Zpravodaj Československého sdružení uživatelů TeXu

Zdeněk Wagner

Z LaTeXu přes PostScript do PDF

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 9 (1999), No. 1-2, 78–105

Persistent URL: http://dml.cz/dmlcz/149838

Z LaTeXu přes PostScript do PDF

Zdeněk Wagner

V tomto článku je popsán jeden z možných postupů tvorby PDF z LaTeXu. Autora k vytvoření této metody přinutila nouze a časová tíseň, kdy mezi začátkem studia a nejzazším termínem odevzdání PDF souborů byl jediný večer a následující noc. Na počátku se jednalo pouze o pár maker v LaTeXu a PostScriptu. Postupným vývojem však vznikl zajímavý nástroj, který je již skoro rutinně využíván pro převod starších čísel tohoto Zpravodaje (počínaje ročníkem 1996) do PDF. V textu je dále ukázáno, jak se podařilo vyvrátit mýtus o nemožnosti řádkového zlomu uvnitř hypertextového odkazu. Přitom je vše řešeno výhradně na úrovni maker, není nutno psát žádné programy ve vyšších programovacích jazycích.

1. Úvod

S rozvojem výpočetní techniky poněkud ustupuje význam tištěných informací a do popředí vystupují elektronické publikace. Již dávno se autoři nespokojují s obyčejnými textovými soubory a grafický vzhled je velmi důležitý.

Elektronické dokumenty jsou šířeny na kompaktních discích a vystavovány na Internetu. Dostávají se tak širokému okruhu čtenářů, kteří na svých počítačích používají různé operační systémy. Pokud je v zájmu autora, aby všichni čtenáři viděli na svých obrazovkách (nebo po vytištění na papíře) dokument ve stejném tvaru, musí zvolit standardizovaný formát, který není závislý na platformě. Jedním z nich je Portable Document Format – PDF.

Tvůrcem formátu PDF je firma Adobe. Formát vychází z PostScriptu, z něhož jsou odstraněny zejména programovací konstrukce a jsou přidány příkazy podporující hypertextové odkazy jak uvnitř dokumentu, tak ke vnějším zdrojům. Je tím zajištěno rychlé zobrazování i flexibilita vazeb na lokální i vzdálené objekty dostupné na Internetu.

Každý dokument lze nějakým způsobem převést na PDF. V dalším textu se ovšem zaměříme pouze na dokumenty, které jsou psány v LaTeXu. Pro jejich převod do PDF máme v principu dvě možnosti:

1. Program pdfTeX, který místo DVI vytváří přímo PDF.
2. Vytvoření PostScriptového souboru, který pak převedeme do PDF buď komerčním programem Distiller nebo jiným ekvivalentním nástrojem (např. pstill).

Bohužel pdfTeX nebyl dosud kompilován jako nativní aplikace pro OS/2. Je dostupná implementace využívající extender DJGPP, ale DOSové programy

nemohou v OS/2 používat soubory s „dlouhými jmény“. Pokud již někdo má v OS/2 rozsáhlý TEXový strom obsahující stovky (nebo i tisíce) souborů s dlouhými jmény, pak je pro něj takto „omezený“ program nepoužitelný. Překlad pdfTEXu ovšem vyžaduje řadu unixovských nástrojů a ukázalo se, že to není jednoduchá záležitost. Schůdná tedy byla pouze druhá cesta.

Užitečným pomocným nástrojem by mohl být balík HYPERREF. Zpravodaj ovšem řeší řadu maker netradičním způsobem, a byto by naivní předpokládat, že si HYPERREF poradí s dokumentem, který se dost odlišuje od LATEXového standardu. Bylo by nutno zasáhnout do některých maker a vzhledem ke značnému rozsahu balíku HYPERREF jsem nabyl dojmu, že by to vyžadovalo delší studium. Protože specifikaci formátu PDF jsem si již dříve pročítal, rozhodl jsem se, že vytvořím vlastní jednoduchá makra, která vyřeší alespoň základní problémy. Hlavním cílem bylo vytvoření takových maker, která by neovlivnila zlom již vysázeného dokumentu a byla použitelná pro co nejširší třídu dokumentů s co nejmenšími zásahy.

2. Základní makra

Distiller je vlastně PostScriptový interpret, jehož výstupem je PDF. Můžeme jej přirovnat k TEXovému expand procesoru. Distiller interpretuje PostScriptové programové konstrukce a převádí je na objekty a operátory přípustné v PDF. Abychom mohli přímo vytvářet hypertextové odkazy, je jazyk, který je implementován v Distilleru, obohacen o několik PostScriptových operátorů. Pokud bychom PostScriptový soubor, určený pro Distiller, chtěli vytisknout, skončilo by to chybou kvůli nedefinovanému operátoru. Tomu předejdeme tím, že do prologu PostScriptového souboru vložíme následující kód:

```
/pdfmark where
{pop}{userdict /pdfmark /cleartomark load put} ifelse
/setdistillerparams where
{pop}{userdict /setdistillerparams {pop} put} ifelse
/currentdistillerparams where
{pop}{userdict /currentdistillerparams {1 dict} put
} ifelse
```

Pochopitelně nebudeme tyto složité příkazy opisovat do každého dokumentu. Vložíme je společně s dalšími makry, která uvedeme později, do souboru s názvem `pdfhdr.ps` a ve svém dokumentu načteme obvyklým příkazem:
`\usepackage{pdfhdr}`
styl, který obsahuje povel:
`\special{header=pdfhdr.ps}`
Tím budou makra zavedena automaticky do každého dokumentu.

Práci Distilleru lze do jisté míry ovlivňovat. Parametry lze nastavit přímo v konfiguraci programu, ale můžeme je také zapsat do vstupního PostScriptového souboru. To je vhodné zejména v případě, kdy nemáme přímou kontrolu nad konfigurací programu, například při používání síťové verze Distilleru. Pro Zpravodaj se používá následující nastavení:

```
<< /CompatibilityLevel 1.2
   /CompressPages true
   /AutoFilterGrayImages false
   /EncodeGrayImages true
   /GrayImageFilter /FlateEncode
   /DownSampleGrayImages false
   /EncodeMonoImages true
   /MonoImageFilter /FlateEncode
   /DownSampleMonoImages false
   /AutoFilterColorImages false
   /ColorConversionStrategy /LeaveColorUnchanged
   /EncodeColorImages true
   /ColorImageFilter /DCTEncode
   /ConvertColorImagesToRGB true
   /DownSampleColorImages false
   /EmbedAllFonts true
   /SubsetFonts true
   /MaxSubsetPct 99
>> setdistillerparams
```

Je dobrým zvykem ukládat PostScriptové definice do vlastního slovníku. Jak se později ukáže, lze toho využít i k užitečnému triku. Tento slovník si musíme vytvořit a vyhradíme mu prostor pro 80 definic. Slovník otevřeme a všechna další makra se tak budou ukládat do něj.

```
/zwPDFdict 80 dict def
zwPDFdict begin
```

Na konci souboru musí být pochopitelně slovník uzavřen operátorem `end`.

3. Vytváříme jednoduché odkazy

Tuto část zahájíme citací. Hàn Thế Thành ve svém článku [1] napsal:

Pokud jde o implementaci „neTEXovských“ prvků (hypertextových odkazů, záložek), museli bychom vložit z TEXu do DVI *souboru speciální sekvence, které označují pozice hypertextových odkazů a další potřebné informace. Potom program* ***dvi2pdf*** *na základě těchto sekvencí bude generovat příslušné objekty do* PDF. *Tato cesta nám neumožňuje využít řadu důležitých informací o dokumentu, které TEX poskytuje, např. o strukturách boxů v dokumentu. Kdybychom např.*

chtěli mít víceřádkový hypertextový odkaz, který obsahuje několik vět v odstavci, a chceme, aby výskyt hypertextového odkazu neovlivnil sazbu odstavce, museli bychom nejdříve sázet tento odstavec bez vložení hypertextového odkazu. Potom prohlížíme sazbu a ručně přidáme na příslušná místa v textu sekvence, které nám označují pozice, kde chceme mít hypertextový odkaz. Protože ve chvíli, kdy píšeme texty v TEXu, neznáme řádkové zlomy, nemůžeme určit, kam máme tyto sekvence napsat, abychom dostali požadovaný výsledek bez prohlížení dokumentu po sazbě. Při použití tohoto postupu máme dvě možnosti. Buď budeme muset sázet hypertextové odkazy do boxu a tím zakazujeme řádkový zlom uvnitř hypertextového odkazu, nebo budeme muset hypertextové odkazy ručně přidávat na příslušná místa, pokud chceme mít řádkový zlom uvnitř hypertextového odkazu.

Aktivní oblastí pro hypertextový odkaz totiž musí být obdélník a jeho souřadnice musíme do PDF-souboru zapsat. TEX nám poskytne jen omezenou informaci. Pokud vysázíme část textu do boxu, můžeme zjistit jeho rozměry a zapsat je vhodným způsobem do DVI-souboru. PostScriptový operátor `currentpoint` nám pak poskytne počátek textu, takže příkaz pro hypertextový odkaz můžeme při spolupráci TEXových a PostScriptových maker vytvořit.

Začneme tím, že si napíšeme pomocná makra pro zápis povelů `\special` do DVI-souboru. Budeme potřebovat dvě makra. První makro zapíše PostScriptový povel pro otevření našeho slovníku. Druhé makro budeme používat v případech, kdy je již slovník otevřen.

```
\def\pdf@write#1{\special{ps:zwPDFdict begin #1}}
\def\pdf@@write#1{\special{ps:#1}}
```

Aktivní text budeme sázet do boxu a budeme jej měřit. Potřebujeme tedy alokovat box a rozměrový registr a vytvoříme si jednoduché makro, které odřízne desetinnou část a jednotky.

```
\newbox\pdf@hbox  \newdimen\pdf@tempdim
\def\pdf@#1.#2\pdf@{#1}
```

Aktivní oblast vytvoříme trochu větší než vlastní box. Použijeme k tomu následující makro:

```
\def\pdf@dimensions{%
  \pdf@tempdim\wd\pdf@hbox \advance\pdf@tempdim .9pt
  \edef\pdf@w{\expandafter\pdf@\the\pdf@tempdim\pdf@}%
  \pdf@tempdim\ht\pdf@hbox \advance\pdf@tempdim .9pt
  \edef\pdf@h{\expandafter\pdf@\the\pdf@tempdim\pdf@}%
  \pdf@tempdim\dp\pdf@hbox \advance\pdf@tempdim 1pt
  \edef\pdf@d{\expandafter\pdf@\the\pdf@tempdim\pdf@}}
```

Hypertextový odkaz pak budeme vytvářet makrem `\pdfannot`, které má tři parametry. Prvním parametrem určujeme typ odkazu. Povolené hodnoty jsou:

PDFlink pro odkaz uvnitř dokumentu,
PDFurl pro odkaz na objekt na Internetu,
PDFfile pro odkaz na soubor na lokálním disku.

Makro ovšem hodnotu parametru nijak netestuje. Použití nepovoleného parametru způsobí až při dalším zpracování PostScriptovou chybu. Druhým parametrem makra `\pdfannot` je cíl odkazu, který popíšeme později. Třetím makrem je aktivní text.

```
\def\pdfannot#1#2#3{%
  \setbox\pdf@hbox=\hbox{#3}\pdf@dimensions
  \box\pdf@hbox
  {\let\null\empty % for \pageref
  \pdf@write{\pdf@w\space\pdf@d\space\pdf@h\space #2 #1 end}}}
```

Z definice je patrné, že makro pouze změří box a zapíše vhodné informace do DVI-souboru. Hlavní práce je ponechána PostScriptovým makrům.

Rozměry, které TEX vypočte a naším makrem zapíše do DVI-souboru, jsou uvedeny v bodech. Program DVIPS ovšem nechce spoléhat na aritmetiku v pohyblivé čárce, protože v některých PostScriptových RIPech se vyskytují chyby. Provádí tedy přepočet sám s ohledem na rozlišení výstupního zařízení. My se dopustíme drobného hříchu: pro převod rozměrů použijeme makra, která ovšem budou vyžadovat aritmetiku v pohyblivé čárce od PostScriptového RIPu. Horizontální a vertikální rozlišení uloží DVIPS do proměnných `Resolution` a `VResolution`. Pro převod pak napíšeme makra:

```
/HXD {exch Resolution 72 div mul round def} def
/VXD {exch VResolution 72 div neg mul round def} def
```

Často budeme vkládat hodnotu z vrcholu zásobníku do nějaké proměnné. Použijeme k tomu makro `XD` definované následujícím způsobem:

```
/XD {exch def} def
```

Vlastní makro pro vytvoření hypertextového odkazu má tři části. Ty se postupně postarají o všechny parametry makra. Prvními třemi parametry jsou šířka, hloubka a výška aktivního obdélníku. Následuje buď dvojice parametrů specifikující cíl odkazu a typ akce, nebo jeden parametr typu `array` osahující všechny potřebné informace. Makro vypadá následujícím způsobem:

```
/PDFannot {
  startPDFannot
  /pdf_w HXD
  endPDFannot
} def
```

Parametry jsou odebírány odzadu. Nejprve je provedeno makro `startPDFannot`:

```
/startPDFannot {
  dup type /arraytype eq
  { /pdf_action XD /pdf_target {aload pop} def }
  { /pdf_action XD /pdf_target XD } ifelse
  /pdf_h VXD /pdf_d VXD
  pdf_h /pdf_dx XD
} def
```

Toto makro uschová typ a cíl odkazu (všimněte si testování, zda se jedná o `array`) a výšku a hloubku textu. V zásobníku tedy zbyde šířka textu, kterou `PDFannot` uschová do `pdf_w`. Sestavení odkazu pak provede makro `endPDFannot`:

```
/endPDFannot {
  currentpoint dup pdf_h add /pdf_ury XD pdf_d sub /pdf_lly XD
  dup pdf_w sub /pdf_llx XD /pdf_urx XD
  [/Rect [pdf_llx pdf_lly pdf_urx pdf_ury]
   /Color [.3 0 .6]
   pdf_action pdf_target
   /Subtype /Link
   /ANN pdfmark
   /PDFlinkrect where {pop PDFlinkrect} if
} def
```

Jak vidíme, toto makro pouze vypočte absolutní souřadnice aktivního obdélníka a zapíše všechny parametry příkazu `pdfmark`, z něhož pak Distiller vytvoří hypertextový odkaz. Makro `PDFlinkrect` slouží ladicím účelům. Při prohlížení Ghostscriptem vykreslí barevný rámeček okolo aktivního odkazu, v PostScriptové tiskárně a v Distilleru je neúčinné. Jeho definici si nebudeme uvádět.

Vraťme se ještě k typům hypertextových odkazů. Pokud se odkazujeme na soubor nebo objekt na internetu, pak je cílem odkazu string obsahující jméno souboru nebo URL objektu. Odkazy uvnitř dokumentu lze realizovat různě. My ovšem budeme používat výhradně pojmenovaná místa. Pro zjednodušení si zavedeme pomocná makra:

```
% <width> <depth> <height> <dest> PDFlink
/PDFlink {/Dest PDFannot} def

% <width> <depth> <height> <url> PDFurl
/PDFurl {
  /pdf_url XD
  << /Subtype /URI /URI pdf_url >> /Action PDFannot
} def

% <width> <depth> <height> <file> PDFfile
/PDFfile {
  /pdf_file XD
  [ /Action /GoToR /File pdf_file ] PDFannot
} def
```

Snadno si ověříme, že právě tyto příkazy zapisuje makro `\pdfannot`.

Podobně jsme si vytvořili LaTeXové zkratky:

```
% Link
\DeclareRobustCommand\pdflink[1]{\pdfannot{PDFlink}{/#1}}
```

```
% URL
\DeclareRobustCommand\pdfurl[1]{\pdfannot{PDFurl}{(#1)}}

% Link to external file using /Action /GoToR
\DeclareRobustCommand\pdffile[1]{\pdfannot{PDFfile}{(#1)}}
```

Makra mají dva parametry. Prvním parametrem je cíl odkazu a makro jej naformátuje podle požadavků formátu PDF. Druhým parametrem je vlastní aktivní text, který bude zpracován až makrem `\pdfannot`.

Použitelnost těchto maker je omezena jen na jednoduché případy. Pokud je totiž aktivní text složen z několika slov odstavce, pak vzhledem k jeho sazbě do boxu mizí roztažitelnost mezislovních mezer. Pokud se takový text objeví na řádku, kde jsou mezery maximálně staženy nebo rozpáleny, pak je rozdíl ve velikosti mezer na první pohled patrný. Nemožnost úpravy šířky mezer může vést k jinému řádkovému zlomu a v krajním případě může ovlivnit i počet řádků odstavce. Ruční doladění je sice možné, ale to je přesně to, co nechceme.

Další potíž nastává při použití maker `\pdfurl` a `\pdffile`. Obvykle je aktivní text současně cílem hypertextového odkazu. U těchto maker však musíme cíl i aktivní text uvádět explicitně, tedy stejnou informaci zapisujeme dvakrát. Vzhledem k tomu, že tato makra jsou již překonána a v balíku přežívají hlavně z důvodů kompatibility se starými dokumenty, nebude tato nepříjemnost odstraněna.

4. Víceřádkové hypertextové odkazy

Nyní si ukážeme, jak lze rozdělit práci mezi LaTeX a PostScript. Měření boxů v LaTeXu je triviální. V PostScriptu sice také máme operátory pro měření velikosti objektů, ale jejich použití v již vysázeném textu by bylo značně komplikované. Při vytváření víceřádkového odkazu tedy zjistíme některé informace LaTeXovými makry, zapíšeme je do DVI-souboru a další práci přenecháme PostScriptu.

Víceřádkový odkaz budeme sázet makrem `\PDFannot`. Bude mít opět tři parametry. První z nich určuje typ odkazu. Jsou povoleny typy:

StartPDFlink pro odkaz uvnitř dokumentu,
StartPDFurl pro odkaz na objekt na Internetu,
StartPDFfile pro odkaz na soubor na lokálním disku.

Další dva parametry jsou stejné jako u makra `\pdfannot`, tedy cíl odkazu a aktivní text.

```
\def\PDFannot#1#2#3{%
  \setbox\pdf@hbox=\hbox{\let\\=\empty #3}%
  \edef\pdf@strut{\vrule width 0pt height \the\ht\pdf@hbox\space
                        depth \the\dp\pdf@hbox}%
```

```
  \pdf@dimensions
  \ifvmode \leavevmode \fi
  \pdf@write{\pdf@d\space\pdf@h\space #2 #1}\pdf@strut \kern\z@
  #3\kern\z@ \pdf@strut
  \pdf@@write{EndPDFmark}}
```

Činnost je poněkud odlišná. Všimněte si, že jsme sice aktivní text vysázeli do boxu, ale na předposledním řádku makra vysázíme tento text znovu. Použijeme tedy běžného algoritmu řádkového zlomu. Box je použit výhradně pro změření výšky a hloubky. Při sestavování tohoto pomocného boxu musíme nejprve předefinovat makro `\\`. Dále si nadefinujeme podpěru správných rozměrů a nakonec zavoláme již známé makro `\pdf@dimensions`. Před zápisem pomocných informací musíme ještě zajistit, abychom byli v odstavcovém režimu. Pokud bychom `\special` zapsali ve vertikálním režimu, utekl by nám od bodu sazby. Zapsané informace pak zpracujeme PostScriptovými makry.

V PostScriptové části budeme přepínat mezi starými a novými makry. Standardně budeme předpokládat stará makra, což vyjádříme nastavením:

```
/NewType false def
```

Další dvě makra nám pomohou ve stanovení aktivního obdélníku. První z nich uschová aktuální bod sazby, druhé z aktuálního bodu sazby a uschované hodnoty vypočte šířku aktivního obdélníka. Vypočtenou hodnotu vloží do proměnné `pdf_w`, kterou již známe z předchozích maker.

```
/cpt {currentpoint /pdf_Y_ XD /pdf_cpt XD} def
/ecpt {currentpoint pop pdf_cpt sub /pdf_w XD} def
```

Vlastní práci zahájí makro `StartPDFmark`:

```
/StartPDFmark {
  startPDFannot
  /NewType true def
  cpt
} def
```

Všechny parametry jsou zpracovány makrem `startPDFannot`. Poté se přepneme na nový typ maker a uschováme si bod sazby. Na konci odkazu máme makro `EndPDFmark`:

```
/EndPDFmark {
  ecpt
  pdf_w 0 gt {endPDFannot} if
  /NewType false def
  end
} def
```

Zde nejprve zjistíme šířku aktivního textu a v případě, že je kladná, zavoláme makro `endPDFannot`, které vytvoří hypertextový odkaz podle uložených hodnot. Pak zrušíme nastavení nového typu maker a uzavřeme slovník, který byl do zásobníku slovníků vložen LaTeXovým makrem. Takto jednoduše makra fungují,

pokud v odkazu nedošlo k řádkovému zlomu.

Nyní se musíme podívat, jaké příkazy používa DVIPS při sestavování stránky. Najdeme je v souboru `texc.pro` nebo v lidsky čitelnější podobě v `texc.lpro`. vybereme pouze ty, které jsou pro nás důležité:

```
/N {def} def
/B {bind def} N
/p {show} N
/a {moveto} B
/y {3 2 roll p a} B
```

Všimněte si, že pro často používané sekvence jsou vytvořeny jednopísmenné zkratky. Stejný přístup využívají všechny komerční programy, které generují PostScriptové soubory. Výsledné soubory jsou sice nečitelné, ale jsou výrazně kratší.

Nejzajímavější je pro nás makro `/y`. To je používáno k absolutnímu posunu bodu sazby, tedy zejména k přechodu na nový řádek. Jiný typ přechodu na nový řádek jsem v soborech generovaných programem DVIPS nenašel. Jakmile tedy narazíme v souboru na makro `y`, víme, že je nutno ukončit aktivní obdélník a parametry makra nám určují souřadnice počátku následujícího aktivního obdélníku. Ve skutečnosti musíme ještě ošetřit několik dalších komplikací, ale nejprve se podívejme na nové makro:

```
/y {
  3 2 roll p
  NewType
    {1 index currentpoint pop sub pdf_dx lt 1 index
     pdf_Y_ sub pdf_h neg gt or
      {ecpt pdf_w 0 gt
        {endPDFannot} if
        a cpt
      }
      {a}
      ifelse
    }
    {a}
  ifelse
} def
```

Nejprve vytiskneme připravený text. Pak si ověříme, že zpracováváme skutečně makro nového typu. Za určitých okolností, které vysvitnou později, se toto makro může objevit i při zpracování `PDFannot`. Pak nám vnější podmínka `ifelse` zajistí, že se provede pouze posun bodu sazby. Makra nového typu vyžadují ještě další vnořenou podmínku. Při jejím splnění provádíme podobnou činnost jako při ukončení odkazu. Pomocí makra `ecpt` vypočteme šířku obdélníka, makrem `endPDFannot` vytvoříme odkaz, posuneme bod sazby a jeho novou

polohu uložíme prostřednictvím makra `cpt`. Tímto mechanismem rozdělíme víceřádkový odkaz na několik obdélníků se stejnou akcí a stejným cílem (tyto informace byly zaznamenány makrem `startPDFannot` a nebyly přepsány).

Zbývají nám dvě zmíněné komplikace. Makro `y` totiž slouží též ke vkládání kerningu a k vertikálním posunům. Například ve slově LaTeX budou všechna vnitřní písmena považována za samostatné řádky a odkaz se tak zbytečně rozpadne do pěti obdélníků. Složitá vnitřní podmínka tedy testuje, zda je posun dostatečně velký. Při malém posunu se odkaz nerozdělí a pouze se posune bod sazby.

Druhý problém je záludnější. Kvůli optimalizaci používá DVIPS řadu dalších zkratek pro malé posuny. Při nich se hodnota horizontální mezery ukládá do proměnné `delta` a teprve později dochází k posunu. Za určitých podmínek je tato definice vložena do našeho slovníku, který je odstraněn dříve, než ke skutečnému posunu dojde. Tím dochází ke ztrátě některých mezislovních mezer. Musíme tedy zajistit, aby se definice vždy ukládaly do slovníku `TeXDict`, v němž má DVIPS svá makra. Naštěstí používá DVIPS pro definice výhradně svoji zkratku `N`. Nápravy tedy docílíme jednoduchou definicí:

```
/N {TeXDict 3 1 roll put} def
```

Máme-li smůlu, dostane se do víceřádkového odkazu stránkový zlom. Program DVIPS ovšem vkládá stránky mezi `save` a `restore`, takže všechny uschované hodnoty budou ztraceny. Uděláme tedy alespoň po sobě úklid, aby nás nepřekvapovaly PostScriptové chyby.

```
/eop {/NewType false def end eop} def
```

Zde jsme nastavili starý typ makra (asi je to nadbytečné) a po zavření slovníku zavoláme původní `eop`. Na příští stránce ovšem najdeme `EndPDFmark`. Toto makro již nesmí provádět žádnou činnost, ale musí být definováno. Vnoříme tedy jeho prázdnou definici do slovníku `userdict`:

```
userdict /EndPDFmark {} put
```

Stejně jako v minulé části si zavedeme pomocná makra, která připraví typ akce a cíl odkazu:

```
% <depth> <height> <dest> StartPDFlink
/StartPDFlink {/Dest StartPDFmark} def

% <depth> <height> <url> StartPDFurl
/StartPDFurl {
  /pdf_url XD
  << /Subtype /URI /URI pdf_url >> /Action StartPDFmark
} def

% <depth> <height> <file> StartPDFfile
/StartPDFfile {
  /pdf_file XD
```

```
  [ /Action /GoToR /File pdf_file ] StartPDFmark
} def
```
Pro uživatele vytvoříme odpovídající LaTeXová makra:
```
% Link
\DeclareRobustCommand\PDFlink[1]{\PDFannot{StartPDFlink}{/#1}}

% URL
\DeclareRobustCommand\PDFurl{\PDFpath{StartPDFurl}}

% Link to external file using /Action /GoToR
\DeclareRobustCommand\PDFfile{\PDFpath{StartPDFfile}}
```
Již zmíněnou výhodou těchto maker je, že umožňují řádkový zlom. Makra `\PDFurl` a `\PDFfile` mají ještě další příznivou vlastnost: jejich jediný parametr slouží současně jako cíl odkazu i jako aktivní text. Ušetříme si tedy psaní a omezíme možnost překlepů. Funkci těchto maker si podrobněji popíšeme v následující kapitole.

5. Odkazy na externí soubory a internetové zdroje

URL zdroje na Internetu jakož i plné jméno souboru může být dlouhé, a proto někdy vyžaduje řádkový zlom. Pro tyto účely vytvořil Philip Taylor balík `path.sty`. Vhodné znaky jsou aktivní a expandují na `\discretionary`. V této podobě budeme v poslední fázi text sázet. Pro zápis cíle odkazu však potřebujeme prostý text. Nemůžeme tedy jednoduše pozřít parametr makra a vložit jej na dvě místa v různých podobách. Musíme si pomoci trochu nečistým trikem.

Uvedli jsme, že chceme mít stejný řádkový zlom jako bez použití maker pro tvorbu PDF. Dále požadujeme co nejmenší zásahy do textu. Odkazy tedy budeme vytvářet pouhou náhradou `\PDFurl` či `\PDFfile` místo původního `\path`. Syntakticky musí všechna makra vypadat stejně. V některých případech však můžeme vyžadovat, aby skutečný odkaz směřoval jinam, než uvádí následující text. Proto připustíme první nepovinný parametr v hranatých závorkách, který definuje cíl odkazu.

V předchozí kapitole jsme napsali, že odkazy tohoto typu vytváříme makrem `\PDFpath`. Zde je jeho definice:
```
\DeclareRobustCommand\PDFpath[1]{%
  \def\pdf@annot@type{\PDFannot{#1}}\pdf@path}
```
Nejprve si definujeme typ akce a potom voláme pomocné makro, které umožní zpracování nepovinného parametru:
```
\def\pdf@path{\let\pdf@path@dest\relax
  \@ifnextchar[\pdf@path@prep\pdf@path@exec}
```
První příkaz zrušil případnou předchozí definici cíle odkazu a dále se přepínáme

podle toho, zda následuje levá hranatá závorka.

Další dvě makra uschovávají cíl odkazu. Protože spoléháme na `path.sty`, vypůjčíme si makro `\c@tcodes`, které změní kategorii všech speciálních znaků na 12.

```
\def\pdf@path@prep{\begingroup
  \c@tcodes=12 \pdf@path@optarg}

\def\pdf@path@optarg[#1]{\endgroup
  \def\pdf@path@dest{#1}\pdf@path@exec}
```

Po provedení těchto maker se dostáváme k příkazu `\pdf@path@exec`:

```
\def\pdf@path@exec{\begingroup
  \let\p@th\pdf@hacked@path \path}
```

Chceme mít stejný řádkový zlom, jaký by vytvořilo samotné makro `\path`. Abychom zajistili správný zápis cíle odkazu, musíme pozměnit chování makra. Zavoláme jej tedy uvnitř skupiny, v níž místo jednoho z vnitřních maker vložíme vlastní verzi. Tu zde uvedeme i s původními komentáři Philipa Taylora:

```
\def \pdf@hacked@path #1%% <delim>
% it starts by opening a group (ended in \p@@th) ...
% switches to \tt, and inhibits hyphenation;
% allows breaks at \discretionaries;  saves the
% catcode of the initial <delim>, because if it has also been
% declared within \discretionaries, the initial and final <delim>
% won't match (which would be a disaster); \c@tcode is overloaded,
% but there is no conflict that I can detect ...
% invokes \discr@ti@n@ri@s to render active the set of special
% characters which have previously been declared as
% \discretionaries; each of these characters will expand to a real
% \discretionary, with replacement texts <self> <null> <self>; and
% re-instates the catcode of the initial <delim>.
{\begingroup
   \tt
   \c@tcode = \catcode `#1
   \discr@ti@n@ri@s
   \catcode `\ = \active
   \expandafter \edef \activesp@ce {\passivesp@ce \hbox {}}%
   \catcode `#1 = \c@tcode
   % it next defines an inner macro \p@@th with delimited parameter
   % structure, the final delimiter being the same as the initial
   % delimiter which it has itself received as #1.
   \def \p@@th ##1#1% <chars> <delim>
      % Within \p@@th,
      % permissible breakpoints are specified,
```

```
      % the destination for PDF is obtained,
      % the path is typeset as a link,
      % and the group(s) ended.
     {\ifx\pdf@path@dest\relax
       \begingroup
         % The contents of \pdf@path@dest should be written
         % as a PostScript string within a \special. Therefore
         % it may only contain ordinary characters. Unfortunately,
         % a great many characters were made active and defined
         % as discretionaries. In order to make it work, we
         % redefine \char and \discretionary. After these
         % modifications all characters will be expanded
         % to normal letters.
         \def\char`{\noexpand\noexpand\noexpand}%
         \def\discretionary####1####2####3{####3}%
         \xdef\pdf@path@dest{##1}%
       \endgroup
      \fi
      \ifvmode \leavevmode \fi
      \pdf@annot@type{(\pdf@path@dest)}{##1}%
      \endp@th
      \endgroup % started in \pdf@path@exec
     }%
\p@@th
}
```

Změna proti původní verzi se vyskytuje v makru `\p@@th`. Pokud jsme makro `\pdf@path@dest` nenadefinovali, musíme před vysázením sestavit řetězec pro cíl odkazu. Mnoho znaků je ale aktivních a při zápisu uvnitř `\special` by docházelo k nežádoucí expanzi. Musíme tedy předefinovat `\discretionary` tak, aby při rozvoji zapsalo pouze svůj poslední parametr. Podíváte-li se na makra v `path.sty`, která mění kategorie znaků, zjistíte, že řada znaků se do výstupu dostane pomocí primitivu `\char`. Navíc tyto znaky také mohou být aktivní. To jsou další dvě komplikace, které by nám zničily řetězec s cílem odkazu. Musíme tedy předefinovat primitiv `\char`, přičemž necháme automaticky odstranit zpětný apostrof. K první expanzi znaku dojde v definici `\xdef` o dva řádky níže, druhá expanze je prováděna uvnitř `\special`. Abychom tedy nežádoucí expanzi znaku zabránili, potřebujeme tři primitivy `\noexpand`. Vlastní sazba odkazu je pak triviální. Protože cílem odkazu je vždy string, vložíme jej do kulatých závorek. Nakonec uzavřeme skupinu, čímž se makru `\p@th` vrátí jeho původní význam.

6. Návěští a jména

Uvedli jsme, že všechny odkazy uvnitř dokumentu budou směřovat výhradně na pojmenovaná místa. Tato jména budeme vždy odvozovat od čísla stránky a vytvoříme si pro ně následující makra:

```
\def\pdfpage#1{@#1@}
\DeclareRobustCommand\pdfdest[1]{\pdf@write
    {/\pdfpage{#1} PDFdest end}}
```

V PostScriptové části nadefinujeme odpovídající makro, které zapíše potřebný příkaz pro definici návěští v PDF-souboru.

```
/PDFdest {
  /pdf_dest XD
  [/Dest pdf_dest /View [/FitV 0] /DEST pdfmark
} def
```

Abychom stránky pojmenovávali automaticky, musíme si předefinovat živé záhlaví či zápatí. Pro styly `plain` a `empty` je to učiněno následovně:

```
\def\ps@plain{%
  \def\@oddhead{}%
  \def\@evenhead{}%
  \def\@oddfoot{\hfil\normalfont\thepage\pdfdest{\thepage}\hfil}%
  \let\@evenfoot\@oddfoot}

\def\ps@empty{%
  \def\@oddhead{}%
  \def\@evenhead{}%
  \def\@oddfoot{\pdfdest{\thepage}\hfil}%
  \let\@evenfoot\@oddfoot}
```

Používáte-li jiné styly, musíte je předefinovat obdobně.

Tato jména použijeme pro mechanismus odkazů `\ref` a `\pageref` na místa označená návěštím `\label`. Hypertextové odkazy vytvoříme pomocí speciálních maker `\PDFref` a `\PDFpageref`. Obě makra mají dva parametry, z nichž první je nepovinný, a uvádí se tudíž v hranatých závorkách. Druhý parametr představuje jméno návěští stejně jako v makrech `\ref` a `\pageref`. První parametr specifikuje text, který má být aktivován. V tomto parametru můžeme použít speciální makro `\PDFreftext`, které bude expandováno na výsledek makra `\ref` resp. `\pageref`. Chceme-li tedy napsat, že o víceřádkových odkazech se píše v kapitole 4, přičemž celý text „v kapitole 4“ má být aktivní, použijeme zápis:

```
... se píše \PDFref[v~kapitole~\PDFreftext]{zw:multiline}, ...
```

Obě makra provádějí velmi podobnou činnost. Proto nejprve uloží požadovaný typ a zavolají další makro, které uskuteční žádanou akci.

```
\DeclareRobustCommand\PDFref{\let\PDF@reftype\ref \PDF@setref}
\DeclareRobustCommand\PDFpageref{\let\PDF@reftype\pageref
```

```
                \PDF@setref}
```
Při prvním průchodu nejsou odkazy nalezeny. Pokud bychom vytvářeli hypertextové odkazy na nenalezená návěští, skončilo by to z hlediska uživatele nepochopitelnými chybovými zprávami. Proto musíme testování provádět sami. Pouze v případě, že návěští existuje, můžeme zapsat informaci pro tvorbu hypertextového odkazu do DVI-souboru. Protože toto makro neexistovalo v původní verzi balíku, vytvoříme nyní pouze variantu, která umožňuje víceřádkový odkaz.
```
\newcommand\PDF@setref[2][\PDFreftext]{%
  \def\PDFreftext{\PDF@reftype{#2}}%
  \expandafter\ifx\csname r@#2\endcsname\relax
    \@latex@warning {Reference `#2' on page \thepage
                     \space undefined}%
    \def\PDFreftext{??}\def\PDF@next{#1}%
  \else
    \def\PDF@next{\PDFlink{\pdfpage{\expandafter
      \expandafter\expandafter
      \@secondoftwo\csname r@#2\endcsname}}{#1}}%
  \fi \PDF@next}
```

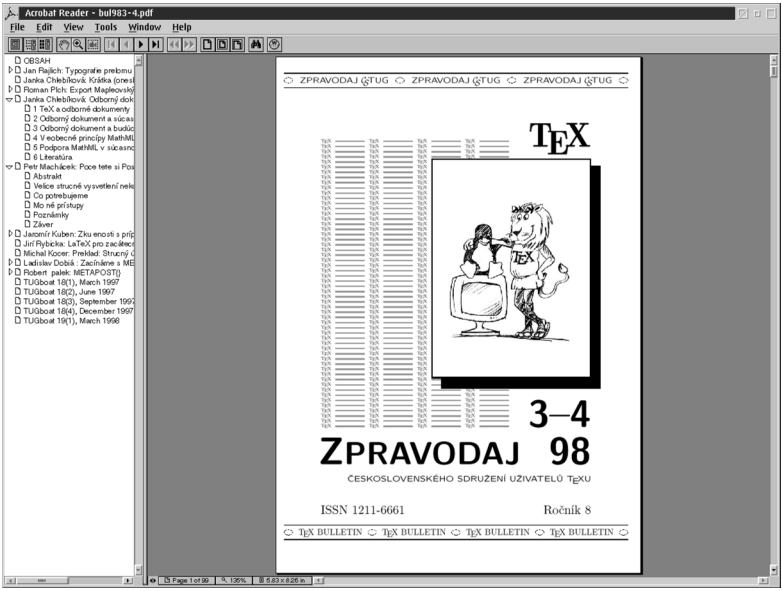
7. Záložky

Záložky neboli „outlines“, často nazývané též „bookmarks“, representují obsah dokumentu, který Acrobat Reader volitelně zobrazuje v levé části okna. V Ghostscriptu verze 5.50 a nižší nejsou záložky zobrazovány vůbec. Záložky mohou být strukturovány nejvýše do tří úrovní. Po zobrazení záložek mohou být nižší úrovně otevřeny (tj. viditelné), nebo zavřeny. Uživatel je pak může otevírat a zavírat kliknutím na odpovídající šipku. Příklad PDF-dokumentu s otevřenými i zavřenými záložkami najdete na obrázku 1.

Každá záložka má text (titul) a přiřazený cíl, na nějž se odskakuje, pokud je záložka vybrána. Dále obsahuje počet položek nižších úrovní. Je-li toto číslo záporné, pak budou nižší úrovně standardně zavřeny. Původní verze tohoto balíku nepodporovala víceúrovňové záložky. Zbyla nám tedy makra definující pouze základní úroveň.

Nejprve si ukážeme PostScriptovou část, protože je velmi jednoduchá. Celou práci zde bude provádět LaTeX.
```
% <title> <dest> PDFtoc
/PDFtoc {0 PDFctoc} def

% <title> <dest> <count> PDFctoc
/PDFctoc {
  /pdf_cnt XD
```



Obrázek 1: Příklad PDF-dokumentu se záložkami

```
  /pdf_dest XD
  /pdf_tit XD
  [/Title pdf_tit
   pdf_cnt dup 0 eq {pop} {/Count exch} ifelse
   /Dest pdf_dest /OUT pdfmark
} def
```

Do DVI-souboru zapíšeme příslušné informace těmito LaTeXovými makry:

```
\def\l@pdf#1#2{% PDF title, Page number
  #1%
  \pdf@write{(\pdf@temp) /\pdfpage{#2} PDFtoc end}}
\def\l@pdfc#1#2#3{% PDF title, Page number, Count
  #1%
  \pdf@write{(\pdf@temp) /\pdfpage{#2} #3 PDFctoc end}}
```

Záložky musí mít kódování `/PDFDocEncoding`. To ovšem neobsahuje všechny české a slovenské znaky. Navíc i znaky, které jsou v tomto kódování definovány, nejsou zobrazeny v některých verzích Acrobat Readeru. Tento problém by podle firmy Adobe měl být odstraněn v Acrobat Readeru 4.0. Již nyní by záložky v PDF měly podporovat UNICODE, ale zatím to nebylo testováno.

LaTeX umožňuje překódování znaků. Není tedy problém vytvořit příslušné

kódování a nadefinovat si fiktivní font:

```
\input{PDFenc.def}
\DeclareFontFamily{PDF}{PDF}{}
\DeclareFontShape{PDF}{PDF}{m}{n}{<->pdfdoc}{}
\DeclareFontFamily{PDF}{cmr}{}
\DeclareFontShape{PDF}{cmr}{m}{n}{<->pdfdoc}{}
```

Vraťme se ale k úrovním záložek. Můžeme je strukturovat až do tří úrovní, což je méně, než připouští LaTeX. V některých případech tedy budeme nuceni LaTeXové úrovně nadpisů sdružovat, pokud budeme chtít, aby v záložkách byly viditelné. Jednotlivým typům nadpisů přiřadíme úroveň (smysl mají hodnoty od nuly do tří) a čítačem `PDFtocdepth` určíme, které úrovně záložek budou viditelné a které budou otevřeny. Vše ozřejmí tabulka 1.

Tabulka 1: Přehled úrovní záložek, × znamená neviditelné

	Úroveň			
PDFtocdepth	0	1	2	3
−2	otevřené	zavřené	zavřené	×
−1	otevřené	zavřené	×	×
0	otevřené	×	×	×
+1	otevřené	otevřené	×	×
+2	otevřené	otevřené	zavřené	×
+3	otevřené	otevřené	otevřené	×

Makra pro nastavení úrovní přiřazují stejnou hodnotu počínaje specifikovaným typem a pokračují do všech nižších typů LaTeXových nadpisů. Pořadí specifikací je tedy důležité. Zde jsou příslušné definice:

```
\def\PDFsubparagraph#1{\def\pdf@subparagraph{#1}}
\def\PDFparagraph#1{\def\pdf@paragraph{#1}\PDFsubparagraph{#1}}
\def\PDFsubsubsection#1{%
    \def\pdf@subsubsection{#1}\PDFparagraph{#1}}
\def\PDFsubsection#1{\def\pdf@subsection{#1}\PDFsubsubsection{#1}}
\def\PDFsection#1{\def\pdf@section{#1}\PDFsubsection{#1}}
\def\PDFchapter#1{\def\pdf@chapter{#1}\PDFsection{#1}}
\def\PDFpart#1{\def\pdf@part{#1}\PDFchapter{#1}}

% Default settings
\PDFpart 0 \PDFsubsection 3
\newcounter{PDFtocdepth} \setcounter{PDFtocdepth}{0}
```

Záložky by měly být vytvářeny s co nejmenším úsilím. Jako nejvhodnější se jeví předefinování makra `\contentsline`. Způsob tvorby záložek musí být ovšem dostatečně flexibilní. Proto si nadefinujeme vlastní makro

\PDF@contentsline a budeme předpokládat, že zápis do obsahu provede nové makro \ZW@contentsline.

```
\def\PDF@contentsline#1#2#3{{%
  \def\footnote##1{}%
  \addtocontents{bkm}{\protect\pdfline
     {\csname pdf@#1\endcsname}{#2}{#3}}%
  \ZW@contentsline{#1}{\pdftoclink
     {\pdfpage{#3}}{#2}}{\pdflink{\pdfpage{#3}}{#3}}}}
```

Makro plní hned dvě funkce. Pomocí \addtocontents vytvoříme záznam v pomocném souboru, který při dalším průchodu vygeneruje záložky v PDF. V makru \ZW@contentsline definujeme hypertextové odkazy uvnitř vytištěného odkazu. Odkaz kolem stránkové číslice je vytvořen jednoduchým makrem, které jsme si již popsali. Za zmínku stojí jen způsob definice cílu odkazu s využitím dalšího známého makra \pdfpage. Složitější je hypertextový odkaz spojený s textem. Zde se často vyskytuje číslo kapitoly. Zahrnutí čísla kapitoly do aktivního textu obvykle nepůsobí esteticky. K jeho vyloučení potřebujeme další nečistý trik. Makro, které příslušnou činnost zahájí, musí mít dva parametry: cíl odkazu a text nadpisu kapitoly.

```
\DeclareRobustCommand\pdftoclink[2]{%
  \def\pdf@linkdest{#1}\pdf@toclink #2\@pdf}
```

Cíl odkazu jsme si uschovali a můžeme zavolat následující makro:

```
\def\pdf@toclink #1#2\@pdf{\begingroup
  \ifx #1\numberline
    \let\pdf@numberline\numberline
    \def\numberline ##1{\pdf@numberline{##1}\pdf@@@link}%
  \else
    \let\pdf@toc@link\pdf@@@link
  \fi \pdf@toc@link #1#2\@pdf}
```

Za okamžik budou hrát neméně důležitou roli další dvě makra:

```
\def\pdf@toc@link{}
\def\pdf@@@link #1\@pdf{\endgroup
  \PDFlink{\pdf@linkdest}{#1}}
```

V makru \pdf@toclink budeme měnit definice standardních příkazů. Otevřeme si tedy skupinu, aby tyto změny byly jen lokální. Pak otestujeme, zda je prvním tokenem řídicí sekvence \numberline. V kladném případě musíme zařídit, aby při vytváření hypertextového odkazu byla tato sekvence i se svým parametrem přeskočena. Proto si uschováme původní definici do \pdf@numberline a nadefinujeme si nový význam. Po ukončení podmínky tedy budeme mít ve vstupním proudu token \pdf@toc@link následovaný celým původním textem a oddělovačem \@pdf. Expanze prvního tokenu je prázdná. Makro \numberline pak bude nahrazeno makrem \pdf@numberline s parametrem, ale do tohoto makra jsme schovali původní význam \numberline. Následující makro pozře

zbývající text a vytvoří z něj hypertextový odkaz, v němž je povolen řádkový zlom. Přitom je nutno zavřít skupinu, kterou jsme pře okamžikem otevřeli. Číslo kapitoly bylo tímto trikem přeskočeno.

Zbývá ještě probrat případ, kdy text názvu kapitoly neobsahuje příkaz `\numberline`. Zde stačí, když předefinujeme `\pdf@toc@link` tak, aby expandovalo přímo na `\pdf@@@link`. Veškerá činnost je tím hotova.

Uvedený přístup nabízí další možnosti, k nimž nám pomohou následující makra:

```
\newcommand\PDFnotoclinks{\DeclareRobustCommand
                          \pdftoclink[2]{##2}}
\newcommand\PDFfulltoclinks{\DeclareRobustCommand
                            \pdftoclink{\PDFlink}}
```

Makrem `\PDFnotoclinks` zařídíme, že hypertextové odkazy budou spojeny pouze se stránkovou číslicí. Při použití makra `\PDFfulltoclinks` budou do hypertextového odkazu zahrnuty i čísla kapitol.

Nyní se vrátíme zpět k makru `\PDF@contentsline`. V něm jsme také zapsali do pomocného souboru s příponou `bkm` příkaz `\pdfline` podobný standardnímu `\contentsline`. Makro `\pdfline` má také tři parametry: úroveň, text a číslo stránky. Pro zápis jsme použili `\addtocontents`, protože číslo stránky již známe. Zpracování proběhne až při dalším průchodu a budeme přitom potřebovat tři čítače. Ty budou střádat počty nadpisů ve vnořených úrovních, musí tedy být navzájem sunchronizovány podobným způsobem, jako LaTeXové čítače pro číslování kapitol:

```
\newcounter{pdf@0}
\newcounter{pdf@1}[pdf@0]
\newcounter{pdf@2}[pdf@1]
```

Při tvorbě záložek ovšem potřebujeme znát počet vnořených názvů předem. Máme tedy jedinou možnost: při prvním průchodu zapsat pomocné informace do souboru a v dalším průchodu je načíst. Připravíme si makra, která tuto činnost zajistí.

```
\def\pdf@aux#1#2{\immediate\write\@mainaux
                 {\string\pdf@count@def{#1}{#2}}}
\def\pdf@count@def#1{\expandafter\gdef\csname zw@pdf@#1\endcsname}
```

Následující makro nejprve inkrementuje požadovaný čítač, poté připraví makro `\pdf@level` podle již známé hodnoty (ta obvykle pochází z předchozího průchodu) a nakonec zapíše informace do pomocného souboru. Všimněte si, že pro danou úroveň zapisujeme postupně definice stejných maker s jinou hodnotou. Při načítání pomocného souboru v příštím průchodu se „zapamatuje“ pouze poslední definice, což je přesně to, co potřebujeme.

```
\def\pdf@step#1{%
  \stepcounter{pdf@#1}%
  \let\pdf@prefix\empty
```

```
  \ifnum\c@PDFtocdepth<0 \def\pdf@prefix{-}\fi
  \ifcase #1
    \edef\pdf@level{zw@pdf@\arabic{pdf@0}}%
  \or
    \edef\pdf@level{zw@pdf@\arabic{pdf@0}.\arabic{pdf@1}}%
    \pdf@aux{\arabic{pdf@0}}{\pdf@prefix\arabic{pdf@1}}%
  \else
    \def\pdf@level{relax}%
    \ifnum\c@PDFtocdepth=2 \def\pdf@prefix{-}\fi
    \pdf@aux{\arabic{pdf@0}.\arabic{pdf@1}}%
            {\pdf@prefix\arabic{pdf@2}}%
  \fi}
```

Nyní se již můžeme podívat na makro `\pdfline`. Pro zjištění, které úrovně mají být viditelné, potřebujeme čítač, jehož hodnota bude v intervalu od nuly do dvou. Původní hodnotu čítače `PDFtocdepth` však musíme zachovat. Potřebujeme tedy pomocný čítač. Po zavolání makra `\pdf@step` zjistíme, zda máme vložen počet nadpisů nižších úrovní v makru `\pdf@level`, a podle toho použijeme příslušný typ zápisu.

```
\def\pdfline#1#2#3{%
  \count@=\c@PDFtocdepth
  \ifnum\count@<0 \count@=-\c@PDFtocdepth\fi
  \ifnum\count@>2 \count@=2\fi
  \ifnum #1>\count@ \else
    \pdf@step{#1}%
    \ifPDFdebug \typeout{PDFline{#1}{#2}}\fi
    \@ifundefined{\pdf@level}%
      {\ifPDFdebug \typeout{\pdf@level \space undefined}\fi
        \l@pdf{\pdf@dest{#2}}{#3}}%
      {\ifPDFdebug \typeout{\pdf@level
          \space = \csname \pdf@level\endcsname}\fi
        \l@pdfc{\pdf@dest{#2}}{#3}{\csname\pdf@level\endcsname}}%
  \fi}
```

Makro `\contentsline` je používáno i v seznamu obrázků a tabulek. Tyto informace by ovšem v záložkách být neměly. Změnu definice tedy provedeme uvnitř modifikovaného `\tableofcontents`:

```
\let\PDF@ZW@tableofcontents\tableofcontents

\def\tableofcontents{\PDFlisting
  \let\ZW@contentsline\contentsline
  \let\contentsline\PDF@contentsline
  \PDF@ZW@tableofcontents
  \let\contentsline\ZW@contentsline}
```

Záložky pak zpracuje makro \PDFlisting:

```
\def\PDFlisting{\begingroup
  \pdf@setup
  \def\numberline##1{##1 }%
  \def\az{-}\def\pdf@dest{\def\pdf@temp}%
  \csname pdf@hook\endcsname
  \input{8859def}\usefont{PDF}{PDF}{m}{n}
  \@starttoc{bkm}\endgroup}
```

Činnost je analogická standardnímu vytváření obsahu makrem \@starttoc. Protože texty musí být kódovány podle PDFDocEncoding, zavedeme nejprve soubor 8859def.tex, který změní kategorie všech akcentovaných znaků (podle ISO 8859-2, tj. kód $\mathcal{CS}$-fontů) na aktivní a definuje je pomocí příslušných řídicích sekvencí, a zvolíme fiktivní font. V názvech kapitol ale mohou být použita nejrůznější makra. V záložkách jsou povoleny jen prosté texty. Musíme tudíž řadu maker předefinovat. Základní makra pro různá TEXová loga, přepínače fontů apod. jsou uvedena v makru \pdf@setup. Zde je pouze jeho část pro ilustraci:

```
{\catcode`\|=0 |catcode`\\=12
|gdef|PDF@bs{\\}}

\def\pdf@setup{%
  \let\@typeset@protect\empty\set@typeset@protect
  \def\footnote##1{}%
  \def\TeX{TeX}%
  \def\LaTeX{LaTeX}%
  \def\LaTeXe{LaTeX2e}%
  \let\raggedright\empty
  \let\noindent\empty
  \let\leavevmode\empty
  ...
  \def\\{}%
  \def\dots{\textellipsis}%
  \def\cmd##1{\PDF@bs##1}%
  \def\THANH{Han The Thanh}%
  \let\raggedright\empty
  \let\ignorespaces\empty
  \let\footnote\@gobble
  \let\mdseries\empty
}
```

Velmi důležitý je první řádek. Potřebujeme totiž, aby expanze makra \protect byla prázdná, nikoliv \relax. Pokud bychom však provedli jednoduché přiřazení \let\protect\empty, skončilo by to katastrofou. Zde použitý postup je jediný

správný. Pro uživatelské definice je ponecháno makro `\pdf@hook`. To je voláno prostřednictvím `\csname`, takže nevadí, pokud jej uživatel nenadefinuje.

Zbývá ještě vysvětlit, proč definujeme makro `\pdf@dest`. Při zápisu záložek totiž musíme expandovat jak všechna makra tak překódovat akcentované znaky. Není jednoduché zařídit, aby všechny expanze proběhly v požadovaném okamžiku. Makro `\pdfline` proto vkládá parametry pro `\l@pdf` a `\l@pdfc` do pomocného makra `\pdf@dest`. Makra `\l@pdf` i `\l@pdfc` provedou první parametr, což v našem případě nadefinuje makro `\pdf@temp`. Toto makro je pak zapsáno jako `\special` do DVI-souboru, přičemž dojde k požadovaným expanzím. Nepodařilo se mi objevit jiný způsob, který by za všech okolností provedl všechny požadované činnosti.

Uvedené definice umožňují ovlivňovat hypertextové odkazy, které se vytvoří. Pokud nechceme mít v dokumentu záložky, stačí předefinovat:
`\let\PDFlisting\relax`
Pokud nechceme ani hypertextový obsah, napíšeme:
`\let\tableofcontents\PDF@ZW@tableofcontents`

8. Nedostatečně a přespříliš robustní makra

Při zápisu názvů kapitol do pomocných souborů nesmíme připustit expanzi. LaTeX dělí makra do dvou skupin: na robustní, která v těchto situacích expandována nejsou, a křehká, která se expandují a uživatel musí expanzi zabránit sám.

LaTeX 2.09 používal pro definici robustních maker následující trik:
```
\def\RobustMacro{\protect\pRobustMacro}
\def\pRobustMacro{... definice ...}
```
Z toho též vycházelo doporučení, jak zacházet s křehkými makry. Pokud jsme chtěli použít křehké makro v názvu, museli jsme před ně psát `\protect`.

Podívejme se, co se stane při zápisu takového makra do pomocného souboru pro tvorbu obsahu a později pro záložky. Pokud zapíšeme do názvu kapitoly `\RobustMacro`, dojde při zápisu k expanzi na `\protect\pRobustMacro`. Ochranné makro `\protect` přitom zmizí, ale `\pRobustMacro` expandováno nebude. Při sazbě obsahu narazíme na `\pRobustMacro` a expandujeme jej. Zde je vše v pořádku, ale problém nastane při zápisu tohoto makra do souboru pro tvorbu záložek. Toto makro je totiž křehké a pravděpodobně způsobí nějaký typ katastrofy. Abychom tomu zabránili, museli bychom před `\pRobustMacro` napsat `\protect\protect\protect`. Tak je nutno zacházet s křehkými makry používanými v názvech.

Lepší mechanismus robustních maker nabízí LaTeX 2_ε. Při použití definičního makra `\DeclareRobustCommand` se totiž vytvoří pomocné makro pojmenované `\csname␣RobustMacro␣\endcsname`. Všimněte si koncové mezery. Ta při načí-

tání pomocného souboru bude mít opět kategorii 10 a podle známých pravidel bude spolknuta. Na vstupu tedy dostaneme původní robustní makro a zápis do souboru pro tvorbu záložek nebude činit žádné potíže.

Reliktem ze starých dob je makro `\numberline`. To je křehké a standardně se vkládá prostřednictvím `\protect`. Přebytečný `\protect` nám obvykle nevadí, takže problém vyřešíme tím, že makro předefinujeme novým způsobem. Nabízí se automatické použití příkazů:

```
\let\ZW@PDF@numberline\numberline
\DeclareRobustCommand\numberline{\ZW@PDF@numberline}
```

To bude fungovat do okamžiku, než někdo nadefinuje `\numberline` ve svém stylu skutečně robustně pomocí `\DeclareRobustCommand`. Podívejme se, co se stane.

Pokud bylo `\numberline` křehké nebo definováno jako robustní způsobem, který užíval LaTeX 2.09, pak jeho nová definice bude nejprve expandovat na `\protect` a `\csname␣numberline␣\endcsname`. Druhý token pak bude expandovat na původní význam. Pokud ale bylo `\numberline` definováno pomocí `\DeclareRobustCommand`, pak uschovaná definice obsahuje makro, jehož expanzí je také `\protect` následovaný tokenem `\csname␣numberline␣\endcsname`. Výsledkem takové přespříliš robustní definice bude nekonečná smyčka.

Při definici tedy musíme být rafinovanější. Napřed si vytvoříme pomocné makro simulující robustní `\numberline`. Pokud se shoduje s definicí `\numberline`, nebudeme provádět nic, protože makro je již dostatečně robustní. Předefinování použijeme v opačném případě. Abychom snížili pravděpodobnost, že nám uživatel dodatečně podsune vlastní křehkou definici, provedeme tyto akce později prostřednictvím `\AtBegindocument`:

```
\AtBeginDocument{% Fix \numberline
  \edef\ZW@PDF@numberline{\noexpand\protect
    \expandafter\noexpand\csname numberline \endcsname}%
  \ifx\ZW@PDF@numberline\numberline % it was already robust
  \else
    \let\ZW@PDF@numberline\numberline
    \DeclareRobustCommand\numberline{\ZW@PDF@numberline}%
  \fi}
```

Pokud zapomeneme v názvech kapitol nedostatečně robustní makra, začnou se dít (většinou až při třetím průchodu) podivné věci. Někdy se LaTeXová makra pomatou natolik, že budou opakovaně načítat týž soubor. To ovšem brzy povede k tomu, že TeX ohlásí příliš mnoho otevřených souborů. Pak je vhodné prohlédnout si soubory s příponami `toc` a `bkm`. Obvykle v nich snadno najdeme podivně expandovaná makra a vystopujeme křehké makro, které příslušný problém způsobilo.

9. Nastavení vlastností dokumentu

Prvním parametrem, který musíme nastavit, je velikost stránek. V PDF-souborech máme jednak tzv. MediaBox, a současně CropBox. Oba se zadávají jako souřadnice obdélníka a prohlížeč zobrazí průsečík těchto dvou obdélníků. MediaBox se nastaví na rozměry papíru. Ty můžeme specifikovat buď pomocí příkazu `\special{papersize...}` nebo na příkazovém řádku parametrem `-t` nebo `-T` s odpovídající hodnotou. CropBox nastavíme následujícím makrem:

```
\providecommand\CropBox{0 0 595 842}
\def\PDF@CropAllPages{\pdf@write{[\CropBox] CropAllPages end}}
\AtBeginDocument{\PDF@CropAllPages}
```

Standardně zavedeme rozměry formátu A4. Použití `\providecommand` zajistí, že se nezruší definice, kterou vytvořil uživatel před zavedením tohoto stylu.

Někdy potřebujeme změnit CropBox aktuální stránky. K tomu použijeme makro:

```
\def\CropThisPage#1{\pdf@write{[#1] CropThisPage end}}
```

Na rozdíl od předchozího makra, které expanduje na příslušné rozměry, máme zde makro s jedním parametrem, jímž je čtveřice čísel.

Zde jsou pak příslušné PostScriptové povely:

```
% [ <llx> <lly> <urx> <ury> ] <type> CropMark
/CropMark {
 [ /CropBox 4 2 roll pdfmark
} def

% [ <llx> <lly> <urx> <ury> ] CropAllPages
/CropAllPages { /PAGES CropMark } def

% [ <llx> <lly> <urx> <ury> ] CropThisPage
/CropThisPage { /PAGE CropMark } def
```

Další makro definuje způsob, jak se má dokument otevřít. `\PDFdocview` musí expandovat na příslušné povely, které PDF vyžaduje. Standardní definice jsou následující:

```
\providecommand\PDFdocview{/PageMode /UseOutlines /View [/FitV 0]}
\def\PDF@docview{\pdf@write{[\PDFdocview\space PDFdocview end}}
\AtBeginDocument{\PDF@docview}
```

PostScriptová část má jediné makro:

```
/PDFdocview {/DOCVIEW pdfmark} def
```

Nakonec si ukážeme, jak zapíšeme základní informace o dokumentu. Datum vytvoření a modifikace musí mít tvar `(D:YYYYMMDDhhmmss)`[1], kde jednotlivé skupiny písmen zastupují rok, měsíc, den, hodiny, minuty, sekundy. Údaj nemusí

[1] Toto je zjednodušená podoba, úplnou syntaxi lze nalézt v literatuře uvedené na konci článku.

být úplný a může být ukončen po kterékoliv skupině. My budeme obvykle zapisovat jen datum bez časového údaje. Je zřejmé, že potřebujeme makro, které převede hodnotu čítače na dvě číslice:

```
\def\pdf@num#1{\ifnum #1<10 0\fi \the#1}
```

Nyní si nastavíme datum zpracování dokumentu. Makro má jeden parametr, a tím je jméno makra, do něhož se uloží datum ve formátu požadovaném v PDF. Definice je globální, což umožňuje zavolat toto makro uvnitř skupiny, kde jsme lokálně předefinovali `\day`, `\month` a `\year`. Automaticky si nadefinujeme i `\DateOfTeXing`.

```
\def\SetTeXingDate#1{\xdef#1{\the\year\pdf@num\month\pdf@num\day}}
\SetTeXingDate\DateOfTeXing
```

Dále si nadefinujeme i čas zpracování, přestože jej obvykle nepoužijeme:

```
\count@=\time   \divide\count@ by 60
\edef\@temp{\pdf@num\count@}%
\multiply\count@ by -60         \advance\count@ by \time
\edef\TimeOfTeXing{\@temp\pdf@num\count@}%
```

Mohli bychom modifikovat makro `\DateOfTeXing` následujícím způsobem:

```
\edef\DateOfTeXing{\DateOfTeXing\TimeOfTeXing}
```

Makro, obsahující datum modifikace, je totiž definováno takto:

```
\def\pdf@modt{(D:\DateOfTeXing)}
```

Informaci o dokumentu pak zapíšeme jedním z následujících maker. Makro `\DocInfo` má pevnou syntaxi. Jeho tři parametry jsou postupně autor, název dokumentu a datum publikování. Další informace nelze zadat. Naproti tomu makro `\DocInfoArray` umožňuje zadání libovolných informací, ale za cenu toho, že je uvádíme jako jediný parametr v hranatých závorkách, přičemž musíme dodržet syntaxi, kterou vyžaduje formát PDF. Automaticky se doplňuje pouze datum modifikace:

```
\newcommand*\DocInfo[3]{\@pdf@info@true
  \edef\pdf@title{(#1)}%
  \edef\pdf@auth{(#2)}
  \edef\pdf@crtdt{(D:#3)}%
  \edef\pdf@moddt{(D:\DateOfTeXing)}%
  \pdf@write{\pdf@title\space\pdf@auth\space\pdf@crtdt\space
             \pdf@moddt\space DocInfo}}
\newcommand*\DocInfoArray[1][]{\@pdf@info@true
  \edef\pdf@moddt{(D:\DateOfTeXing)}%
  \pdf@write{[ #1 \pdf@moddt\space DocInfoArray}}
```

Alespoň jedno z těchto maker musí být v dokumentu uvedeno. Při opomenutí chceme být varováni, ale nebudeme to považovat za velkou chybu. Použijeme tedy následující příkazy:

```
\newif\if@pdf@info@
\def\pdf@info@test{\if@pdf@info@ \else
```

```
  \typeout
   {*** ERROR: Document Information was not specified ***}\fi}
\AtBeginDocument{\pdf@info@test}
```

Zde máme odpovídající PostScriptová makra:

```
/DocInfo {
  /pdf_modif XD /pdf_creat XD /pdf_auth XD /pdf_tit XD
  [/Author pdf_auth
   /CreationDate pdf_creat
   /Producer (LaTeX)
   /Title pdf_tit
   /ModDate pdf_modif
  /DOCINFO pdfmark
} def

/DocInfoArray {
  /ModDate exch
  /DOCINFO pdfmark
} def
```

10. Podmíněné příkazy

V některých situacích mohou příkazy specifické pro PDF vadit. Nechceme tedy využívat vlastnosti balíku PDFHDR. Nemůžeme jej ale zakomentovat, protože bychom v dokumentu měli nedefinovaná makra. Zavedeme tedy parametry `create` (to bude default) a `nocreate`:

```
\newif\ifPDFcreate     \PDFcreatetrue
\DeclareOption{create}{\PDFcreatetrue}
\DeclareOption{nocreate}{\PDFcreatefalse}
\ProcessOptions
```

Pokud v příkazu `\usepackage` uvede uživatel parametr `nocreate`, znefunkčníme některá makra:

```
\ifPDFcreate \else
  \let\pdf@write\@gobble
  \let\pdf@@write\@gobble
  \let\pdf@info@test\@gobble % Should it really be here?
  \let\PDFlisting\relax
\fi
```

Makro `\ifPDFcreate` úmyslně ve svém názvu nemá zavináč. Je to proto, aby jej uživatel mohl snadno použít pro vlastní rozhodování, zda je generován kód pro PDF.

11. Omezení

Jak bylo ukázáno, činnost popisovaných maker je zcela závislá na PostScriptových příkazech, které vytváří DVI-ovladač. Současná verze tedy funguje pouze s programem DVIPS od Radical Eye Software. Konkrétně byl balík testován s verzí 5.58 pro OS/2, ale pravděpodobně je použitelný i s jinou verzí. Je však evidentní, že s jinými ovladači DVI→PS pracovat nebude. Protože všechny ovladače budou zřejmě převádět DVI-soubory podobným způsobem, bylo by jistě možné vytvořit odpovídající PostScriptová makra. Balík by pak mohl mít volbu ovladače, případně by se podle (ne)existence jistých maker mohl přepínat automaticky.

12. Dostupnost

Balík je stále ještě v alfa verzi a teprve nedávno v něm byly odhaleny závažné chyby. Po několika dalších testech a doplnění (anglické) dokumentace bude pravděpodobně na podzim roku 1999 vystaven na `http://www.icpf.cas.cz/wagner/ftp/tex.html` a nabídnut na CTAN.

13. Literatura

V seznamu literatury uvedeme sice primární zdroje informací, ale zejména se soustředíme na články ve Zpravodaji a v jiných dokumentech, které jsou pro čtenáře snadno dostupné, zejména v elektronické podobě.

13.1. Program pdfTEX

1. Hàn Thế Thành: *Alternativní výstup programu TEX – PDF*. Zpravodaj Československého sdružení uživatelů TEXu, **6** (2), 69–85 (1996).

13.2. Informace o formátu pdf

Knihy o formátu PDF byly vydány nakladatelstvím Addison-Wesley. Jejich elektronická forma (PDF) je dostupná též na stránce `http://partners.adobe.com/supportservice/devrelations/technotes.html`. Úmyslně neuvádíme přesná URL jednotlivých dokumentů, protože se v budoucnu mohou změnit. Zájemce si je na této stránce jistě najde.

2. *pdfmark Reference Manual*. Technical Note # 5150.
3. *Portable Document Format Reference Manual, Version 1.3*.

13.3. Makra pro tvorbu obsahu

Tato problematika je popsána v následujících dokumentech:

4. M. Goossens, F. Mittelbach, A. Samarin: *The LaTeX Companion*. Addison Wesley, Reading 1994, ISBN 0-201-54199-8.
5. Z. Wagner: *LaTeXová kuchařka/3*. Zpravodaj Československého sdružení uživatelů TeXu, **7** (3), 140–167 (1997).

Zdeněk Wagner
wagner@mbox.cesnet.cz

TUGboat 19(2), June 1998