adobe a Japan1) z `CIDSystemInfo` slovníku ve fontu.
3. Sestavíme název druhé CMap spojením registru a řazení získaných v 2. bodě následujícím způsobem: *`registr-řazení`*`-UCS2` (např. `Adobe-Japan1-UCS2`)
4. Získáme CMap mapu podle názvu sestrojeného v bodě 3. (dostupné z ASN webové stránky www.itu.int)
5. Namapujeme získané CID z bodu 1. podle mapy získané z bodu 4. a tím získáme Unicode kód.

Pokud všechny tři způsoby pro získání Unicode hodnoty selhaly, pak zde již není možnost určit, jaký text znaky reprezentují. Jedině možná v některých případech

použitím OCR na vykreslený glyf, ale to už je téma hodné samostatné bakalářské práce.

2.2.2.1. ToUnicode mapa

Je to speciální objekt, který slouží pro mapování kódů znaků použitých v řetězci na Unicode kódy. Mapa pro Unicode znaky používá kódování UTF-16BE, takže je třeba to brát na vědomí a v případě potřeby převést na jiné kódování. K mapě se dostaneme přes fontový slovník a při extrakci textu musíme použít fontový slovník aktuálního fontu, protože v souboru může být těchto map více. Tato mapa vychází z obecnější definice objektu CMap, ale nebudeme potřebovat znát celou definici obecné mapy. CMap objekt je plně zdokumentován v Adobe Technical Note #5014 a částečně (ale dostatečně) v PDF standardu (viz. Seznam použité literatury), já zde popíši části nutné pro použití ToUnicode mapy.

Níže je příklad, jak taková mapa může vypadat a tučně jsem vyznačil části, které nás budou zajímat:

```
16 0 obj
<< /Length 433 >>
stream
/CIDInit /ProcSet findresource begin
12 dict begin
begincmap
/CIDSystemInfo
<< /Registry (Adobe)
/Ordering (UCS)
/Supplement 0
>> def
/CMapName /Adobe-Identity-UCS def
/CMapType 2 def
1 begincodespacerange
<0000><FFFF>
endcodespacerange
2 beginbfrange
<0000><005E><0020>
<005F><0061>[<00660066> <00660069> <00660066006C>]
endbfrange
1 beginbfchar
<3A51><D840DC3E>
endbfchar
endcmap
CMapName currentdict /CMap defineresource pop
end
end
endstream
endobj
```

První, co je v mapě důležité, jsou rozsahy kódů znaků. Znaky, jak víme, jsou často více-bajtové a v řetězci se mohou různě střídat znaky s různou velikostí. Proto potřebujeme nějakým způsobem pro načtení jednoho znaku zjistit, kolik bajtů z

řetězce vzít. K tomu slouží právě definice rozsahů znaků mezi klíčovými slovy `begincodespacerange` a `endcodespacerange`. Rozsahy jsou definovány pomocí dvojic řetězců, které určují počátek a konec rozsahu. Posloupnost bajtů považujeme za znak, pokud platí, že je v některém z rozsahů a má stejnou délku. Rozsah `<0000><FFFF>` z prvního příkladu říká, že znaky jsou vždy dvoubajtové s hodnotami od `0h` do `FFFFh`. V následujícím příkladu je vidět, jak může vypadat definice rozsahů, je-li jich více a s různou velikostí znaků.

```
4 begincodespacerange
< 00 >< 80 >
< 8140 >< 9FFC >
< A0 >< DF >
< E040 >< FCFC >
endcodespacerange
```

Rozsahy by se neměly překrývat a číslo, které je před `begincodespacerange` je spíše informační a určuje počet záznamů (v definici rozsahů). Když už nyní víme, jaké posloupnosti bajtů jsou znakové kódy, popíši nyní další sekce ToUnicode mapy, které určují mapování těchto kódů.

Objekt může obsahovat dva druhy sekcí pro mapování. První druh je sekce uzavřena mezi klíčová slova `beginbfchar` a `endbfchar`, která stejně, jako definice rozsahů, obsahuje dvojice řetězců a počet dvojic je před klíčovým slovem `beginbfchar`. První z dvojce je vždy kód znaku a druhý je jeho Unicode hodnota. Takže `bfchar` sekce z příkladu výše nám říká, že znak `3A51h` má Unicode hodnotu `D840DC3Eh` (UTF-16BE).

Druhý druh určuje mapování pro určitý rozsah kódů znaků. Jedná se o sekci uzavřenou mezi `beginbfrange` a `endbfrange` klíčová slova, uvozena opět počtem záznamů v sekci. Záznamy v této sekci jsou vždy trojice objektů a to buď tři řetězce, nebo dva řetězce a pole.

V prvním případě, kdy záznam je složen z tří řetězců se mapování provádí takto: První a druhý řetězec určuje rozsah kódů znaků, kterých se daný záznam týká a třetí řetězec je Unicode hodnota pro první znak v daném rozsahu (tedy první řetězec z trojice). Všechny další znaky z rozsahu se mapují inkrementálně k počáteční hodnotě rozsahu. Vyjasním na příkladu uvedeném výše. V sekci `begin/endbfrange` je následující záznam: `<0000><005E><0020>`. To znamená, že znak s kódem `0000h` má Unicode hodnotu `0020h`, a taky, že kód `0001h` má `0021h`, `0002h` má `0022h` atd. ... až `005Eh` má `007Eh`.

Ve druhém případě, kdy záznam je složen ze dvou řetězců a pole se mapování

provede následovně: První dva řetězce mají stejný význam jako v případě se třemi řetězci – tedy definují rozsah kódů, kterých se daný záznam týká a pole obsahuje pro každý kód z rozsahu jeden Unicode řetězec. Takže první kód z rozsahu se mapuje na první řetězec v poli a každý další kód se mapuje na další řetězec v poli. Pole musí mít přesně tolik řetězců, kolik je kódů v daném rozsahu. Tento případ se často používá pro mapování kódu na více Unicode znaků, proto, jak je vidět z příkladu, jsou Unicode řetězce delší. Ze záznamu s polem z příkladu výše tedy vyplývá následující mapování:

```
<005F> → <00660066>
<0060> → <00660069>
<0061> → <00660066006C>
```

Zbývá už jen dodat, že ToUnicode mapa je většinou nepřímý stream objekt a je třeba pamatovat na použití Filtrů na ni, jsou-li zadány ve slovníku na začátku objektu (viz. 1.2.2.7 Stream Object).

3. Program

Jak již bylo zmíněno v abstraktu, pro účel extrakce jsem také vytvořil program. Je napsán v jazyce C++ a ve vývojovém prostředí Microsoft Visual Studio 2008. Je odladěn v prostředí Microsoft Windows a krátce testován v Linux Ubuntu (verze 11.10). Pro testování v prostředí Mac jsem neměl prostředky, tudíž nemůžu funkčnost ani „kompilovatelnost“ zaručit. Při programování jsem však dbal na to, abych nepoužíval žádné windows-specific knihovny a tedy, aby projekt bylo možné zkompilovat i v jiných systémech. Podrobněji na téma knihoven píši v následující podkapitole 3.1 Nutné součásti. Na přiloženém DVD je také programátorská dokumentace, která může být užitečná v případě oprav nebo dalšího rozvoje programu.

3.1. Nutné součásti

3.1.1. Součásti pro kompilaci

Program potřebuje externí knihovnu zlib pro dekompresi streamů komprimovaných pomocí zlib/deflate. Proto mezi zdrojovými kódy najdete knihovnu `zdll.lib` pro Windows a knihovny `libz.a` a `libz.so` pro UNIX systémy v podsložce `lib` . Mezi zdrojovými kódy je pak ještě `zlib.h` hlavičkový soubor, nutný pro oba dva typy systémů.

Dále program pro převod kódování z `UTF-16BE`, které se používá v PDF, do `wchar_t` v C++, používá knihovnu iconv. Problém s touto knihovnou je, že verze pro Linux není kompatibilní s verzí pro Windows. Mají jiný hlavičkový soubor a také jiné předpisy funkcí a nedokázal jsem ani jednu verzi zprovoznit na obou typech systému najednou. Linux Ubuntu, na kterém jsem program testoval, obsahuje tuto knihovnu již ve standardních složkách, proto jsem ji nedával do mého projektu, a je tam pouze verze pro Windows. Kód pro pre-kompilátor pak vybere podle systému, jestli se má použít systémový `<iconv.h>` v Linuxu nebo projektový `"iconv.h"` ve Windows. Pokud uživatel v Linuxu tuto knihovnu nainstalovanou nemá, bude ji muset pro správnou kompilaci doinstalovat. Samotná kompilace v Linuxu se pak provede následujícím příkazem z místa se zdrojovými kódy:

```
g++ *.cpp -o pdfToText -L ./lib -lz
```

3.1.2. Součásti pro spuštění

Přiložené DVD (viz. A. Obsah přiloženého DVD) obsahuje spustitelný program jak pro Windows (pdfToText.exe), tak pro Linux (pdfToText) ve složce `Release`. Pro běh ve Windows musí být spolu s programem (a taky jsou na přiloženém DVD) ve stejné složce runtime knihovny `zlib1.dll` a `libiconv2.dll` z důvodů popsaných výše v kapitole 3.1.1Součásti pro kompilaci. Jak se program spouští a používá včetně jeho grafické nástavby pro Windows je popsáno v příloze B. Uživatelská dokumentace. Pro Linuxovou verzi není třeba mít knihovny ve stejné složce, protože jsou zakompilovány do programu.

Do výsledného souboru programu se vypisuje přímo řetězec typu wchar_t. Tento typ je platform specific – tedy může mít různou implementaci a kódování na různých systémech. Běžně ve Windows prostředí je použito pro `wchar_t` kódování `UCS-2` (což je starší verze `UTF-16`) a v Linuxových prostředích `UTF-32`. Je tedy potřeba mít v systému pro otevření výsledných souborů textový editor, který daná kódování podporuje. Není to ale žádný problém, protože všechny běžné systémy takové editory mají. Ve Windows se správně soubor otevírá v poznámkovém bloku. V Linuxu jsem jej otevíral pomocí programu gedit, který se mně při prvním otevření ptal na kódování. Po vybrání `UTF-32` se výsledek správně zobrazil.

3.2. Co program umí

Dá se říci, že to, co jsem probíral v předchozích kapitolách (viz. 1.2 Syntaxe, 1.3 Struktura souboru, 1.4 Struktura Dokumentu, 2.2 Celý postup extrakce), program provádí, kromě věcí uvedených v kapitole 3.4 Co programu chybí. Umí přeskakovat komentáře za běhu. Nenačítá celý soubor aby jej jedním průchodem vyčistil z komentářů. A ani by to moc dobře nešlo kvůli binárním sekcím v PDF souboru.

3.2.1. Podporované soubory

Nedá se přesně říci, jakou verzi PDF souboru program podporuje, protože při programování jsem se neřídil rozdíly mezi verzemi, ale spíše jsem se snažil pokrýt funkcionalitou většinu běžných souborů. PDF je formát, ve kterém verze souboru nerozhoduje o tom jak složitá bude extrakce textu. Proto program bez problémů zpracovává PDF soubory vytvořené pomocí OpenOffice (LibreOffice v Linuxu) nebo Microsoft Office kancelářských balíků, ale má problém s některými (i ve verzi staršími) ostatními PDF soubory.

3.2.2. Logování

Program navíc, kromě věcí které „musí umět“, ještě má dost podrobné logování. Logy jsou dvojího typu – chybové a informační. Chybové logy program vypisuje do souboru `errorfile.txt` a informační do `logfile.txt` v místě jeho spuštění. Přípravu na logování jsem provedl hned ze začátku psaní programu a v průběhu programování jsem neustále přidával nové chybové a informační logy. Proto se jen zcela výjimečně stává, že by program výstup nevygeneroval a v `errorfile.txt` nebyl uveden důvod proč. Informační `logfile.txt` slouží pak hlavně pro programátora, aby mohl z něj zjišťovat průběh extrakce a případně zjistit ve které části se zastavil.

3.3. Jak program funguje

Pro správné pochopení této části práce je třeba, aby čitatel již byl seznámen a chápal předchozí kapitoly 1 PDF soubor a 2 Extrakce textu. Pokusím se nyní popsat, co vlastně program provádí a jak.

3.3.1. Příprava programu

Na samém začátku po spuštění se provádí příprava. Nejdříve nastaví počáteční časy pro měření doby běhu a otevřou se oba logovací soubory v režimu pro přidávání (append). Pak se kontrolují argumenty, otevírá vstupní i výstupní soubor, a inicializuje iconv. Vše je samozřejmě doprovázeno příslušnými kontrolami.

(Na konci programu se spočítá čas běhu, a pak se zase všechny soubory zavírají.)

3.3.2. Získání tabulky nepřímých objektů

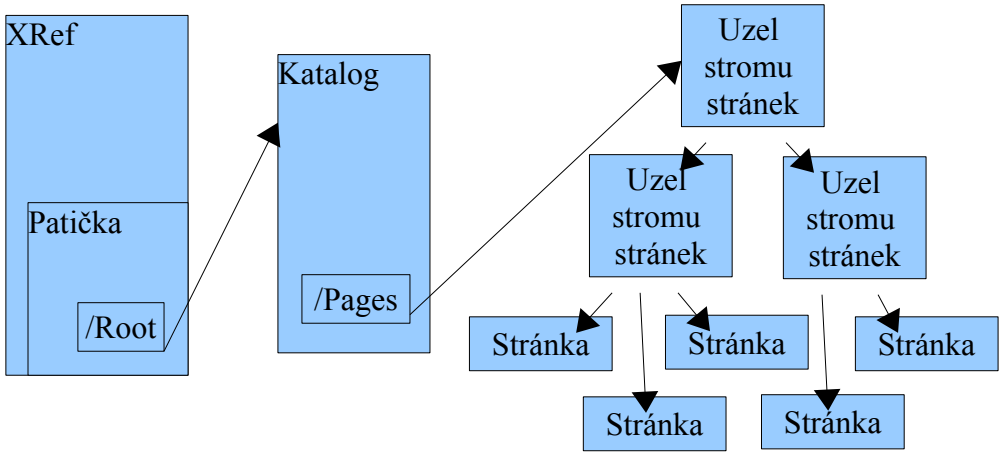
Pro načtení referenční tabulky jsem vytvořil speciální objekt `XRef`. Jedná se o reprezentaci jedné sekce referenční tabulky. Tento objekt načte patičku souboru, z ní zjistí pozici první sekce referenční tabulky a tu následně celou rozparsuje a uloží do paměti. Objekt poskytuje také onu důležitou funkcionalitu pro načítání dalších sekcí referenční tabulky. Zjistí si z jemu příslušné patičky, jestli a kde se nachází další referenční sekce a je-li na něm zavolána funkce `getNextXRef()`, vytvoří a vrátí další `XRef` objekt, který obsahuje tuto sekci. Takto se dají rekurzivně získat všechny sekce tabulky z objektu reprezentující první sekci.

Vyhledávání v referenční tabulce je jedna z nejdůležitějších věcí při práci s PDF, proto se následně prochází všechny sekce a vytváří jednotlivé objekty, které

jsou pak uloženy do mapy objektů. Vzhledem k tomu, že STL kontainery z C++ jsou dobře optimalizované na takové věci, použil jsem přímo třídu `map<>` pro tento účel. Nepřímé objekty v mapě jsou indexovány dvojicí čísel, jak je popsáno v předešlých kapitolách.

Tato mapa je pak výborným prostředkem pro rychlý přístup k objektům a jelikož se používá téměř v každé třídě, byla jako jedna z mála proměnných zvolena jako globální.

3.3.3. Načtení struktury

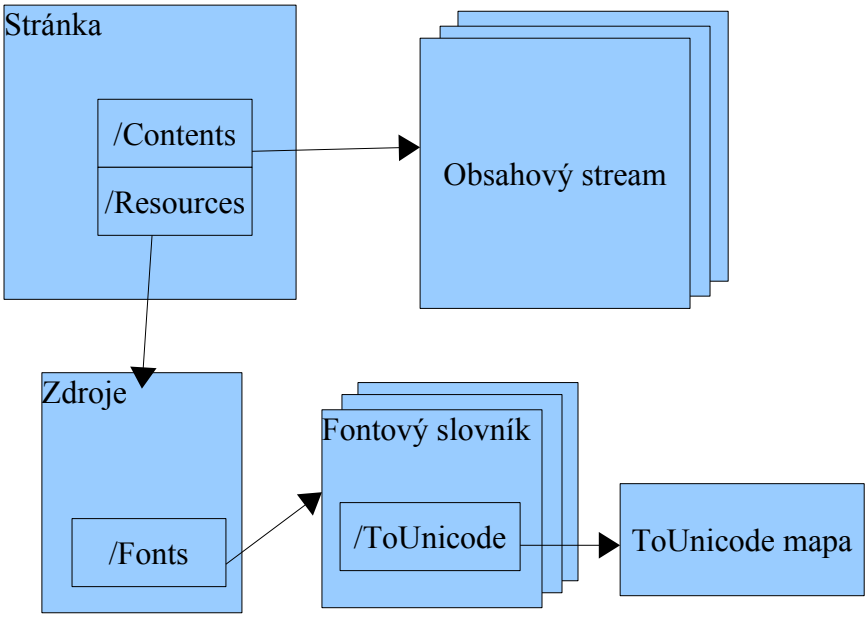


Ilustrace 4: Nákres načítání struktury

Postup načítání struktury dokumentu jsem se snažil znázornit graficky výše (viz. Ilustrace 4), abych jej nemusel složitě slovně vysvětlovat. Ze slovníku v patičce první referenční sekce program získává odkaz na katalog dokumentu. V katalogu je pak odkaz na kořenový uzel stromu a ten se pak se všemi potomky postupně celý načítá do stromové struktury. Program si následně z oné struktury vygeneruje ve správném pořadí pouze obyčejný seznam stránek (tedy „posbírá“ všechny listy). Pro doplnění – uzly na sebe navzájem ukazují pomocí položek `/Kids` a `/Parent`. Pro reprezentaci patičky a katalogu používám v programu třídu `DictionaryObject` a pro uzly i stránky třídu `PageTreeNode`.

3.3.4. Získání textu ze stránek

Po získání seznamu stránek se pak každá stránka zpracovává samostatně. Program prochází stránky a volá na nich metodu `getText()`, které do parametru dá odkaz na otevřený výstupní soubor, a o zbytek se už stará stránka a objekty pod ní. V kapitole 2.2 Celý postup extrakce jsem se snažil podrobně popsat extrakci řetězců a převod na Unicode a nechci se opakovat, proto se tady zaměřím na trochu vedlejší



Ilustrace 5: Propojení důležitých objektů ze stránky

věci, které jsou přímo spojeny s programem. Ilustrace 5 znázorňuje jak jsou jednotlivé objekty propojeny. Role stránky v programu je, že stránka zavolá na všech svých obsahových streamech (třída ContentStream) metodu getText(), která vrací wchar_t text a ten uloží do souboru. Celý proces získání textu je tak přenechán obsahovému streamu, což z něj činí jeden z nejdůležitějších objektů programu. Obsahový stream je Stream Object, takže je třeba jej nejdříve dekomprimovat je-li komprimovaný. Na to je ve třídě `IndirectObject` metoda `processAsStream()`, která použitím zlib knihovny toto zprostředkuje. Obsahový stream je stream objekt a každý stream objekt je nepřímý (Indirect) objekt, proto je tato metoda v `IndirectObject` a ne přímo v `ContentStream`. Ten se jen pomocí odkazu na `IndirectObject` instanci vytvoří.

Při samotném procházení obsahového streamu program musí přes stránku a její zdroje získávat informace z fontových slovníků. Pokud stránka odkaz na slovník se zdroji nemá, je třeba jej hledat ve vyšších větvích stromu stránek, protože zdroje se ve stromu dědí. Dědičnost je ale velmi zjednodušena. Zdrojové slovníky se nijak nespojují. Pokud není slovník definován na nějaké stránce, postupuje se stromem směrem nahoru, až program na nějaký zdrojový slovník narazí a ten pak je třeba použít. Zbývá jen dodat, že výsledné Unicode řetězce, které jsou postupem popsaným v kapitole 2.2 Celý postup extrakce získány, program převádí z UTF-16BE do wchar_t pomocí iconv knihovny a řetězce, které kvůli chybějícím částem (viz. 3.4 Co programu chybí) program prozatím neumí zpracovat, jednoduše převede na `wchar_t` jako by to byl obyčejný `char`, pomocí `msctowcs()` funkce. Což

sice je špatně, ale lepší než nic, protože se ve většině případů zachovají aspoň ASCII znaky.

3.4. Co programu chybí

V této sekci popíši zhruba věci, které programu chybí, aby správně zpracoval všechny běžné PDF soubory. Sice se může zdát, že chybějících částí je mnoho, ale fakt je, že důležitých je jen pár prvních, protože ostatní případy se velmi zřídka používají a postihují tak odhadem cca 1-2% dostupných PDF souborů. Proto vypíšu chybějící části v pořadí podle četnosti použití v PDF souborech.

- Jako první a asi nejdůležitější nedostatek je, že v programu chybí zpracování funkcí zpětného lomítka definovaných v kapitole 1.2.2.3.1 Písemný řetězec. Doprogramování této části by nemělo být nijak složité pro případné zájemce.
- Druhým nejdůležitějším, ale také relativně jednoduchým nedostatkem je druhý způsob převodu kódů znaků na Unicode hodnoty, který je popsán v kapitole 2.2.2 Převod kódů znaků na Unicode kódy. Jedná se o dvojité mapování kódů znaků, pokud je použit jednoduchý font. Bylo by třeba pro tento účel přepsat tabulky definované v PDF standardu do kódu a také rozparsovat a použít Adobe glyph list (viz. Seznam použité literatury).
- Program umí pro dekódování Stream Objektu požít jen filtr FlateDecode (viz. 1.2.2.7.1 Filtry). Ten je v externí knihovně zlib. V ne příliš častých případech se pro kódování (kompresi) Content Streamu, ToUnicode mapy a jiných objektů použitých při extrakci textu, používá LZW kompresní metoda a je třeba použít filtr LZWDecode a ten již v programu není a bylo by jej třeba doprogramovat (nebo získat již hotovou knihovnu). Teoreticky by se mohly používat i ostatní filtry (Tabulka 1.4: Seznam filtrů Stream objektu), ale prozatím jsem nenarazil na takový případ. Program v chybovém logu `errorfile.txt` informuje o tom, pokud narazí na filtr, který není implementován.
- Od verze 1.5 PDF souboru mohou být i nepřímé objekty uloženy ve streamu (nazývá se Objects stream – neplést se Stream Object) a stejně tak i referenční tabulka (Cross-Reference Stream), která pak na tyto (a třeba i na ostatní) objekty odkazuje. V průběhu testování jsem narazil pouze na jeden soubor, který tento způsob referenční tabulky a uložení nepřímých objektů obsahoval, avšak pokud má být program kompletní, je třeba i toto brát v

úvahu. Myšlenka Objects streamu a Cross-reference streamu je dobrá a myslím, že se bude v budoucnu podporovat a rozšiřovat mezi PDF soubory. Program opět vypíše do chybového logu informaci, je-li v aktuálním PDF použita tabulka v Cross reference streamu.

- Jedná se o třetí postup pro mapování znakových kódů na Unicode hodnoty. Týká se případů, kdy je použit kompozitní font a nemá ToUnicode mapu. Podrobněji je to popsáno a taky vysvětleno, proč není tento postup příliš důležitý, v kapitole 2.2.2 Převod kódů znaků na Unicode kódy.
- Program nepodporuje šifrované PDF soubory. Ty lze poznat tak, že po otevření PDF v textovém editoru není vidět text, ale nějaká binární data.
- V některých výjimečných případech se můžeme setkat s PDF soubory které jsou v podstatě více souborů (PDF dokumentů) v jednom. Dobrým příkladem je samotná PDF Reference, na kterou se tak často v této práci odkazuji. Takové soubory nejsou podporovány a jsou programem pouze částečně zpracovány.
- V kapitole 1.2.2.7 Stream Object popisuji, že samotný stream se může nacházet i v externím souboru a ne přímo v PDF souboru. Takový případ taky není programem očekáván a myslím, že se ani pro Content Streamy, ToUnicode mapy a další potřebné objekty nikdy nepoužívá. Je tu pouze teoretická možnost, a v praxi nepodstatná, proto jsem tento „nedostatek“ nechal na konec.

3.5. Efektivita a jiné vlastnosti

3.5.1. Rychlost

Rychlost zpracování souboru se velmi liší podle konkrétního souboru a použitého počítače. Asi nemá smysl sem psát konkrétní časy, ale pouze orientačně konverze trvá řádově desetiny sekundy a pro větší soubory se mohou časy pohybovat v řádech několika sekund.

Program je jedno-vláknový a nevidím příliš možnost jak jej rozdělit na vice vláken. Jediné co mně napadá, je zpracovávat více stránek najednou v několika separátních vláknech, protože zpracování stránek je oddělitelné. Na druhou stranu nejsem si jist, jestli by režie nebyla větší než přínos, který by to mělo. Prostor pro optimalizaci kódu (když pominu samotnou paralelizaci) zde určitě je, avšak

nemyslím si, že by se dal zefektivnit nějak razantně (o víc než 20-40%).

3.5.2. Paměť

Paměťová náročnost je možná slabou stránkou tohoto programu, protože jsem počítal s tím, že v dnešní době je paměti v počítačích dost, a neustále přibývá. Takže je zde dost veliký prostor pro paměťovou optimalizaci. Orientačně jen řeknu, že pro větší PDF o velikosti 5MB spotřebuje za běhu cca. pětinásobek paměti – tedy kolem 25MB. Ovšem opět velmi záleží na konkrétním souboru. Pokud je v PDF mnoho obrázků a jiných netextových objektů, tak ty se nedekomprimují ani nijak nezpracovávají tudíž příliš paměť nezatěžují. Dalo by se dokonce jednoduše program upravit tak, aby se ani nenačítaly ze souboru. Byl-li PDF upravován a jsou v něm již nepoužívané objekty, pak tyto jsou ignorovány úplně a ani se do paměti ze souboru nenačítají.

Program se také, z rychlostních důvodů, nezabývá okamžitě mazáním použitých objektů, až na některé výjimky. Proto za běhu programu paměť jen roste a až po skončení se celá uvolní. Neviděl jsem to jako prioritu a bylo by to na úkor rychlosti. Dal jsem si však pozor, abych objekty, které se používají vícekrát, nevytvářel pořád, ale cachoval.

3.5.3. Návratová hodnota

Samozřejmě, po úspěšném ukončení programu po celém procesu extrakce, je návratová hodnota 0 a obecně nastane-li vážnější problém, pak je větší než 0. Podle toho také grafická nástavba pro Windows rozpoznává, zda se konverze do textu povedla a má otevřít výsledný soubor, nebo ne. Pro případ, že by to někdo potřeboval podrobněji, popíši jaké kladné hodnoty mohou být vráceny. Byl-li program spuštěn se špatnými parametry (špatný počet), návratová hodnota je 1. Nepovedlo-li se otevřít vstupní nebo výstupní soubor, pak je vrácena hodnota 2. Program vrátí 4 pokud se nepovede načíst správně referenční tabulku a mapa nepřímých objektů je tak prázdná (není pak důvod pokračovat). V případě, že nastane chyba při načítání struktury dokumentu, která brání v pokračování programu, je návratová hodnota 3. A nepovede li se inicializovat iconv, je v takovém případě vrácena hodnota 5. Ve všech těchto případech se podrobnosti dají vyčíst z chybového logu. Program nemá žádnou speciální návratovou hodnotu pro chybu v samotné extrakci textu, protože v takovém případě se hned neukončuje, ale pokouší se pokračovat dalším řetězcem, streamem či stránkou.

3.5.4. Ostatní vlastnosti

V kódu jsou také zohledněny newline znaky při exportu používané v jednotlivých systémech, ve kterých je kompilován. Podle systému se jako newline příslušný Windows použije kombinace CR+LF, pro Linux CR a pro Mac LF.

A jako poslední detail ještě uvedu, že program měří čas běhu a vypisuje jej do obou logů na konec, takže se dá tento údaj použít při optimalizaci kódu.

Doslov

Jak již jsem psal v předmluvě na samém začátku, snažil jsem se touto prací dosáhnout jakéhosi návodu pro získání textu z PDF souborů a poskytnout program, který sice pro použití v praxi (např. někde na serveru) vyžaduje ještě další úpravy, ale poskytuje dobrý výchozí bod pro práci a může rovněž mít informativní účel. Snažil jsem se vysvětlit, vybrat a přeložit z PDF specifikace ty části, které jsou podstatné, aby se člověk nemusel seznamovat s celou specifikací a zároveň, aby měl dostatečné informace pro splnění úkolu, jakým je extrakce textu. Trávil jsem dlouhé hodiny studováním PDF specifikace a hledáním odpovědí na otázky, které mi v průběhu práce vyvstávaly a doufám, že se mi touto prací podaří někomu tyto hodiny ušetřit.

Seznam použité literatury

1. Network Working Group, Internet RFC 1950, ZLIB Compressed Data Format Specification
2. Network Working Group, Internet RFC 1951, DEFLATE Compressed Data Format Specification
3. Adobe Systems Inc., PDF Reference sixth edition and Related Documentation
4. Adobe Systems Inc., ISO standard PDF32000-1:2008 Portable document format - Part 1: PDF 1.7
5. The Unicode Consortium. The Unicode Standard, Version 5.2.0, defined by: The Unicode Standard, Version 5.2 (Mountain View, CA: The Unicode Consortium, 2009. ISBN 978-1-936213-00-9). (http://www.unicode.org/versions/Unicode5.2.0/)
6. Adobe Systems Inc., Adobe Glyph List, Table version: 2.0
7. Adobe Systems Inc., Adobe Technical Note #5014, Adobe CMap and CIDFont Files Specification

Seznam tabulek

Seznam použitých zkratek

PDF:	Portable Document Format
PDL:	Page Description Language
DDL:	Data Description Language
OCR:	Optical Character Recognition
ISO:	International Organization for Standardization
EOL:	End Of Line
EOF:	End Of File
LF:	Line Feed
CR:	Carriage Return
ASCII:	American Standard Code for Information Interchange
UTF-8:	UCS Transformation Format 8bit
UCS:	Universal Character Set
LZW:	Lampel-Ziv-Welch
CCITT:	Comitè Consultatif Internationale de Tèlègraphie et Tèlèphonie
CID:	Character identifier

Přílohy

A) Obsah přiloženého DVD

- Tato bakalářská práce v elektronické formě.
- Program `pdfToText` se zdrojovými kódy a spustitelným souborem ve složce `pdfToText`. Spustitelné soubory jsou v podsložce `Release`.
- Dokumentace programu v html ve složce `Documentation`
- Program `pdfToTextGui` se zdrojovými kódy a spustitelným souborem `pdfToTextGui.exe` ve složce `pdfToTextGui`. Spustitelné soubory opět v podsložce `Release`.
- Všechny položky použité literatury (viz. Seznam použité literatury) ve složce `Specification and related documents`. (kromě 7. Adobe Technical Note #5014, a 5. The Unicode Standard)

B) Uživatelská dokumentace

Jedná se v podstatě o dva programy a oba mají velmi jednoduché použití. (Poznámka: Před spuštěním je ale třeba je nakopírovat na nějaké místo, kde mají právo zápisu. Oba programy totiž generují logy do textových souborů v místě spouštění. Proto není možné je spustit např. z CD či DVD, protože tam nelze soubory vytvořit ani upravit.) První program `pdfToText.exe` je command line aplikace (- aplikace pro příkazovou řádku) a používá se následovně:

```
pdfToText.exe [vstupní PDF soubor] [výstupní textový soubor]
```

Tedy název programu, za ním cesta k PDF souboru, který chceme převést a následuje název (nebo cesta) souboru, který se má vytvořit nebo přepsat. (Poznámka: Je potřeba, aby ve stejné složce, kde se nachází program byl i soubor zlib1.dll a libiconv2.dll)

Druhý program `pdfToTextGui.exe` je pro uživatelé operačního systému Microsoft Windows pouze pomůcka, která spouští command line převaděč `pdfToText.exe`. Po spuštění vypadá jako na obrázku níže a obsluha je podle mě již intuitivní. Uživatel vybere po stisknutí tlačítka „Select...“ PDF soubor por převod nebo sám vyplní cestu do textového pole vedle tlačítka, v případě potřeby změní název výstupního souboru a stiskne Run. Pokud se převod povede, program automaticky otevře výsledek v prohlížeči textových souborů (většinou se jedná o notepad – Poznámkový blok) a zavře se. Pokud se převod nepovede vypíše se v dolní části informace, že převod nebyl úspěšný a uživatel může zkontrolovat zadané cesty nebo zkusit převést jiný PDF soubor.