Putování písmene ř z klávesy na papír

Petr Olšák

V tomto článku podrobně rozeberu, co se odehraje v počítači mezi zmáčknutím písmene ř na klávesnici a vytištěním písmene ř na papíře. Ono písmeno je v jednotlivých fázích zpracování interpretováno jako číselný kód. Ukážeme, že se zdaleka nemusí jednat o kód jediný. V průběhu zpracování může písmeno ř (a ostatní písmena a znaky, které používáme) absolvovat až sedm různých kódování, přičemž v každém z nich může (ale nemusí) být naše písmeno reprezentováno jiným kódem.

Článek by měl dokumentovat velkou flexibilitu přístupu k různým kódováním při zpracování dokumentu TEXem. Uvidíme, že možností je mnoho a těžko bychom hledali u jiného softwaru obdobu.

Putování písmene ř rozdělíme do několika fází. V každé fázi může být tento znak reprezentován jiným kódem. Jednotlivé fáze můžeme zhruba načrtnout do následujícího schématu:

$\boxed{\text{klávesa ř}}$	
K-kódování	… hardwarové kódování klávesnice
↓	… ovladač klávesnice
A-kódování	… kódování aplikace textového editoru
↓	… uložení dokumentu na disk
D-kódování	… kódování dokumentu na disku
↓	… input preprocesor TEXu
I-kódování	… kódování dokumentu na vstupní straně maker
↓	… expanze maker
T-kódování	… kódování na úrovni hlavního procesoru a v dvi
↓	… expanze dvi podle virtuálních fontů
F-kódování	… kódování fontu, nebo jeho Encoding vektoru
↓	… Encoding vektor
P-kódování	… vnitřní kódování PostScriptového fontu
$\boxed{\text{tištěné ř}}$	

Uvedeme seznam nejběžnějších kódování, která se na úrovni jednotlivých fází zpracování obvykle v českém a slovenském jazyce používají:

K-kódování	… scan kódy klávesnic
A,D,I-kódování	… ISO-8859-2, Kamenických, PC-Latin2=CP852, CP1250, …
I,T,F-kódování	… kódování CS-fontů, T1=Cork, …
F,P-kódování	… Adobe StandardEncoding, …

V dalším textu se jednotlivými fázemi zpracování dokumentu budeme věnovat podrobněji.

K-kódování ⟶ A-kódování

Stisknutí klávesy se reprezentuje číselným kódem, který je odeslán do počítače. Například na klávesnicích PC se jedná o tzv. „scan kódy", kde klávesy mají vesměs čísla podle svého rozmístění. Klávesa Esc má číslo jedna, klávesa A číslo 30, S 31, D 32 atd. V počítači pracuje ovladač klávesnice, který jednotlivou posloupnost stisků kláves interpretuje a běžící aplikaci již vrací aplikační kód. Je-li onou aplikací textový editor (což je při přípravě dokumentu pro TEX obvyklé), pak aplikace zobrazí znak na obrazovce a uloží si jej do paměti. K tomu zobrazení na monitoru potřebuje použít font, který musí mít rovněž A-kódování, jinak bychom byli zmateni.

Přehled o tom, co kdy bylo zmáčknuto a uvolněno a co to v dané situaci znamená, musí udržovat ovladač klávesnice. Například stisk levé klávesy Shift vyvolá scan kód 42. Pokud uživatel klávesu drží, pak ovladač musí následující klávesy interpretovat odlišným způsobem. O puštění klávesy Shift se ovladač dozví podle scan kódu 42+128. Ovladač klávesnice obvykle pracuje v několika režimech (český/standardní). Naše klávesa ř samostatně neexistuje. Písmeno ř může být v českém režimu vyvoláno buď použitím smluvené klávesy (obvykle klávesa 5/%) nebo stiskem a uvolněním smluvené „mrtvé klávesy“ (obvykle klávesa +/=) a následným stiskem klávesy r. Vše může vypadat i jinak. Záleží na tom, jak je ovladač klávesnice naprogramován.

V UNIXu je scan kód z klávesnice zpracován rovněž ovladačem klávesnice (v jádru operačního systému) na tzv. key kód. Pokud běží X Window System, pak je key kód obvykle zpracován X aplikací podle globální mapovací tabulky, společné všem aplikacím. Tuto tabulku je možné měnit programem `xmodmap`. Aplikace může také převzít inteligenci ovladače klávesnice do svých rukou a starat se o přepínání režimů klávesnice a transformaci znaků. Mám na mysli třeba Emacs při použití maker z balíku `emacs-czech` [EC]. Kód znaku se tedy může měnit na třech místech: v jádru systému, na úrovni globální mapovací tabulky a uvnitř konkrétní aplikace. Ve všech případech existují konfigurační možnosti, jak do konverzního algoritmu zasáhnout.

Font, který používá aplikace k zobrazení právě stisknutého znaku, je obvykle fontem, který je nějakým způsobem instalován v operačním systému. A-kódování tedy volíme shodně s kódováním fontu v operačním systému. U jiného druhu textového zpracování tím veškerá starost o kódování končí (např. MS Word a jiné), protože pro tisk dokumentu používá aplikace rovněž fonty, které nabízí operační systém.

My uživatelé TEXu víme, že mezi fontem použitým v operačním systému a fontem určeným pro kvalitní typografické zpracování sazby může být propastný rozdíl. Font operačního systému volíme takový, aby hezky vypadal a byl dobře čitelný na obrazovce, zatímco pro tisk volíme font, který nejlépe odpovídá typografickému záměru, kvalitě papíru, tiskové technologii apod. Proto se pusťme do dalších fází zpracování dokumentu.

A-kódování $\longrightarrow$ D-kódování

Jakmile požádáme aplikaci (textový editor), aby uložila dokument na disk, může aplikace při této činnosti spustit nějaký filtr, který může změnit kódování dokumentu. Například pro MS Windows existují (prý) editory, které vytvářejí a zobrazují dokument v CP1250, ale po uložení jej máme třeba v PC-Latin2. Vstupní bránu TEXu pak musíme připravit na to, že bude číst kódování v PC-Latin2, ačkoliv pracujeme s operačním systémem, který používá CP1250.

Obvykle při ukládání dokumentu kódování neměníme. V této fázi zpracování budeme měnit kódování asi jen ve zcela ojedinělých a opodstatněných případech.

Na druhé straně je běžnou praxí převádět dokumenty před zpracováním TEXem z jednoho kódování do druhého. Označme to jako D_1-kódování $\longrightarrow$ D_2-kódování. Ve většině TEXovských instalací jsou konverzní utilitky k dispozici. Konverzi provádíme například tehdy, když obdržíme dokument v jiném kódování, než používáme v našem systému.

D-kódování $\longrightarrow$ I-kódování

Tato konverze již probíhá uvnitř TEXu na úrovni jeho input procesoru. Ve zdrojovém kódu TEXu jsou implementovány vektory `xord` a `xchr`, které se starají o odstínění kódování použitého v hostitelském operačním systému od vnitřního kódování TEXu. Toto vnitřní kódování se předpokládá jednotné a nezávislé na kódování, které je podporováno hostitelským operačním systémem. To byl původní záměr autora TEXu. Knuth v sedmibitových dobách uvažoval existenci systémů s kódováním EBDIC, zatímco on volí v TEXu kódování ASCII. Na systémy s ASCII jsou tedy vektory `xord` a `xchr` nastaveny tak, že se chovají jako identické zobrazení, zatímco na těch druhých systémech dnes už vymřelých dinosaurů byly uvedené vektory příslušným způsobem pozměněny.

Je třeba poznamenat, že vektory `xord` a `xchr` se starají o oboustranné konverze, tj. nejen „dovnitř" TEXu, ale též při výstupu na terminál a do logu (nikoli do dvi). Proto TEX nabízí všechny své textové výstupy v kódování podle hostitelského operačního systému, ačkoli si sám interně může pracovat v jiném kódování.

Bohužel, nastavení překódovacích vektorů `xord` a `xchr` je možné jen před kompilací TEXu ze zdrojového souboru `web`. Za provozu TEXu lze měnit tyto vektory jen na některých implementacích. Tam, kde změna kódování za provozu není možná, je obvykle pevně nastaveno identické zobrazení.

Implementace TEXu zvaná emTEX (DOS, OS/2) je vybavena možností uvedené vektory měnit velmi volně prostřednictvím tzv. `tcp` tabulek. Tyto tabulky lze nejprve editovat, pak převést do binární podoby a předložit je TEXu v době inicializace formátu. Většinou se používají tabulky, které mají na vstupní straně kódování podle Kamenických, PC-Latin2 nebo CP1250 a na výstupní straně kódování CS-fontů nebo T1 nebo expanze na TEXovské sekvence. Poslední možnost způsobí, že naše `ř` vstupuje do TEXu nakonec jako `\v r`. Tuto třetí možnost nedoporučuji, protože znemožňuje výskyt písmene ř ve verbatim prostředí.

UNIXové instalace TEXu, které vycházejí z implementace `web2c` a jsou ozáplatovány kódem podle `ftp://ftp.muni.cz/pub/tex/local/cstug/skarvada` mají rovněž možnost měnit uvedené vektory `xord` a `xchr` za provozu TEXu, ovšem kódovací tabulku nemůžeme editovat, pouze si můžeme vybrat jednu z několika málo možností. Mezi zajímavou možnost patří zřejmě konverze ISO-8859-2 $\longrightarrow$ T1.

Existuje i jiná úprava TEXu, kterou jsem si udělal sám, a která umožňuje přímý přístup k vektorům `xord` a `xchr` za běhu TEXu [ENCTEX].

I-kódování ⟶ T-kódování

Než se naše písmeno `ř` propracuje z input procesoru do hlavního procesoru TEXu, může s ním expand procesor pořádně zatočit. Této problematice se budeme nyní věnovat podrobněji.

V nejběžnějším případě, tj. když není vstupní písmeno `ř` aktivní, se nestane nic. Takže I-kódování je rovno T-kódování. Tak tomu je například v csplainu.

V LATEXu též implicitně nedochází ke konverzi těchto dvou kódování. Pokud se ale použije balík `inputenc.sty`, například:

```
\usepackage[cp852]{inputenc}
```

pak se všechny znaky s kódy nad 128 stávají aktivními a jejich význam je definován v souboru `*.def` (v našem příkladě `cp852.def`) prostřednictvím sekvence `\DeclareInputText`. Například pro písmeno `ř`, které je v CP852 kódováno jako `"FD` (hexadecimálně), najdeme v definičním souboru:

```
\DeclareInputText{"0FD}{\v r}
```

Znamená to, že náš znak `ř` se bude expandovat na `\v r`. Co se dále stane s touto sekvencí, záleží na volbě T-kódování, a tím se budu zabývat za chvíli.

Ve stylu `inputenc.sty` je zajímavým způsobem řešeno definování znaku, který je aktivní, a jehož kód makro `\DeclareInputText` přečte ze svého parametru. Je k tomu použit primitiv `\uppercase` zhruba takto:

```
\def\DeclareInputText #1{\bgroup \uccode`~=#1
   \uppercase{\egroup \def~}}
```

Zápis

```
\DeclareInputText{"0FD}{\v r}
```

tedy expanduje na

```
\def ř{\v r}
```

přičemž ono `ř` je aktivní token kategorie 13 s kódem `"0FD`. Jak je to uděláno? Znaku `~` je přechodně (mezi `\bgroup` a `\egroup` přiděleno `\uccode` odpovídající načtenému parametru, takže primitiv `\uppercase` změní v prvním svém průchodu znak `~` na aktivní `ř`. V druhém průchodu se uzavírá skupina (`\egroup`) a otevírá definice `\def ř`.

V případě LATEXu existuje jeden drobný zádrhel se zápisem `\'a` například pro písmeno `á`. Makro `\'` má totiž v LATEXovém prostředí tabbing pozměněný význam. Kdyby uživatel v tomto prostředí použil písmeno `á` a ono by se expandovalo na `\'a`, pak by se uživatel asi nestačil divit. Balík `inputenc.sty` proto používá pro `á` expanzi na `\@tabacckludge'a` a LATEXové jádro expanduje `\@tabacckludge'` na `\'` jen v případě, že zrovna nejsme v prostředí tabbing. Pokud v něm jsme, řeší LATEX problém dlouhého á jinak.

V LATEXu a jmenovitě v NFSS je velmi důkladně ošetřeno další chování sekvencí typu `\v r`, resp. `\'a`. To záleží na volbě T-kódování, kterému LATEX říká `\fontencoding`. Je-li zvoleno kódování CM fontů, které se v LATEXu jmenuje OT1, pak zmíněné sekvence expandují na `\accent20r`, resp. `\accent19a`. To je řečeno v souboru `ot1enc.def` na následujících řádcích:

```
\DeclareTextAccent{\'}{OT1}{19}
\DeclareTextAccent{\v}{OT1}{20}
```

T-kódování je vnitřní TEXovské kódování, ve kterém jsou načteny vzory dělení slov a ve kterém TEX vystupuje do `dvi`. Aby bylo možno použít české dělení slov, musí být T-kódování bezpodmínečně osmibitové. Předchozí příklad s kódováním OT1 je tedy pro češtinu nepoužitelný. Uvažujme proto například kódování CS-fontů, kterému LATEX říká IL2. V souboru `il2enc.def` je řečeno k našemu písmenu `ř` toto:

```
\DeclareTextComposite{\v}{IL2}{r}{248}
```

Pokud se použije font s kódováním IL2, pak bude sekvence `\v r` expandovat na znak s kódem 248. Podobných řádků, jako je tento, najdeme v souboru `il2enc.def` více. LATEX se do tohoto souboru podívá, pokud je nastaveno `\fontencoding` na IL2. Uživatel může toto kódování nastavit takto:

```
\usepackage[IL2]{fontenc}
```

Jak je makro `\DeclareTextComposite` definováno, nebudu pro nedostatek místa rozvádět. Zkuste si nastavit `\tracingmacros=2` a nechte zpracovat LATEXem sekvenci `\v r` jednou v režimu OT1 a jednou třeba při IL2. V obou případech se po pohledu do `log` souboru zhrozíte, jak intenzivně a komplikovaně je tímto problémem zatížen expand procesor. Nicméně víme, že expanze nakonec při OT1 vede na `\accent20r` zatímco při IL2 se věc expanduje na samotný znak s kódem 248.

Aktivovat akcentované znaky použitím balíku `inputenc.sty` může mít smysl třeba tehdy, pokud chceme v průběhu jednoho dokumentu měnit I-kódování nebo T-kódování. Na oba případy jsou uvedená makra připravena. Při změně I-kódování (pro přepínání píšeme `\inputenc{KÓD}`) makro jednoduše předefinuje všechny aktivní znaky podle nového kódu. Při změně T-kódování (pro přepínání píšeme `\fontencoding{jinykod}`) se zase změní způsob expanze sekvencí `\v`, `\'` a podobných.

Chceme-li změnit T-kódování v rámci zpracování jediného dokumentu, nesmíme zapomenout současně přehodit vzory dělení slov, které jsou závislé na použitém T-kódování. Je tedy potřeba načíst před zpracováním dokumentu tolik variant vzorů dělení, kolik budeme používat různých T-kódování. Změna tohoto kódování uvnitř odstavce je ovšem velice problematická. Je sice možno uvnitř odstavce změnit vzory dělení, ale už není možno změnit tabulku `\lccode` jednotlivých znaků, která je interpretována globálně pro celý odstavec. Bohužel tato tabulka také ovlivňuje dělení slov. Je proto lepší T-kódování v průběhu sazby nestřídat a raději na různě kódované fonty navázat prostřednictvím virtuálních skriptů.

Je-li I-kódování $\neq$ T-kódování, pak je potřeba upravit konverzi znaků z malých na velké a naopak, na kterou dříve stačily primitivy `\uppercase` a `\lowercase`. Tyto primitivy totiž pracují podle `\lccode` a `\uccode` znaků *před* případnou expanzí posloupnosti tokenů. Přitom `\lccode` musí být nastaveno podle T-kódování, protože tyto kódy mají přímou souvislost s dělením slov. Hodnoty `\uccode` se z důvodu symetrie také nastavují podle T-kódování. Máme-li tedy I-kódování rozdílné, musíme nejprve provést expanzi a pak teprve použít primitiv `\uppercase` nebo `\lowercase`. Problém může řešit makro `\Uppercase`, které definujeme třeba takto:

```
\def\Uppercase #1{\edef\act{\uppercase{#1}}\act}
```

Uděláme si nyní malé shrnutí konverze I-kódování ⟶ T-kódování v LaTeXu.

1a. Znak ř není aktivní: I-kódování = T-kódování.
1b. Znak ř je už v I-kódování rozložen na `\v r`: jdi na 2.
1c. Znak ř je aktivní (použitím `inputenc.sty`): jdi na 3.
2a. Jsme ve verbatim prostředí: vytiskne se `\v r`.
2b. Nejsme ve verbatim prostředí: `\v r` ⟶ T-kódování.
3. I-kódování ⟶ `\v r` ⟶ T-kódování (bez závislosti na verbatim prostředí).

Cílem tvůrců maker na konverzi I-kódování ⟶ T-kódování v LaTeXu je snaha po uživatelsky příjemnější přenositelnosti dokumentů na úrovni `*.tex` a oddělení problematiky vstupního kódování dokumentu od kódování fontů. Pokud každý „standardní" LaTeXový dokument bude obsahovat volání stylu `inputenc.sty`, pak je vlastně v hlavičce řečeno, jaké vstupní kódování použil autor dokumentu. Dokument pak není závislý na kódování, které používá příjemce dokumentu. Příjemce se totiž nemusí starat o konverzi dokumentu do svého kódování, ale rovnou dokument LaTeXuje.

Tento záměr má ale k ideálu hodně daleko. Příjemce se totiž musí starat, jak jsou v dokumentu vyznačeny konce řádků. Pokud to nevyhovuje použitému operačnímu systému, musí provést před zpracováním konverzi dokumentu (o problémech s konci řádků viz [TBN] stranu 15). Může se také stát, že příjemce dostane dokument z elektronické pošty ve „svém" kódování, zatímco odesílatel jej poslal v jiném „svém" kódování — o konverzi se automaticky postaraly systémy pro přenos elektronické pošty. Pak zápis v hlavičce dokumentu o kódování odesílatele dokonce lže a může působit potíže. Snaha o co největší pohodlí uživatele je oprávněná právě u nástrojů, které umožňují přenos dokumentů z různých operačních systémů. V tomto případě jde o programy na práci s elektronickou poštou. Snažit se o totéž v samotném LaTeXu mi připadá nevhodné s tím, že to přináší více komplikací než užitku. Laického uživatele většinou vůbec nezajímá, jaké používá kódování. Takže se mu může jevit požadavek na uvedení tohoto údaje do hlavičky v `\usepackage[...]{inputenc}` jako zbytečnost, o kterou by se měl správně nainstalovaný systém postarat sám bez jeho pomoci. Používání tohoto balíčku také předpokládá, že A-kódování = D-kódování = I-kódování, což neguje původní Knuthův záměr, aby se rozdílnosti použitých operačních systémů řešily na úrovni D-kódování ⟶ I-kódování.

Pokud jsme po zpracování TeXem ztratili přehled o tom, jak dopadlo naše písmeno ř (to se vzhledem ke složitosti cesty od D-kódování po T-kódování může stát), pak je vhodné se podívat přímo do dvi souboru. Uděláme jednoduchý TeXový soubor, který tiskne do dvi pouze jediné písmeno `ř`. Pak použijeme program `dvitype`, který vypíše obsah dvi souboru v textové podobě. Najdeme tam například povel pro sazbu znaku písmene `ř` jako `set_1 248`.

I-kódování ⟶ T-kódování — další možnosti

Při popisu transformace mezi I-kódováním a T-kódováním nesmíme zapomenout na běžné situace, které vlastně dělají TEX TEXem. Jedná se o tyto možnosti: (1) transformace mezi zápisem názvu makra a výsledným kódem v `dvi`, (2) transformace v matematickém módu podle `\mathcode` a (3) proměna skupin znaků na ligatury.

Transformace kontrolních sekvencí na výstup do `dvi` je řízena makrojazykem TEXu, který má v této oblasti velmi široké možnosti. Z důvodu stručnosti uvedeme jen jeden z mnoha příkladů. Sekvence `\alpha` na vstupu se při použití rodiny fontů Computer Modern převede na kód `"0B` (hexadecimálně) ve fontu `cmmi*`. Vysvětlit přesně, jak je to zařízeno, by bylo na delší vypravování, takže bez komentáře pouze zmíním, že to souvisí s touto deklarací:

```
\mathchardef\alpha="010B
```

V matematickém módu podléhají transformaci nejen kontrolní sekvence, ale i všechny běžné znaky. Napíšeme-li například `$-$`, pak toto „mínus" se při použití rodiny Computer Modern převede na znak s kódem 0 ve fontu `cmsy*`, protože bylo v makrojazyku řečeno:

```
\mathcode`\-="2200
```

Nakonec zmíním přeměnu skupin znaků na ligaturu. Tato vlastnost je definována v metrice fontu `tfm`. Proměna v ligaturu se odehrává na úrovni T-kódování. Máme-li například na vstupu `---` a je zrovna použit font `cmr10`, pak se tyto tři znaky promění ve znak jediný s kódem `"7C`, tj. znak „—".

T-kódování ⟶ F-kódování

Pod F-kódováním rozumíme kódování použitých fontů ve formátu PK, METAFONTu nebo Encoding vektoru PostScriptového fontu.

Je-li v `dvi` odkazováno na font, který je instalován jako `*.mf`, `*.pk` nebo PostScript a není přítomen skript virtuálního fontu `*.vf`, je T-kódování = F-kódování. To je dosti častý případ. Zde ale rozebereme případ, kdy mezi kódováním `dvi` souboru (T-kódováním) a F-kódováním stojí skript virtuálního fontu. Ten může obsahovat požadavek na změnu pozice znaku. Uvedeme abstraktní příklad. Nechť naše písmeno ř je v `dvi` na pozici 176 a má být vysázeno fontem `vcsr10`. Tento font je virtuální a odkazuje na font `csr10`, přitom pozici 176 mapuje na pozici 248. Pak bude nakonec naše písmeno ř sázeno fontem `csr10` z pozice 248. Uvedeme nyní, jak takový virtuální font `vcsr10` zhruba vypadá.

Především někde v instalaci musí být přítomen soubor `vcsr10.vf`. Je tam, kde ho najdou použité `dvi` ovladače. Chceme-li se mu podívat na zoubek, převedeme jej do čitelné podoby pomocí programu `vftovp`. Po konverzi najdeme v souboru `vcsr10.vpl` mimo jiné tyto informace:

```
(VTITLE Pouze priklad)
...
(MAPFONT D 0
```

```
  (FONTNAME csr10)
  )
...
(CHARACTER O~260
   (CHARWD R 0.391668)
   (CHARHT R 0.694445)
   (MAP
      (SETCHAR O~370)
      )
   )
```

Vidíme tedy, že znaky jsou implicitně čerpány z fontu `csr10` (viz `MAPFONT D 0`) a že pozice 260 (oktalově) je vysázena jako znak z implicitního fontu z pozice 370 (rovněž oktalově). Takto je možné překódovat i ostatní pozice. Podrobněji o virtuálních fontech — viz například [TST] stranu 75.

Na úrovni T-kódování ⟶ F-kódování může též dojít k „rozkladu" našeho písmene ř na samostatné písmeno r a samotný háček. Prostřednictvím virtuálních skriptů lze `dvi` ovladači říci, ve kterém fontu hledat toto písmeno a akcent, a popsat přesný způsob, jak tyto segmenty sesadit k sobě. V naší předchozí ukázce virtuálního fontu by mohlo být napsáno:

```
(CHARACTER O~260
   (CHARWD R 0.391668)
   (CHARHT R 0.694445)
   (MAP
      (PUSH)
      (PUSH)
      (MOVERIGHT R -0.054167)
      (SETCHAR O~24)
      (POP)
      (SETCHAR C r)
      (POP)
      )
   )
```

Poznamenejme, že tato praxe „rozkladu" znaku právě popsaným způsobem je hojně používána při použití standardních PostScriptových fontů v CSTEXu. U těchto fontů se totiž nepředpokládá, že v nich bude kresba písmene ř samostatně obsažena, protože třeba většina fontů z Adobe Type Library obsahuje pouze sadu znaků podle tzv. Adobe StandardEncoding. Tato sada má písmena anglické abecedy a samostatné akcenty všeho druhu, takže všechna akcentovaná písmena naší abecedy lze prostě poskládat ze segmentů.

Pokud ale PostScriptový font obsahuje kompletní znak ř, pak není třeba provádět zmíněný rozklad na segmenty a nastupuje poslední fáze našeho překódovacího martyria:

F-kódování ⟶ P-kódování

Pod pojmem P-kódování rozumíme pořadí PostScriptových procedur ve fontu na vykreslení jednotlivých znaků. Toto pořadí je z hlediska sázecího systému naprosto nezajímavé, protože font je v PostScriptovém kódu použit vesměs prostřednictvím operátoru `show` (a alternativ). Tento operátor konvertuje pozici sázeného znaku na název procedury pro vykreslení požadovaného znaku. Konverze je definována prostřednictvím tzv. Encoding vektoru, který je součástí každého PostScriptového fontu. Obsahuje obvykle 256 jmen PostScriptových procedur. Vysvětlíme si to na příkladě.

Nechť PostScriptový font obsahuje proceduru na vykreslení kompletního znaku ř. Tato procedura má obvykle název `/rcaron`. Předpokládejme, že naše písmeno ř má v F-kódování pozici 248. Operátor `show` se při požadavku na vykreslení tohoto znaku podívá do Encoding vektoru na pozici 248 a měl by tam najít název `/rcaron`. To je název procedury, která je použita pro vykreslení požadovaného znaku. RIP nyní interpretuje proceduru `/rcaron` a znak vykreslí. Sláva, konečně vidíme znak ř po mnoha zákrutách na papíře!

Může se stát, že máme nějaký PostScriptový font s úplnou sadou české abecedy a že nám nevyhovuje jeho kódování. Jak vyplynulo z předchozího, není nic snadnějšího, než pozměnit tomuto fontu Encoding vektor. Ten najdeme v čitelné podobě například v souborech s příponou `pfb` nebo `pfa`. Před editováním je vhodné převést font z formátu `pfb` do `pfa` (například pomocí balíku `t1utils`, programy `t1binary` a `t1ascii`) a v tomto formátu provést požadované změny Encoding vektoru. Vlastní algoritmy procedur bývají většinou zašifrované, ovšem to nám nevadí. Nakonec převedeme font zpět do formátu `pfb` a máme jej připraven v požadovaném kódování.

Pokud je font přímo implementován v PostScriptovém RIPu a jeho sada znaků obsahuje úplnou českou abecedu, pak překódování takového fontu můžeme provést níže uvedenými povely, které zařadíme na začátek PostScriptového kódu. V tomto případě nelze přímo přepsat Encoding vektor, protože font je speciální případ tzv. „slovníku“, který je nastaven pouze ke čtení. Je proto potřeba tento slovník zkopírovat, v kopii pozměnit Encoding vektor a označit jej jako nový font pod jiným názvem. Například v RIPu máme font `Helvetica-E` s Encoding vektorem podle CP1250. My bychom ale chtěli používat tento font v kódování ISO-8859-2. Připravíme si proto Encoding vektor v našem kódování (kráceno):

```
/ISOLatin2Encoding [/.notdef /./notdef ...
  /space /exclam /quotedblright /numbersign /dollar /percent ...
  /at /A /B /C /D ...
  /rcaron /uring /uacute /uhungarumlaut /udieresis /yacute ...] def
```

a definujeme nový font s názvem `Helvetica-ISOLatin2`:

```
/Helvetica-E findfont
dup length dict begin
  {1 index /FID ne {def} {pop pop} ifelse} forall
  /Encoding ISOLatin2Encoding def
```

```
  currentdict
end
/Helvetica-ISOLatin2 exch definefont pop
```

Nakonec místo fontu `Helvetica-E` použijeme font `Helvetica-ISOLatin2`. Podrobněji o této technologii viz [RB].

Zásahy do fontu na úrovni F-kódování $\longrightarrow$ P-kódování jsou nezávislé na použitém sázecím systému. Není-li ale k dispozici PostScriptový font s úplnou sadou českých znaků, pak je nutno provést „rozklad" akcentovaných znaků na segmenty už na úrovni T-kódování $\longrightarrow$ F-kódování, což je ovšem výsadou TEXovských instalací a virtuálních fontů.

Výsadou TEXovských instalací je rovněž ovladač `dvips`, kterému můžeme sdělit požadovaný Encoding vektor použitého fontu přímo prostřednictvím konfiguračního souboru `psfonts.map`. Je-li tam například napsáno:

```
csr10 dcr10 "XL2encoding ReEncodeFont" <xl2.enc <dcr10.pfb
```

pak se pro sazbu fontu `csr10` použije font `dcr10.pfb`, ovšem s pozměněným Encoding vektorem. Program `dvips` zavede do PostScriptového kódu font `dcr10.pfb`, ale změní mu Encoding vektor tak, jak je řečeno v souboru `xl2.enc`. To je textový soubor, kde je požadovaný Encoding vektor zapsán. Nemusíme tedy tento vektor měnit přímo v souborech `pfb` a `pfa`. Podrobněji viz [DVIPS].

Reference

[WWW] Stránka o kódování češtiny: `http://www.cuni.cz/cestina`.

[FTP] Přehledné a důkladné tabulky jedenácti kódování používaných v češtině: `ftp://math.feld.cvut.cz/pub/tkadlec/text/codestab.ps`. Dostupné též ze stránky: `http://math.feld.cvut.cz/tkadlec/others.htm`

[RB] Adobe Systems Incorporated. *PostScript Language Reference Manual.* Addison-Wesley, Reading, MA, USA, second edition, 1990.

[DVIPS] Tom Rokicki. `dvips` — *dvi to PostScript.* Manuál k programu `dvips` ve formátu `*.tex` přiložen v distribuci programu.

[EC] Milan Zamazal. Balík pro počeštění Emacsu `emacs-czech`, `http://www.fi.muni.cz/~pdm/emacs-czech.html`.

[ENCTEX] Petr Olšák. *EncTEX – změny konverzních tabulek TEXu.* Zpravodaj 3/1997. Balík encTEX rozšiřující standardní TEX o možnost manipulace s `xord`/`xchr` vektory. `ftp://math.feld.cvut.cz/pub/olsak/enctex/`.

[TBN] Petr Olšák. *TEXbook naruby.* Konvoj, Brno 1997. V elektronické podobě plný text k dispozici na `http://math.feld.cvut.cz/olsak/tbn/`.

[TST] Petr Olšák. *Typografický systém TEX.* CSTUG 1995.