Zpravodaj Československého sdružení uživatelů TeXu

Zdeněk Wagner

Záludnosti PostScriptu

Zpravodaj Československého sdružení uživatelů TeXu, Vol. 4 (1994), No. 2, 76–87

Persistent URL: http://dml.cz/dmlcz/149706

Terms of use:

© Československé sdružení uživatelů TeXu, 1994

Institute of Mathematics of the Czech Academy of Sciences provides access to digitized documents strictly for personal use. Each copy of any part of this document must contain these *Terms of use*.

This document has been digitized, optimized for electronic delivery and stamped with digital signature within the project *DML-CZ: The Czech Digital Mathematics Library* http://dml.cz

Všem členům CSTUGu, kteří se podíleli na tvorbě české distribuce emTEXu — CSTEX.
Jiřímu Zlatuškovi za jeho program ACCENTS.

Tomáš Mráz
xmraz@cslab.felk.cvut.cz

Záludnosti PostScriptu

Zdeněk Wagner

Každý, kdo se vážně zabývá typografií, dostane se dříve nebo později k PostScriptu. PostScript je jazyk pro popis stránek, který poskytuje další možnosti, jež lze TEXem a METAFONTem realizovat jen s obtížemi nebo vůbec. To samozřejmě vyžaduje určité znalosti. Lze se však spokojit s přístupem uživatele, který programem DVIPS vytvoří PostScriptový soubor a ten vytiskne na PostScriptové tiskárně nebo osvitové jednotce. Přesto je vhodné o PostScriptu něco vědět, aby se člověk vyhnul záludnostem, které v sobě skrývá jak PostScript tak další TEXovské pomocné programy.

Murphy by PostScript definoval jako nástroj, s jehož pomocí lze ve velmi krátkém okamžiku znehodnotit značné množství drahého materiálu. Experimentálně bylo totiž ověřeno, že rychlost PostScriptového zařízení je nepřímo úměrná smysluplnosti tištěného textu.

Smyslem tohoto článku je upozornit na hlavní záludnosti. Vše, co bude dále uvedeno, bylo objeveno většinou metodou pokusů a omylů a bohužel i finančně nákladných chyb. Jiná, zřejmě i lepší řešení jsou jistě možná...

Moderní tiskárny mají PostScript level 2, zatímco starší zařízení znají pouze level 1. Nebudeme se zde příliš zabývat rozdíly. Téměř vše, co je v tomto článku uvedeno, platí pro level 1. Odlišnosti budou zdůrazněny pouze tam, kde mohou působit problémy.

PostScript je podobně jako TEX interpretační jazyk. Společným znakem je možnost definice uživatelských maker. Oba jazyky však mají řadu odlišností. Hlavní rozdíl spočívá v tom, že PostScript neumí lámat řádky a odstavce. V PostScriptu je nutno popsat již zlomenou stránku.

TEX čte vstupní proud (input stream[1]), expanduje makra a výsledek expanze vrací zpět do vstupního proudu. Pokud nějaké makro vyžaduje parametry, vezme je TEX ze vstupního proudu z textu, který následuje za příslušným makrem. V PostScriptu probíhá expanze trochu odlišně. Parametry si expandované makro bere ze zásobníku a případný výsledek opět vkládá do zásobníku. Není to tedy expanze ve stejném smyslu, jak ji známe z TEXu. Pokud chceme makrem `moveto` nastavit aktuální pozici na souřadnice x, y, musíme je do zásobníku vložit předem. Celý příkaz pak bude vypadat:

x y `moveto`

PostScript má čtyři zásobníky, z nichž nás budou prozatím zajímat dva. Tím prvním, který jsme již použili, je zásobník operandů (operand stack). Jak název napovídá, předávají se v něm operandy maker (v PostScriptu se makrům obvykle říká procedury) a výsledky jejich expanze. Dále bude užitečný zásobník slovníků (dictionary stack), jehož význam vysvitne později.

Chceme-li v TEXu definovat nové makro, použijeme `\def`. Přitom nás nezajímá, kam si TEX tuto definici uloží. Chceme-li pouze dočasně předefinovat již existující makro, je vhodné otevřít skupinu (group) a definici provést lokálně. Při interpretaci hledá TEX definici makra počínaje pravě otevřenou skupinou a pokračuje do vnějších skupin, dokud definici nenajde. Po uzavření skupiny se zapomenou lokální definice a obnoví se definice původní. PostScript přistupuje k definicím odlišně. Všechny definice se ukládají do slovníků. Slovníky, se kterými se právě pracuje, jsou uloženy (přesněji řečeno jejich adresy) v zásobníku slovníků. PostScriptový interpret prohledává slovníky v zásobníku odshora dolů. V okamžiku startu interpretu obsahuje zásobník dva slovníky: `systemdict`, v němž jsou všechny systémové příkazy, a `userdict` určený pro uživatelské definice. Tyto dva slovníky nelze ze zásobníku odstranit. Chceme-li otevřít nový slovník, řekněme `MyDict`, vložíme jej napřed do zásobníku operandů a poté použijeme příkaz `begin`. Nové definice se potom budou ukládat do slovníku `MyDict`. Příkazem `end` se slovník ze zásobníku slovníků odstraní. Z toho vidíme dvě odlišnosti:

[1] Česká terminologie mi občas činí potíže. Proto budu v závorkách uvádět anglické termíny.

1. Slovník může být v zásobníku slovníků, a tedy v prohledávacím řetězci několikrát.
2. Po odstranění slovníku ze zásobníku slovníků nejsou definice ztraceny, jsou pouze neaktivní.

V TEXu jsme se nestarali o to, zda máme v otevřené skupině dostatek paměti pro definice maker. To si v PostScriptu nemůžeme dovolit. Když vytváříme slovník příkazem `dict`, musíme jako parametr zadat jeho velikost. Zde je první důležitý rozdíl mezi level 1 a level 2! Je-li slovník přeplněn, level 2 si jej zvětší, zatímco level 1 ohlásí chybu `-dictfull-` a ukončí zpracování. Softwarový interpret PostScriptu, program Ghostscript, má sice implementován PostScript level 1, ale vzhledem k rozšiřování přeplněného slovníku se chová jako level 2.

Tento rozdíl mi způsobil trochu trápení. Soubor, který jsem připravil, se vytiskl bez problémů na tiskárně Hewlett-Packard 4M a dokonce byl dobře zobrazen Ghostscriptem. Na osvitové jednotce se však nevytisklo vůbec nic. Příčinou, jak již možná tušíte, byl přeplněný slovník. Nebylo to ovšem tak jednoduché.

Slovník může obsahovat leccos, například jiné slovníky. Definice fontu je také velmi složitý slovník, v němž kromě řady jiných objektů jsou další slovníky. Jedním z nich je `CharStrings`[2] obsahující vektorové popisy jednotlivých znaků. Ve fontu, který jsem ve zmíněném dokumentu použil, chybělo původně „i bez tečky“ (ı). Přestože nejsem expert na PostScript, nebylo složité převést definici fontu do „lidsky čitelné podoby“, zkopírovat definici písmene „i“, vyhodit příkazy pro nakreslení tečky a výsledek zkompilovat zpět do kompaktního tvaru. Zapomněl jsem však na jednu věc. Do slovníku `CharStrings` jsem přidal další písmeno, ale nezvětšil jsem velikost slovníku. Proto mi osvitová jednotka nic nevytiskla.

Chcete-li vytvořit soubor, který by se měl vytisknout na libovolném zařízení, je vhodné používat pouze příkazy level 1. Program DVIPS to dělá vždy (tedy skoro, viz dále), v jiných programech lze interpretaci zvolit z menu. Je však nutné uvědomit si, že Ghostscript nemůže být měřítkem. V konkrétním případě byl Ghostscript schopen správně zobrazit soubor, který osvitová jednotka nevytiskla. Na druhé straně mi Ghostscript mnohokrát nahlásil nesmyslné chyby v souborech, které osvitovou jednotkou prošly zcela bez problémů.

Tím ovšem PostScriptové patálie nekončí. Nyní si ukážeme příklady, které se mohou často hodit. Budeme již přitom programovat v Post-

[2] Podrobnější popis přesahuje rámec tohoto článku.

Scriptu. Podobně jako v TEXu nejsou mezery mezi objekty významné (tj. stačí jedna). Můžeme psát dlouhé řádky (pokud se nemýlím, maximální povolená délka je 512 znaků). Tak dlouhé řádky jsou však nepřehledné. V příkladech budeme řádky číslovat, abychom se na ně mohli odvolávat, do skutečných PostScriptových programů se však čísla řádků nepíší.

Typickým příkazem, který budeme potřebovat, je zrcadlení stránky. Než přistoupíme k jeho definici, musíme si něco říci o tom, jak vypadá popis stránky generovaný programem DVIPS. Prvním příkazem stránky je `bop`, což je makro definované v souboru `texc.pro`. Za ním následuje vlastní popis stránky a na jejím konci je makro `eop`. Makro `bop` se podívá, zda je ve slovníku `userdict` definován `bop-hook`. V kladném případě je `bop-hook` proveden. Zde je tedy vhodné místo, kam naprogramovat zrcadlení. Lze to provést následujícím programem, který nejprve napíšeme a pak jej vysvětlíme.

```
1 userdict begin
2 /bop-hook
3 {
4   -1 1 scale
5   -595 0 translate
6 }
7 def
8 end
```

Řekli jsme si, že `bop-hook` musí být ve slovníku `userdict`. Nemáme jistotu, který slovník bude na vrcholu zásobníku slovníků v okamžiku zpracovávání naší definice. Proto na řádku 1 otevřeme `userdict`. Řádek 2 obsahuje tzv. „name-literal“, což neumím říci česky. Napíšeme-li `bop-hook`, bude PostScriptový interpret hledat definici a odpovídající příkazy provede. Pokud však zapíšeme jméno s úvodním lomítkem, vloží se do zásobníku jméno objektu, nikoliv objekt. Zápis `/bop-hook` ve spojení s dalšími operacemi říká, že definujeme nový příkaz `bop-hook`.

Definici lze vložit do složených závorek (viz řádky 3 a 6). Pak se do definice uloží vše přesně v tom tvaru, jak je to napsáno, tj. bez expanze. Je to tudíž obdoba TEXového `\def`. Pokud bychom závorky vynechali, vložil by se do definice až výsledek expanze podobně jako v TEXovém `\edef`.

Abychom pochopili další řádky, musíme si opět vysvětlit některé rysy PostScriptu. Bod s nulovými souřadnicemi je totiž v levém dolním rohu

papíru. Zatímco x-ová souřadnice roste směrem doprava, y-ová souřadnice roste ve směru zdola nahoru. V TEXu roste y-ová souřadnice při pohybu shora dolů. Vzdálenost se měří v PostScriptových jednotkách (PostScript unit), jejichž velikost se shoduje s TEXovým „bp“.

Řádek 4 představuje první transformaci souřadnic. Uživatelské souřadnice zde převádíme na souřadnice tiskárny tak, že x vynásobíme hodnotou -1 a y hodnotou 1. Stránka se tím zrcadlí podle levé hrany papíru. Tím ovšem text dostaneme mimo papír a tiskli bychom prázdné stránky, což by byla levnější varianta chyby (ne však na osvitové jednotce, protože i prázdný film se platí). Musíme tedy posunout počátek souřadnic do pravého dolního rohu papíru. Máme-li formát A4, je šířka stránky 210 mm, což je přibližně 595 bp. Protože jsme ale otočili směr osy x, je na řádku 5 záporná hodnota.

Řádek 7 ukončí definici a na řádku 8 odstraníme slovník `userdict` ze zásobníku slovníků, abychom vše vrátili do původního stavu.

Tuto definici lze urychlit. Operátor `def` totiž říká, že PostScriptové příkazy použité v definici se uloží tak, jak jsou, a interpretují se až v okamžiku provádění makra. Pokud bychom na řádku 7 použili `bind def`, vložil by se do definice aktuální význam použitých příkazů `scale` a `translate`. Při provádění příkazu `bop-hook` by se již nehledaly odpovídající definice ve slovnících. To má ještě další výhodu. Příkazy `scale` a `translate` by mohly být později předefinovány, aniž by tím byla narušena funkce makra `bop-hook`!

Zrcadlení stránky A4 jsme tedy vymysleli. Zbývá ještě nalézt vhodné místo, kam tuto definici napsat. V návodu k programu DVIPS se dočteme o příkazu `\special`. Ten umožňuje zapsání PostScriptových příkazů do DVI souborů, které pak DVIPS zkopíruje do PostScriptového souboru. Tímto způsobem lze zapsat i příkaz level 2, což, jak bylo uvedeno dříve, může vést ke katastrofě.

Program pro zrcadlení stránek by měl být na začátku PostScriptového souboru. Proto musí být prvním znakem příkazu `\special` vykřičník. Zcela automaticky napíšeme:

```
 9 \special{!userdict begin /bop-hook {-1 1 scale
10   -595 0 translate} bind def end}
```

Ve většině případů to bude fungovat. Bohužel ne vždycky…

Představte si následující situaci. Soubor nejprve ladíme na vlastní PostScriptové tiskárně nebo pomocí Ghostscriptu. Proto chceme tisknout vše bez transformací. Po odladění přidáme příkaz z řádku 9,

samozřejmě na začátek souboru. Při ladění jsme tiskli celý dokument. Protože první stránka je vakát (nebo obrázek, který bude vytvořen jinou technikou) a film je drahý, použijeme např. program DVI2DVI, jímž z DVI souboru všechny vakáty a stránky pro obrázky vyřadíme a teprve potom vyrobíme PostScriptový soubor programem DVIPS. Výsledek však nebude zrcadlově obrácen. Příkaz `\special` byl totiž na první stránce, kterou jsme vyhodili.

Příkaz `\special` z řádku 9 má ještě jednu nevýhodu. Máme-li hotový dokument a chceme jej pouze zrcadlově obrátit, musíme při tomto přístupu znovu zpracovat dokument TEXem, což vyžaduje určitý čas. Uvedeme tedy dvě řešení, která odstraní oba problémy.

První řešení je použitelné pouze v případě, že tiskneme na vlastní PostScriptové tiskárně, která není připojena v síti. Pak totiž můžeme poslat do tiskárny kratičký soubor obsahující příkazy 1–8 a ihned potom PostScriptový dokument. Je nutno zdůraznit, že příslušný dokument musíme poslat do tiskárny *ihned*. PostScriptová zařízení jsou totiž vybavena dvojsečnou zbraní, kterou je „job timeout“. Tiskárna pracuje, dokud do ní přicházejí data. Je-li tok dat na určitou dobu přerušen, interpret předpokládá, že došlo k chybě. Aby nebyly poškozeny následující soubory, tiskárna se resetuje. Při používání tiskárny na síti je to vlastnost užitečná. Pokud ale používáte trik popsaný v tomto odstavci, musíte být dostatečně rychlí. Nejlepší je, když oba soubory napíšete ve správném pořadí na jeden příkazový řádek.

Zmíněná vlastnost skýtá další záludnost. Může totiž ukrýt chybu, takže na ni přijdete až později. Zpočátku jsem pro zrcadlový tisk používal příkaz `\special`, který jsem vyhazoval programem DVI2DVI, aniž bych si to uvědomil. Tiskl jsem vždy více souborů současně, přičemž, shodou okolností, v prvním souboru byl příkaz `\special` přítomen. Všechny soubory tedy tiskárna považovala za jeden „job“ a zrcadlení fungovalo. Až jednou jsem v tisku udělal přestávku...

Nyní je ale čas pro druhé řešení. První řešení totiž nemůžeme použít, pokud chceme tisknout na síťové tiskárně, nebo v případě, že soubor chceme odnést na osvitovou jednotku. Je sice možné libovolným textovým editorem připsat příkazy pro zrcadlení přímo do PostScriptového souboru, ale DVIPS nabízí ještě efektnější metodu. Vytvoříme soubor pojmenovaný např. `mirror.hdr` obsahující příkazy z řádků 1–8 a DVIPS pak vyvoláme s parametrem `-h mirror.hdr`.

Život však není šedivý. V praxi budeme tisknout i na jiné formáty než A4. Můžeme sice pro každý formát napsat jiná makra, ale kdo se

s tím má dřít? Raději využijeme toho, že DVIPS vezme rozměry z příkazu `\special{papersize=...}` a vloží šířku papíru do konstanty `hsize` a výšku do `vsize`. Program pro zrcadlení pak můžeme napsat obecněji:

```
11 userdict begin
12 /bop-hook
13 {
14   -1 1 scale
15   hsize neg 0 translate
16 }
17 def
18 end
```

Změna je na řádku 15. Místo konkrétního čísla zde použijeme záporně vzatou (`neg`) hodnotu `hsize`. Na řádku 17 nyní nesmíme použít `bind def`, protože v okamžiku definice `bop-hook` není hodnota `hsize` ještě známa.

Uvedli jsme, že osa y má v PostScriptu opačný směr než v TEXu. Proto musíme v příkazu `\special{papersize=...}` uvést skutečné rozměry papíru. Jinak nebude text správně umístěn. Výše uvedené příkazy tedy nebudou fungovat v případě, kdy tiskneme v orientaci „landscape“. Rotace o 90° totiž způsobí, že se prohodí výška a šířka. Místo řádku 15 bychom tedy měli psát

```
19   vsize neg 0 translate
```

Nechceme však mít dva různé soubory pro zrcadlení, protože pak bychom v tom měli zmatek. Orientace je programem DVIPS vložena do konstanty `isls`, což zřejmě značí „is landscape“. Zcela obecně pak napíšeme:

```
20 userdict begin
21 /bop-hook
22 {
23   -1 1 scale
24   isls
25     { vsize }
26     { hsize }
27   ifelse
28   neg 0 translate
29 }
```

```
30 def
31 end
```

Řádky 24–27 tvoří podmíněný příkaz. Je-li hodnota `isls` pravdivá, provede se příkaz v prvním páru složených závorek, tj. na řádku 25 se do zásobníku operandů vloží hodnota `vsize`. V opačném případě se na řádku 26 vloží do zásobníku operandů hodnota `hsize`. Zbytek je již stejný jako dříve.

Nyní přistoupíme k poslednímu, komplikovanějšímu příkladu. Zde budeme demonstrovat, co vše lze s PostScriptem provést. Současně však důrazně varujeme před používáním takových praktik. Přesto však některé obraty mohou být užitečné.

Často chceme tisknout na papír formátu A4 dvě stránky A5 vedle sebe. Lze to provést na úrovni DVI souboru např. programem DVI2DVI. Můžeme to ovšem naprogramovat i v PostScriptu. Použijeme k tomu následující příkazy, které si opět uložíme do souboru `twoup.hdr`, abychom pak mohli volat DVIPS s parametrem `-h twoup.hdr`.

```
32 userdict begin
33 userdict /ZWM known {} { /ZWM false def } ifelse
34 userdict /ZW known {} { /ZW 2 def } ifelse
35 /start-hook { @landscape } def
36 /bop-hook {
37   gsave
38   ZWM { -1 1 scale vsize neg 0 translate } if
39   dup ZW mod
40   dup hsize ZW div neg mul 0 exch translate
41   1 add ZW eq
42   { userdict begin
43       /eop-hook { grestore } def
44       /flushpage {} def
45     end
46   }
47   { userdict begin
48       /eop-hook { end grestore } def
49       /flushpage { showpage } bind def
50     end
51     ZWdict begin
52   }
53   ifelse
```

```
54 } def
55 /flushpage {} def
56 /end-hook { flushpage } def
57 end
58 /ZWdict 5 dict def
59 ZWdict begin /showpage {} def end
```

Na řádku 32 opět otevřeme slovník `userdict`, kam budeme vkládat část definic. Řádky 33 a 34 jsou zde kvůli větší obecnosti. Operátorem `known` testujeme, zda je daný symbol v zadaném slovníku definován. Pokud se příslušný symbol najde, nebudeme dělat nic, proto jsou na obou řádcích první složené závorky prázdné. V druhých závorkách máme standardní hodnotu. Všimněte si, že v těchto definicích není hodnota konstanty v závorkách. Konstantu `ZWM` použijeme k rozhodování, zda se má stránka zrcadlově otočit, `ZW` definuje počet stránek, které se mají tisknout vedle sebe.

Dále potřebujeme zajistit, aby se text vždy tiskl v orientaci „landscape“. DVIPS to zajišťuje příkazem `@landscape`. Protože nemáme jistotu, že uživatel nastaví takové parametry, aby DVIPS tento příkaz vyslal, zařídíme to sami. Ještě před první stránku DVIPS umístí příkaz `@start`. Ten potom zavolá z `userdict` makro `start-hook`. Řádek 35 tedy zajistí tisk v orientaci „landscape“.

Řádky 36 až 54 přinášejí jádro řešení. Před vysvětlením příkazu z řádku 37 však musíme na okamžik odbočit. Na začátku článku jsme psali o zásobnících. Zde přidáme další, a to zásobník grafických stavů. Příkaz `gsave` totiž uschová grafický stav do zásobníku grafických stavů, příkazem `grestore` původní stav obnovíme. Mezi `gsave` a `grestore` obvykle uzavíráme všechny dočasné změny grafického stavu. Může to být např. změna transformace souřadnic. To je i náš případ.

Řádek 38 provede zrcadlení. Zrcadlíme podél svislé osy, abychom nenarušili činnost maker skládajících stránky.

Nyní musíme naprogramovat makra pro skládání stránek. K tomu potřebujeme znát číslo stránky, kterou právě tiskneme. Naštěstí `bop-hook` dostává v zásobníku dva parametry: číslo stránky dosazené TEXem a pořadové číslo stránky (číslované od nuly). Parametry můžeme použít, ale musíme je ponechat v zásobníku. Nám bude stačit pouze pořadové číslo, které máme právě na vrcholu zásobníku. Na řádku 39 jej tedy nejprve zkopírujeme instrukcí `dup` a potom vypočteme zbytek po dělení hodnotou `ZW`.

Po otočení o 90° do orientace „landscape“ bude pravý dolní roh tištěné stránky v levém horním rohu papíru. Pro nultou stránku je to správně, ostatní musíme posunout dolů. Výška papíru, což v orientaci „landscape“ je nyní šířka, je uložena v konstantě `hsize`. Musíme tedy stránku posunout o `hsize/ZW` vynásobenou zbytkem po dělení čísla stránky hodnotou `ZW`. To se právě zařídí na řádku 40. Zbytek po dělení čísla stránky hodnotou `ZW` budeme ještě potřebovat, proto si nejprve uděláme jeho kopii. Potom vypočteme hodnotu svislého posunu a přidáme do zásobníku nulu pro vodorovný posun. Protože parametry jsou nyní v zásobníku v opačném pořadí, než vyžaduje `translate`, musíme je prohodit instrukcí `exch`.

Místo řádku 40 jsme mohli použít příkazy:

```
60 dup /zbytek exch def
61 0
62 hsize ZW div
63 zbytek neg mul
64 translate
```

Příkazy jsme pro větší přehlednost ještě více rozčlenili. Neobvykle vypadá řádek 60. Jeho smyslem je uložení objektu na vrcholu zásobníku do pojmenované konstanty `zbytek`. Nejprve totiž vytvoříme kopii objektu na vrcholu zásobníku instrukcí `dup`. Poté vložíme na vrchol zásobníku „name-literal“ `/zbytek`. Instrukcí `exch` pořadí nejvyšších dvou objektů zaměníme, takže, označíme-li výsledek výpočtu z řádku 40 symbolicky $\langle zbytek \rangle$, mají instrukce

```
dup /zbytek exch
```

stejnou funkci jako

`/zbytek` $\langle zbytek \rangle$

Příkaz `def` tedy dostane vše tak, jak je vyžadováno.

Ostatní řádky jsou již obvyklé. Na řádku 61 vložíme do zásobníku nulovou hodnotu vodorovného posunu. Na řádku 62 vydělíme `hsize` hodnotou `ZW` a podíl vynásobíme na řádku 63 záporně vzatou hodnotou uschovaného zbytku. Na řádku 64 pak zavoláme transformaci `translate`.

Řádky 41–53 tvoří podmíněný příkaz. V něm nejprve zbytek po dělení čísla stránky hodnotou `ZW` zvětšíme o jednotku a výsledek porovnáme se

`ZW`. V případě rovnosti jsme skoro hotovi. Provedou se příkazy na řádcích 42–46. Zde se nadefinuje `eop-hook` tak, aby se instrukcí `grestore` obnovil grafický stav a `flushpage` se nadefinuje jako prázdná operace. Protože nevíme, jaký slovník bude na vrcholu zásobníku slovníků v okamžiku vykonávání `bop-hook`, otevřeme si explicitně `userdict`.

V případě nerovnosti se provedou příkazy na řádcích 47–52. Zde se definuje `eop-hook`, kde nejprve odstraníme aktuální slovník z vrcholu zásobníku slovníků a obnovíme grafický stav. Makro `flushpage` bude ekvivalentní standardnímu PostScriptovému příkazu `showpage`, který tiskne stránku. Na řádku 51 pak vložíme do zásobníku slovníků vlastní slovník `ZWdict`. Právě ten odstraňuje námi definovaný `eop-hook`.

Řádek 55 definuje počáteční význam procedury `flushpage`.

Příkaz na řádku 56 je velmi důležitý. Nemáme totiž jistotu, že počet stránek dokumentu bude celočíselným násobkem `ZW`. Využijeme tedy toho, že na konci dokumentu se volá `end-hook` a nadefinujeme jej tak, aby se volalo naše makro `flushpage`. To jsme během zpracování souboru stále předefinovávali podle toho, zda byla již složena celá stránka.

Na řádku 58 vytvoříme operátorem `dict` slovník `ZWdict`, do kterého se vejde 5 definic. Na následujícím řádku do něj vložíme definici `showpage`, která nebude dělat nic. Na řádku 51 jsme vložili slovník s touto definicí na vrchol slovníku zásobníků. Protože na konci stránky se vždy volá `showpage`, máme tím zajištěno, že se nevytisknou neúplné stránky.

Pokud se po odladění dokumentu rozhodneme, že jej chceme zrcadlově převrátit, stačí použít navíc soubor s příkazem:

```
65 userdict /ZWM true def end
```

Podobně tisk tří stran vedle sebe způsobíme souborem s příkazem:

```
66 userdict /ZW 3 def end
```

Zbývá ještě vysvětlení, proč jsme před používáním tohoto příkladu varovali. Souvisí to s požadavky na členění PostScriptových souborů. Podle standardu Adobe má mít PostScriptový soubor tzv. prolog a skript. Prolog obsahuje definice vyžadované v celém dokumentu, download nerezidentních fontů, nastavení specifických konstant a podobně. Skript obsahuje popis jednotlivých stránek a na jeho konci je „trailer“, který vše uklidí. Stránky musí být samostatné. Žádná stránka nesmí ovlivňovat stránky následující. Právě to jsme v našem příkladu porušili. Existují totiž programy, které dokážou z PostScriptového souboru vybrat některé stránky a poslat je do PostScriptového zařízení. Kdyby stránky

byly na sobě závislé, nedopadlo by to dobře. Příklad, který jsme zde uvedli, pamatuje skoro na vše. Vybereme-li souvislý úsek, vytiskne se vše dobře. Program, který stránky vybírá, musí totiž vždy zkopírovat prolog i trailer. Tím se vše, co jsme napáchali, zase napraví. Pokud ale vybereme několik náhodných stránek, nebude podmíněný příkaz na řádcích 41–53 fungovat správně a výsledkem bude zmatek.

V tomto příspěvku nebylo možné podat vysvětlení PostScriptu. Není zde dostatek prostoru. Jeho cílem bylo pouze upozornění na některé nové možnosti a záludnosti. Pro toho, kdo má zájem dozvědět se o PostScriptu více, lze doporučit např. knížku Davida A. Holzganga: Understanding PostScript (SYBEX, San Francisco), která poslouží jako vhodná základní učebnice pro začátečníky.

Zdeněk Wagner
`<wagner@icpf.cas.cz>`

Kreslení obrázků v TEXu pomocí `mujmfpic`

Jaromír Kuben

V několika posledních číslech TEX bulletinu se objevila řada příspěvků týkajících se otázky, jak vyrobit v TEXovském dokumentu obrázky nejrůznějšího druhu. Chtěl bych se se zájemci o tuto problematiku podělit o své zkušenosti s balíkem `mfpic`, který využívá pro tyto účely program METAFONT, aniž je třeba znát jeho jazyk (což skalní zastánci tohoto skvělého programu odsuzují).

S tímto balíkem ve verzi 0.2 jsem se poprvé setkal před prázdninami 1993. Vzorové obrázky vypadaly pěkně, dobře dopadly i první experimenty, ale současně se objevily i jisté potíže. Protože jsem tehdy měl za sebou určité, sice nevelké ale úspěšné, pokusy s METAFONTem (namaloval jsem obrázky do dvojích skript), pokusil jsem se problémy odstranit. Stálo mne to značné úsilí, ale podařilo se. Přitom mne napadlo, že když už jsem věnoval tolik času na proniknutí do zdrojového kódu, nebylo by od věci rozšířit škálu obrázků, které lze pomocí `mfpic` namalovat. Zejména jsem postrádal křivky dané parametricky nebo v polárních sou-