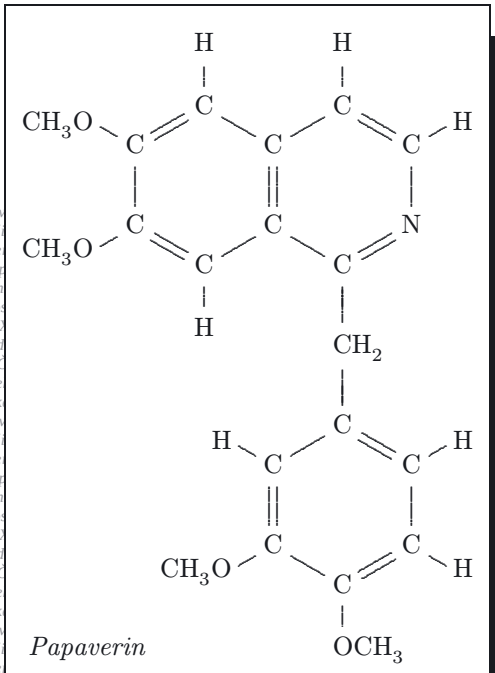
CSTUG



ZPRAVODAJ

Československého sdružení uživatelů TEXu

1

2006

ISSN 1211-6661 Tištěná verze

ISSN 1213-8185 Online verze

Ročník 16

OBSAH

V tomto čísle je dodatečně distribuováno CD/DVD TEX Live 2005.

Zpravodaj Československého sdružení uživatelů TEXu je vydáván v tištěné podobě a distribuován zdarma členům sdružení. Po uplynutí dvanácti měsíců od tištěného vydání je poskytován v elektronické podobě (PDF) ve veřejně přístupném archívu dostupném přes `http://www.cstug.cz/` .

Své příspěvky do Zpravodaje můžete zasílat v elektronické podobě, nejlépe jako jeden archivní soubor (`.zip, .arj, .tar.gz`). Postupujte podle instrukcí, které najdete na stránce `http://bulletin.cstug.cz/` . Pokud nemáte přístup na Internet, můžete zaslat příspěvek na disketě na adresu:

Zdeněk Wagner
Vinohradská 114
130 00 Praha 3

Disketu formátujte nejlépe pro DOS, formáty Macintosh 1.44 MB a EXT2 jsou též přijatelné. Nezapomeňte přiložit všechny soubory, které dokument načítá (s výjimkou standardních součástí CSTEXu), zejména v případě, kdy vás nelze kontaktovat e-mailem.

ISSN 1211-6661 (tištěná verze)
ISSN 1213-8185 (online verze)

Milí přátelé.

Dostáváte první číslo Zpravodaje CSTUGu roku 2006. Ti z vás, kteří byli členy CSTUGu již v loňském roce a zaplatili příspěvky, v něm naleznou jedno DVD a dvě CD (kolektivní členové tři DVD a šest CD) s distribucemi TEXLive a proTEXt. V případě jakýchkoli nesrovnalostí se prosím obraťte na sekretářku našeho sdružení p. Holovskou. Jak již bylo řečeno v minulém čísle, do budoucna počítáme vzhledem k finanční náročnosti s tím, že členové budou dostávat jen kompletní distribuci na DVD vzhledem k tomu, že DVD mechanikou je dnes vybavena naprostá většina počítačů.

Loni 26. 11. proběhla na FI MU v Brně valná hromada CSTUGu. Doprovodnou akcí byla přednáška pana Bogusława Jackowského *Latin modern fonts at eleventh hour*. Přítomní mohli sami posoudit, že šlo o velmi zajímavou a kvalitně připravenou přednášku. Na ni navázala odpoledne rozsáhlá diskuse, která vyústila do rozhodnutí o spolupráci při dokončení projektu LM fontů a o finanční podpoře, kterou CSTUG na tento projekt poskytne.

Zklamáním pro výbor CSTUGu byla velmi malá účast na valné hromadě. Ta proběhla podle následujícího programu:

1. Zahájení valné hromady.
2. Zpráva o činnosti.
3. Zpráva o hospodaření.
4. Zpráva revizorů účtu.
5. Informace o přípravě DVD/CD TEXLive 2005.
6. Rámcový plán činnosti pro rok 2006.
7. Různé, diskuse.
8. Závěr.

Jednotlivé dokumenty z valné hromady i přednášky lze nalézt na stránkách sdružení `www.cstug.cz`.

Výbor CSTUGu schválil grantové projekty, které podle jeho mínění budou užitečné pro českou a slovenskou komunitu uživatelů TEXu:

1. Česká a slovenská podpora pro balík babel LATEXu.
2. Užitečné Latin Modern pro Čechy, Moraváky a Slováky.
3. Český překlad manuálu ConTEXt pro začátečníky

Podrobnosti budou zveřejněny rovněž na stránkách sdružení.

Na závěr bych vám všem chtěl popřát v letošním roce pevné zdraví a hodně osobních i TEXovských úspěchů.

Jaromír Kuben
předseda CSTUGu

Sazba chemických vzorců v TEXu

Ondřej Jakubčík

Sázecí program TEX je mnohokrát osvědčeným nástrojem pro sazbu všech druhů dokumentů. V poslední době jsem se zabýval sazbou chemických textů. Protože se nejedná o problematiku právě triviální, pokusil jsem se poskytnout pár základních informací širšímu okruhu uživatelů programu TEX.

Úvod

Pro sazbu chemických vzorců mi nejprve byl doporučen program ChemSketch. I přes jeho nesporné kvality jsem jej nepoužil, neboť jednak používám jiný operační systém než Windows, a jednak se snažím používat free software.

Nejprve jsem k sazbě používal LATEX a systém XƎMTEX, ale protože zpracování dokumentu trvalo čím dál tím déle a dokument stále nabýval na rozsahu, raději jsem přešel k plainu.

Můj zájem se tedy stočil směrem k METAFONTu. Zde jsem zůstal u tří symbolů pro vazby, protože jsem si nedovedl představit vytváření vzorců složitých organických látek.

Nakonec mne zaujal systém PPCHTEX, který je součástí balíku ConTEXt. Jeho velkou předností je, že funguje i v plainu, tudíž jsem mohl zachovat stávající úpravu dokumentu.

Jaký použít formát

Při výběru, v jakém formátu budeme dokument sázet (pokud není zadán), je dobré vybírat ten, který nám nejvíce vyhovuje. Pokud je dokument velmi dlouhý, je dobré se rozhodovat i podle rozsahu dokumentu, aby např. ConTEXt v půlce dokumentu neskončil s nedostatkem paměti. LATEX zase v základním nastavení rád „plýtvá“ papírem. Pokud si výsledný dokument nebudeme tisknout sami, nebo pokud jej nebudeme předávat k tisku v PostScriptu nebo v PDF, musíme myslet i na konfiguraci systému v tiskárně, a nejlépe použijeme formát, o kterém stoprocentně víme, že je nainstalován všude (plain).

Předávání dokumentu v PostScriptu a PDF může také vyvolat poměrně velkou nevoli, protože zvláště velký dokument se může připravovat k tisku velmi dlouhou. Sto třicet stran dlouhý dokument s dvěma sty vzorci a třiceti skenovanými obrázky v PostScriptu v normální kvalitě (600 dpi) zabíral okolo 77 MB, takže jsem jej při tisku na tiskárně s 8 MB paměti tiskl po deseti stránkách.

Celková úprava dokumentu

Chemie, zvláště pak organická, bývá velmi náročná na místo na stránce. Proto, pokud nejsme omezeni počtem stran a pokud musíme stanovit rozsah předem, je lépe počínat si velkoryse a klidně pár (desítek) stran přidat, samozřejmě podle povahy dokumentu.

Co se týče velikosti, na papír formátu A4 se toho vejde více než na A5. Pokud sázíme vzorce anorganických sloučenin nebo jen velmi jednoduchých organických látek, pak vystačíme i s formátem A5. Pokud však máme sázet anorganické rovnice o několika reaktantech nebo vzorce složitých organických látek (typicky polymerů, bílkovin, nukleových kyselin atp.), je lépe použít papír A4. PPCHTEX sice umožňuje zjemnit kresbu a zmenšit písmo, ale mohlo by to být na úkor čitelnosti.

Písmo

Při tvorbě dokumentu, díky kterému vlastně vznikl tento článek, jsem byl postaven před problém výběru písma. Na prvních třiceti stránkách jsem zkoušel několik desítek písem, od standardních postscriptových, přes různé volně šiřitelné fonty, vyzkoušel jsem i Lido ze Střešovické písmolijny. Přes nesporné kvality dostupných písem jsem se nakonec rozhodl pro $\mathcal{CS}$ variantu Computer Modern, protože v dokumentu se vyskytovalo mnoho matematických vzorců a vzhledem k povaze práce nebylo možné použít (resp. koupit) například Math Times nebo jiný matematický font.

Jednotnost názvů

Každý dokument se čte nejlépe, pokud nejen vypadá pěkně na pohled, ale také pokud pro jednu věc používáme jeden pojem. České chemické názvosloví přímo překypuje možnostmi, ale je nejlépe držet se oficiálních názvů podle IUPAC. Pokud budeme sázet z více předloh, není špatné sjednotit označení, tzn. nepsat jednou I a podruhé J (jod). Pokud nedostaneme odlišné zadání, preferujeme raději nové názvosloví, tj. oxid místo kysličník, nebo sulfid místo sirník atp.

Systematické názvy

Systematická jména anorganických sloučenin a triviální názvy jak organických, tak i anorganických látek jsou ve své podstatě nejjednodušší z hlediska typografie. Sloučeniny jako např. síran draselný, dihydrogenfosforečnan sodný, nebo anilin sázíme většinou základním písmem dokumentu.

S organickými látkami se stává problematika podstatně zajímavější. Protože téměř vždy je nutné udávat pozice atomů v molekule, sázejí se čísla, udávající pozici substituentu, oddělená navzájem čárkou bez mezer a od názvu substituentu oddělená spojovníkem. Prakticky to vypadá asi takto: 1,2-dimethylbenzen. Měli bychom dávat pozor, aby se nám v případě rozdělení slova za spojovníkem napsal spojovník i na začátek dalšího řádku, což můžeme vyřešit použitím sekvence `\discretionary{-}{-}{-}`.

Úvod do PPCHTEXu

PPCHTEX je soubor maker pro práci s chemickými vzorci. Ačkoli je součástí ConTEXtu, je možné ho použít i v LATEXu a plainu. Jediná nutnost je mít nainstalovaný PICTEX nebo PSTricks.

Pokud používáte TeX Live, pak jej možná již máte nainstalovaný, a pokud nikoliv, pak je snadné jej doinstalovat. Pokud používáte jiný systém, je ConTEXt a s ním i PPCHTEX dostupný v CTANu (/macros/context/current).

Po nainstalování a aktualizaci kpathsea je systém připraven k použití.

Zavedení maker

Aby bylo možno pracovat s makry, musíme je nejprve načíst. V plainu to zařídíme přidáním následujících dvou řádků na začátek souboru:

```
\input m-pictex
\input m-ch-en
```

V LATEXu vložíme toto:

```
\usepackage{m-pictex}
\usepackage{m-ch-en}
```

V ConTEXtu je příprava nejsnazší:

```
\usemodule[pictex,chemic]
```

Vkládání vzorců

Samostatné vzorce se vkládají do bloku `\startchemical` ... `\stopchemical`, rovnice se vkládají do bloku uvozeného v LATEXu i v plainu pomocí `$$`, v systému ConTEXtu pomocí `\placeformula\startformula` ... `\stopformula`. V těchto blocích se jednotlivé části vzorce vkládají do příkazu `\chemical`, který se používá i pro vkládání vzorce do normálního textu.

Vzorce anorganických sloučenin

Vzorce anorganických sloučenin vytváříme pomocí příkazu `\chemical`, do kterého napíšeme atomy a indexy stejně jako v TEXu. Indexy a exponenty se nemusí uzavírat mezi dolary. Například vzorec pentahydrátu síranu měďnatého (modrá skalice) $CuSO_4 \cdot 5H_2O$ vložíme do textu jako `\chemical{CuSO_4 \cdot 5 H_2O}`.

Příklad sazby vzorců anorganických sloučenin

Při výrobě elektrolytické mědi se měď rozpouští v horké kyselině sírové, přičemž vzniká síran měďnatý, chemicky $CuSO_4$. Tento děj je popsán rovnicí:

$$\mathrm{Cu} \quad + \quad \mathrm{H_2SO_4} \qquad \longrightarrow \qquad \mathrm{CuSO_4} \quad + \quad \mathrm{H_2}$$

Z roztoku síranu měďnatého se poté elektrolyticky vyloučí měď.

Tento odstavec byl vysázen s pomocí následujícího kódu:

```
Při výrobě elektrolytické mědi se měď rozpouští v~horké kyselině
sírové, přičemž vzniká síran měďnatý, chemicky \chemical{CuSO_4}.
Tento děj je popsán rovnicí:
$$ \chemical{Cu} \chemical{PLUS} \chemical{H_2SO_4}
\chemical{GIVES} \chemical{CuSO_4} \chemical{PLUS}
\chemical{H_2} $$
Z~roztoku síranu měďnatého se poté elektrolyticky vyloučí měď.
```

Jak je vidět v příkladu, PPCHTEX obsahuje i symboly jako plus, reakční šipku, a samozřejmě i jiné. Například iontovou rovnici rozpouštění soli ve vodě

$$\mathrm{NaCl} \quad \underset{\text{krystalizace}}{\overset{\text{rozpouštění}}{\rightleftharpoons}} \quad \mathrm{Na^{\oplus}} \quad + \quad \mathrm{Cl^{\ominus}}$$

můžeme napsat takto:

```
$$ \chemical{NaCl}
\chemical{EQUILIBRIUM}{rozpouštění}{krystalizace}
\chemical{Na^\oplus} \chemical{PLUS} \chemical{Cl^\ominus}$$
```

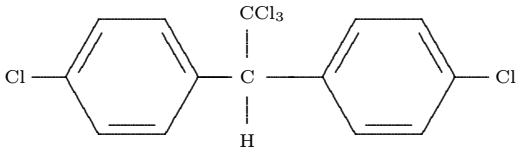
Většina symbolů má i svou zkrácenou formu, takže místo `EQUILIBRIUM` stačí psát `<->`.

Vzorce organických sloučenin

Jako u každého komplexního systému, je mnoho způsobů, jak dosáhnout kýženého výsledku. Prvním způsobem je sázení zleva doprava, tj. píšeme příkazy tak, že začneme od části vzorce nejvíce vlevo a postupně se propracováváme doprava. Tento přístup jsem používal poměrně často, avšak nevýhodu jsem poznal

až poté, co jsem musel vzorce podle zadání opravit. Proto se mi zdá s odstupem času lepší pracovat opravdu chemicky a strukturu rozložit na jednotlivé stavební bloky. Protože PPCHTEX umožňuje ukládat části vzorců, je asi nejlepší vytvořit si seznam stavebních látek a pak už molekuly skládáme dohromady velmi snadno. Výhoda vyvstane v okamžiku korektury, neboť pokud uděláme chybu ve zbytku, opravíme ji jen jednou a nemusíme procházet celý dokument.

Abych ukázal oba způsoby sazby, představme si, že budeme chtít vysázet následující vzorec:



Způsobem zleva doprava by vypadal asi takto:

```
\startchemical
\chemical[ONE,Z0,SIX,ROT2,B,EB135,R36][Cl]
\chemical[MOV1,ONE,Z037,SB37][C,H,CCl_3]
\chemical[SIX,MOV1,ROT2,B,EB135,R63,MOV1,ONE,Z0][Cl]
\stopchemical
```

Druhým způsobem (struktura) asi takto:

```
\definechemical[chlorfenyl1]{
\chemical[SAVE,SUB1,SIX,ROT2,B,EB135,R36,RZ6,RESTORE][Cl]
}

\definechemical[chlorfenyl2]{
\chemical[SAVE,SUB3,SIX,ROT4,B,EB246,R36,RZ6,RESTORE][Cl]
}

\startchemical
\chemical[ONE,Z037,SB1357][C,H,CCl_3]
\chemical[chlorfenyl1]
\chemical[chlorfenyl2]
\stopchemical
```

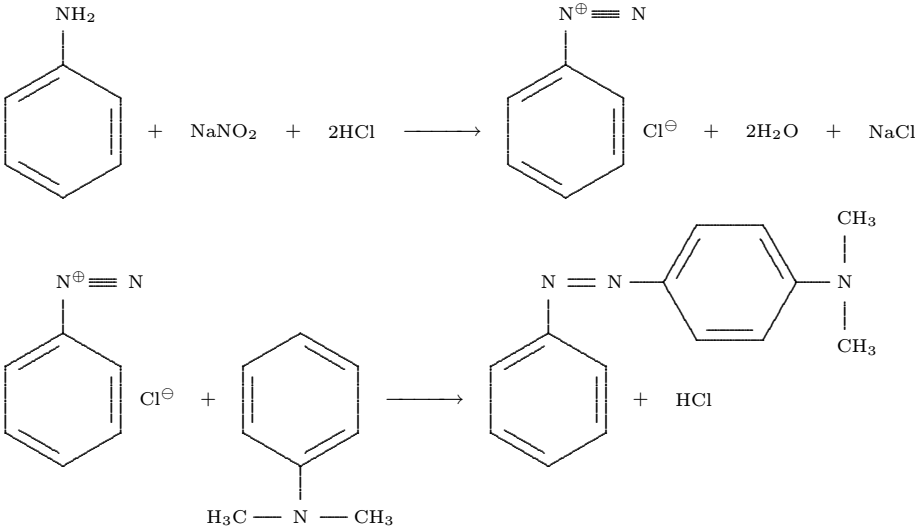
Druhý zápis je delší, nicméně struktura je jasná i z kódu. Na druhou stranu, pokud se jedná o vzorec tak jednoduchý jako H—C≡C—H, bylo by použití něčeho jiného než `\chemical{H,-,C,---,C,-,H}` čiročiré plýtvání.

Jednotlivé příkazy jsou pojmenovány poměrně intuitivně, takže např. MOV je posun, ROT otočení, B vazba, SB jednoduchá vazba, EB dvojná vazba uvnitř cyklu atd.

Čísla u příkazů určují směr nebo pozici. U neotočených struktur je 1 vpravo. Všechny příkazy jsou důkladně vysvětleny v manuálu [1].

Příklad sazby vzorců organických sloučenin

Diazotací anilinu pomocí $NaNO_2$ v kyselém prostředí vznikne benzendiazoniumchlorid, který kopulací s dimethylanilinem přeměníme na azobarvivo — 4-dimethylaminoazobenzen.



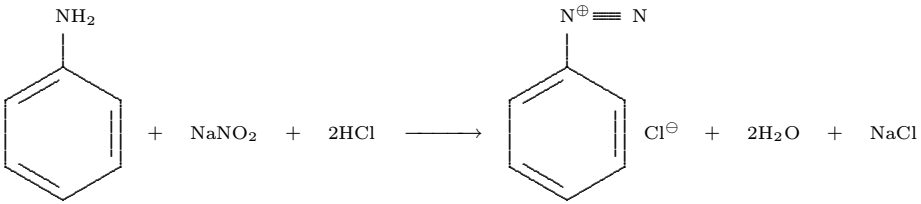
Tyto vzorce vysázíme například takto:

```
Diazotací anilinu pomocí \chemical{NaNO_2} v~kyselém prostředí
vznikne benzendiazoniumchlorid, který kopulací s~dimethylanilinem
přeměníme na azobarvivo --- 4-dimethylaminoazobenzen.

{\setupchemical[width=fit,height=3000,bottom=1000]
\hbox{
\startchemical
\chemical[SIX,B,EB135,R6,RZ6][NH_2]
\stopchemical
\startchemical
\chemical[SPACE,PLUS,SPACE]
\stopchemical
\startchemical
\chemical[CHEM][NaNO_2]
\stopchemical
\startchemical
\chemical[SPACE,PLUS,SPACE]
\stopchemical
\startchemical
```

```
\chemical[CHEM][2HCl]
\stopchemical
\startchemical
\chemical[SPACE,GIVES,SPACE]
\stopchemical
\startchemical
\chemical[SIX,B,EB135,R6,PB:RZ6,ONE,Z0,OFF1,TB1,MOV1,Z0,
PE][\SL{N^\oplus},N]
\stopchemical
\startchemical
\chemical[SPACE,SPACE,Z0][\SL{Cl^\ominus}]
\stopchemical
\startchemical
\chemical[SPACE,PLUS,SPACE]
\stopchemical
\startchemical
\chemical[CHEM][2H_2O]
\stopchemical
\startchemical
\chemical[SPACE,PLUS,SPACE]
\stopchemical
\startchemical
\chemical[CHEM][NaCl]
\stopchemical
}}

{\setupchemical[width=fit,height=5000,bottom=2000]
\hbox{
\startchemical
\chemical[SIX,B,EB135,R6,PB:RZ6,ONE,Z0,OFF1,TB1,MOV1,Z0,
PE][\SL{N^\oplus},N]
\stopchemical
\startchemical
\chemical[SPACE,SPACE,Z0][\SL{Cl^\ominus}]
\stopchemical
\startchemical
\chemical[SPACE,PLUS,SPACE]
\stopchemical
\startchemical
\chemical[SIX,ROT3,B,EB135,R6,PB:RZ6,ONE,Z051,SB51,PE][N,H_3C,
CH_3]
\stopchemical
```

```
\startchemical
\chemical[SPACE,GIVES,SPACE]
\stopchemical
\startchemical
\chemical[SIX,B,EB135,R6,PB:RZ6,ONE,ZO,DB1,MOV1,ZO,SIX,MOV1,ROT2,
B,EB135,R36,MOV1,ONE,ZO37,SB37,PE][N,N,N,CH_3,CH_3]
\stopchemical
\startchemical
\chemical[SPACE,PLUS,SPACE]
\stopchemical
\startchemical
\chemical[CHEM][HCl]
\stopchemical
}}
```

Oku typografa samozřejmě neunikne, že účaří $NaNO_2$ a HCl nebo $2H_2O$ a NaCl jsou vůči sobě posunuta. Tento nedostatek PPCHTEX neřeší a ani nastavení `\fontdimen16` a `\fontdimen17` nepomůže, neboť PPCHTEX oba registry při sazbě přepíše vlastní hodnotou, a obnoví hodnoty, jakmile dosází vzorec. Opticky toto nepůsobí pěkně, proto pokud na vzhledu obzvláště záleží, pokud např. `[HCl]` přepíšeme na `[HCl_{\vphantom{2}}]`, dostaneme již opticky pěkný výsledek:



Závěr

Sazba chemických vzorců v TEXu je poměrně náročná, ale protože PPCHTEX je dobře dokumentován, jde o náročnost spíše časovou. Při práci na chemickém dokumentu není na škodu znalost aspoň středoškolské chemie a je dobré spolupracovat s chemiky, neboť pokud si nejsme jisti například vaznostmi, je pravděpodobné vnesení chyb, které se zpětně špatně hledají a opravují. Periodická tabulka prvků a stručný přehled chemického názvosloví jsou nenahraditelné při jakékoli práci s chemickými vzorci. Korekturu je dobré rozdělit na korekturu faktickou, kterou provádí chemik, a typografickou, kterou provádí sazeč/typograf.

Tento článek jsem psal spíše jako ukázku toho, co je se systémem PPCHTEX možné vysázet, a vzhledem k rozsahu není možné suplovat dokumentaci, o což

jsem se ani nesnažil. Přestože je dokumentace v angličtině, každý příkaz je ilustrován v několika příkladech, a tak se domnívám, že každý, kdo bude sázet chemické vzorce, bude schopen se v manuálu velmi rychle zorientovat.

Reference

[1] Hagen, J., Otten, A. F.: *PPCHTEX: A macropackage for typesetting chemical structure formulas*, Pragma ADE, 2001 (`http://www.pragma-ade.com/general/manuals/mp-ch-en.pdf`)
[2] Olšák, P.: *TEXbook naruby*, Konvoj Brno, 2001
[3] Vacík, J. a kol.: *Přehled středoškolské chemie*, SPN Praha, 1990
[4] Bína, J. a kol.: *Malá encyklopédia chémie*, Obzor Bratislava, 1981

Summary: Typesetting chemical formulas in TEX

Typesetting program TEX is used worldwide for formatting various kinds of documents. In this article, a short presentation of its abilities on typesetting various chemical structures is given.

Ondřej Jakubčík
e-mail: `ojakubcik@seznam.cz`

Hrstka tipů pro TEXovskou sazbu

Petr Březina

Rozsah vyjádřený pomlčkou

Jednou z věcí, kterými se odlišuje knižní sazba od strojopisu, je pomlčka. Zatímco se na psacím stroji používá pro spojovník a pomlčku stejný znak, ve fontech knižního písma nalezneme kromě spojovníku dokonce dvě pomlčky. Jedna je dlouhá (čtverčíková) a druhá je krátká (půlčtverčíková). V anglických textech se každá z těchto dvou pomlček objevuje v jiné funkci. V české sazbě se však ve všech funkcích používá zpravidla jen jedna pomlčka. Začátečníci bývají na rozpacích, kterou pomlčku si mají vybrat. Podíváme-li se do starších českých knih, zjistíme, že mistři sazeči dávali přednost spíše dlouhé pomlčce.

V současnosti se v knihách velmi často objevuje krátká pomlčka. Já mám však raději dlouhou pomlčku a tu také obvykle používám při sazbě svých dokumentů.

Pomlčka může mít v textu řadu funkcí. Jednou z nich je vyjádření rozsahu; to znamená, že pomlčka zastupuje slůvko „až“. V takovém případě se pomlčka klade bez mezer mezi dva výrazy, které označují začátek a konec rozsahu, např. *v letech 1985—1993*. Takto ovšem vznikne poměrně dlouhý celek, který by mohl při řádkovém zlomu činit potíže. Proto je dovoleno v případě potřeby provést řádkový zlom v místě pomlčky; pomlčka ze sazby zmizí a místo ní se do textu vloží ono již zmíněné slůvko „až“, jak to můžete vidět např. zde: *v letech 1985 až 1993*. V TEXu lze definovat makro `\až` tak, že uživatel bude moci do zdrojového textu svého dokumentu zapsat např. `v~letech 1985\až 1993` a TEX sám rozhodne, zda se v daném případě vysází pomlčka, anebo slůvko „až“. Nejdříve se podíváme na to, jak bývá obyčejně toto makro definováno, a potom nabídneme jinou definici tohoto makra.

Pro řešení uvedeného typografického pravidla TEX poskytuje primitivní příkaz `\discretionary`, který má tuto formu zápisu:

`\discretionary{`⟨*prebreak*⟩`}{`⟨*postbreak*⟩`}{`⟨*nobreak*⟩`}`

Parametr ⟨*nobreak*⟩ určuje, co se má vysázet, jestliže v místě výskytu příkazu `\discretionary` nedojde k řádkovému zlomu. V opačném případě se na konec jednoho řádku vysází parametr ⟨*prebreak*⟩ a na začátek druhého řádku parametr ⟨*postbreak*⟩. Významným omezením je skutečnost, že žádný z těchto tří parametrů nesmí obsahovat výplněk typu glue, tj. pružnou mezeru, jakou je mezislovní mezera nebo třeba mezera vytvořená pomocí příkazu `\hskip`.

Obvykle se můžeme setkat přibližně s takovouto definicí makra `\až`:

```
\def\až{\discretionary{\hbox{ až}}{}{---}}
```

Vidíme, že mezeru, kterou je potřeba vložit před slůvko „až“, se autor makra pokusil zamaskovat tím, že ji uzavřel do `\hbox`u. Tímto způsobem sice úspěšně obešel zákaz výskytu výplňků typu glue v parametrech příkazu `\discretionary`, ale mezera ztratila svoji pružnost a stala se z ní mezera s pevnou šířkou. Pevná mezera před „až“ může zaujmout čtenářovu pozornost, a tak ho při četbě rozptylovat, neboť čtenář zde podvědomě očekává obyčejnou mezeru. Jestliže se řádek stáhne, bude se čtenáři zdát, že je tato mezera příliš široká. Pokud se naopak řádek rozpálí, bude se čtenáři tato mezera jevit jako příliš úzká. Míra rušivosti této mezery bude záviset na míře deformace mezislovních mezer v daném řádku. Ve většině případů asi nebude tento jev nijak tragický a nepatřičné mezery si všimne jen vskutku vnímavý čtenář. Bylo by ovšem lepší, kdyby mezera před „až“ podléhala stejné deformaci jako ostatní mezislovní mezery v daném řádku. Tím by se také nepatrně zvětšil manévrovací prostor TEXu při zalamování odstavce do jednotlivých řádků.

Chceme-li popsaný nedostatek makra `\až` odstranit, musíme si na pomoc přibrat primitivní příkaz `\cleaders`. Tento příkaz dovede proměnit libovolný box ve výplněk typu glue. Přesněji řečeno, vytvoří výplněk typu glue požadované velikosti a na tento výplněk opakovaně vykreslí obraz příslušného boxu. V horizontálním módu můžeme příkaz `\cleaders` zapsat takto:

`\cleaders`⟨*box*⟩`\hskip`⟨*velikost*⟩

Parametr ⟨*velikost*⟩ určuje požadovanou velikost (šířku) výplňku typu glue; kromě základní velikosti lze uvést i hodnoty roztažitelnosti a stažitelnosti, uvozené klíčovými slovy `plus` a `minus`. Parametr ⟨*box*⟩ reprezentuje box, jehož obraz se má opakovaně vykreslit na výplněk typu glue; v rámci tohoto parametru je možno pomocí příkazu `\hbox`, `\vbox` či `\vtop` vytvořit nový box nebo pomocí příkazu `\box` či `\copy` použít již hotový box, který je uložen v registru. Obraz boxu bude vykreslen tolikrát, kolikrát se šířka tohoto boxu vejde do definitivní šířky výplňku typu glue. Jednotlivé kopie obrazu budou umístěny těsně vedle sebe a jako celek budou zarovnány na střed výplňku typu glue.

Horizontální výplněk typu glue vytvořený pomocí `\cleaders` je odstranitelný element, v němž je možno provést řádkový zlom, pokud bezprostředně před ním předchází nějaký neodstranitelný element. Z horizontálního seznamu ho lze odstranit pomocí příkazu `\unskip`; na jeho velikost se lze dotazovat prostřednictvím příkazu `\lastskip`. Od běžného horizontálního výplňku typu glue (tj. od pružné horizontální mezery) se liší jednak tím, že na sobě může nést nějaký obraz, jednak tím, že má výšku a hloubku odpovídající výšce a hloubce boxu uvedeného jako parametr ⟨*box*⟩.

Vyzbrojeni vědomostmi o primitivním příkazu `\cleaders` se můžeme pustit do definování makra `\až`. Makro vysází rozsahovou pomlčku, která se v případě potřeby dokáže proměnit ve slůvko „až“, před nímž bude předcházet pružná mezislovní mezera.

```
\def\až{\leavevmode
   \null
   \nobreak\ \skip0=\lastskip
   \discretionary{až}{}{}%
   \nobreak\hskip-\skip0
   \setbox0=\hbox{---}%
   \cleaders\copy0\hskip\wd0
   \null}
```

Nejdříve se ubezpečíme, že nejsme ve vertikálním módu; v zásadě to dělat nemusíme, neboť smysluplné užití makra `\až` ve vertikálním módu nepřipadá v úvahu. Potom do horizontálního seznamu vložíme prázdný `\hbox` a nezlomitelnou mezislovní mezeru; do registru `\skip0` si uschováme velikost této mezery a do horizontálního seznamu dále vložíme `\discretionary`, které se v případě

řádkového zlomu promění ve slůvko „až“, umístěné na samém konci řádku. Pak následuje nezlomitelná záporná mezislovní mezera. Je dobré uvědomit si, že znaménko minus, které jsme zapsali před registr `\skip0`, ovlivní nejen základní velikost mezery, ale také hodnoty roztažitelnosti a stažitelnosti. Potom si do registru `\box0` připravíme `\hbox` s pomlčkou a pomocí `\cleaders` ho proměníme ve výplněk typu glue. Nakonec do horizontálního seznamu přijde prázdný `\hbox`.

Nyní bychom si měli vysvětlit, jak toto makro funguje. Zatím si nevšímejme prázdných `\hbox`ů, které se vkládají do sazby na začátku a na konci makra; k jejich významu se vrátíme za chvilku.

Začněme popisem situace, kdy se materiál z makra `\až` ocitne někde uprostřed řádku, takže v tomto materiálu nedojde k řádkovému zlomu. Nejprve se vysází mezislovní mezera; tato mezera bude vzápětí vyrušena zápornou mezerou, takže ve výsledné sazbě nebude žádná mezera vidět. Ve výsledné sazbě se tedy ukáže pouze pomlčka, která se vykreslí v rámci výplňku typu glue, vytvořeného pomocí `\cleaders`.

Teď si pro změnu představme, že v materiálu z makra `\až` dojde k řádkovému zlomu. V sazbě se nejdříve objeví pružná mezislovní mezera, za ní se z `\discretionary` vloží slůvko „až“, za nímž bude proveden řádkový zlom. Protože za místem řádkového zlomu automaticky mizí všechny odstranitelné elementy, ze sazby zmizí nezlomitelná záporná mezera a pomlčka.

Rozsahová pomlčka se nejčastěji klade mezi dvě čísla, např. *1985—1993;* občas se ale může vyskytnout také mezi dvěma slovy, např. *leden—červen.* Při použití rozsahové pomlčky mezi dvěma slovy by se nám většinou příliš nelíbilo, kdyby na konci řádku došlo k rozdělení jednoho nebo druhého z nich, např. *leden—červen.* Proto makro `\až` vkládá do horizontálního seznamu dva prázdné `\hbox`y. První `\hbox` zabrání dělení slova před pomlčkou; uplatní se zde toto pravidlo: Jestliže hned za slovem po přeskočení znaků, které nejsou písmeny, následuje box, linka, matematický vzorec nebo `\discretionary`, slovo nepodléhá automatickému dělení. Druhý `\hbox` zabrání dělení slova za pomlčkou, neboť slovo podléhá automatickému dělení, pouze pokud těsně před ním po přeskočení znaků, které nejsou písmeny, předchází výplněk typu glue. Dodejme, že výrazem „znaky, které nejsou písmeny“ se rozumí znaky s nulovým `\lccode`; jedná se tedy především o interpunkční znaménka.

Pokud by uživateli nevadilo případné dělení slov obklopujících rozsahovou pomlčku, může samozřejmě z definice makra `\až` odstranit řídicí sekvence `\null`. Pak by ovšem bylo vhodné vložit na konec definice makra `\až` prázdný příkaz `\relax`, tak aby jím byl řádně ukončen parametr ⟨*velikost*⟩ z konstrukce `\cleaders`.

Zvýraznění prvního řádku v odstavci

Začátky kapitol poskytují typografovi vhodnou příležitost, aby uplatnil své umělecké nadání. V úpravě začátků kapitol hraje významnou roli vstupní odstavec; existuje řada možností jeho typografického zpracování. Pro inspiraci uvádíme na následující straně několik příkladů, jak je možno vstupní odstavec ztvárnit. V tomto textu se budeme podrobněji zabývat zvýrazněním prvního řádku odstavce verzálkami, popřípadě i jiným písmem.

Máme-li za úkol vysázet první řádek odstavce z verzálek, mohli bychom postupovat následujícím způsobem.

Nejdříve si definujeme makro, které pomocí příkazu `\uppercase` převede malá písmena na velká a zároveň dočasně zvětší velikost mezislovní mezery.

```
\def\verzalky#1{{\spaceskip=.5em plus.166666em minus.166666em
   \uppercase{#1}}}
```

Potom si na zkoušku vysázíme celý odstavec z verzálek. Protože nás zajímá pouze první řádek tohoto odstavce a rádi bychom dosáhli jeho optimálního vzhledu, nastavíme pomocí příkazu `\parshape` délku druhého a dalších řádků na největší možnou hodnotu. Tím vlastně potlačíme vliv těchto řádků na rozhodování o místě zalomení prvního řádku. Příslušná část kódu ve zdrojovém souboru by mohla vypadat takto:

```
\noindent
\parshape 2 0pt\hsize 0pt\maxdimen
\verzalky{Jiným vhodným způsobem zvýraznění začátků
...
řádku, je už tvořena malými písmeny.}
```

Po zpracování zdrojového souboru TEXem a zobrazení výsledné sazby získáme přesnou představu o tom, jaká část textu se vejde na první řádek. Podle toho upravíme množství textu zahrnutého do parametru makra `\verzalky`, v případě potřeby doplníme na konec tohoto parametru spojovník a pomocí příkazu `\break` si vynutíme řádkový zlom. Samozřejmě musíme také odstranit příkaz `\parshape`. Náš kód by nyní mohl vypadat takto:

```
\noindent
\verzalky{Jiným vhodným způsobem zvý-}\break raznění začátků
...
řádku, je už tvořena malými písmeny.
```

Opakovat tento postup na začátku každé kapitoly, to by byla strašná otrava, zvláště kdybychom sázeli rozsáhlejší knihu s mnoha kapitolami. Navíc je toto řešení takříkajíc na jedno použití, neboť při změně některých parametrů sazby je zapotřebí provést tuto činnost znovu. Proto se pokusíme zvýrazňování prvního řádku odstavce zautomatizovat.

Chcete-li při sazbě knihy vyzdobit vstupní stránky jednotlivých kapitol, můžete u prvního odstavce každé kapitoly použít iniciálu, tj. výrazné počáteční písmeno. V češtině považujeme spřežku ch za jediné písmeno, a proto pokud text začíná touto spřežkou, měla by iniciála zahrnovat obě její složky.

První odstavec každé kapitoly může být sázen bez odstavcové zarážky. V takovém případě by serify iniciály měly přesahovat přes levý okraj sazebního obrazce tak, aby levý okraj obrazu iniciály opticky splýval s levým okrajem sazebního obrazce. Pozornost je třeba věnovat také tomu, aby text plynule navazoval na iniciálu.

Další možností je zapustit iniciálu do textu. Výška iniciály může být dva i více řádků. Účaří iniciály se musí krýt s účařím řádku, k němuž je iniciála zapuštěna. Horní okraj iniciály nesmí být níže než horní okraj prvního řádku. Pokud si budete chtít vytvořit makro, které vypočte vhodnou velikost písma pro sazbu iniciály, můžete se inspirovat v TEXbooku naruby na str. 81—82. Zbytek slova, jehož první písmeno je vyjádřeno iniciálou, má plynule navazovat na iniciálu; další řádek či řádky, do nichž iniciála zasahuje, se od iniciály oddělují půlčtverčíkem (tj. mezerou o velikosti 0,5 em). Pokud je iniciálou vyjádřena jednopísmenná předložka či spojka, zbytek prvního řádku se od iniciály odděluje stejnou mezerou jako ostatní řádky, do nichž iniciála zasahuje.

JINÝM VHODNÝM ZPŮSOBEM ZVÝRAZNĚNÍ začátků kapitol je sazba prvního řádku každé kapitoly verzálkami čili velkými písmeny. Jestliže je nutno slovo na konci prvního řádku rozdělit, sází se pochopitelně jeho první část z verzálek a druhá část, která přijde na začátek druhého řádku, je už tvořena malými písmeny.

JEMNĚJŠÍHO VZHLEDU DOCÍLÍTE TÍM, že použijete verzálky menšího písma. Font kombinující verzálky s kapitálkami vídáme často v situacích, kam se příliš nehodí; ani zde bych ho moc nedoporučoval. Takový font je vhodný především tam, kde by měl vyniknout rozdíl mezi malými a velkými písmeny.

VERZÁLKAMI ovšem nemusí být vysázen první řádek celý. Stačí, když z verzálek bude jen první slovo, první slovní spojení či jiná počáteční část prvního řádku. Zatímco v běžném textu by základní velikost mezislovní mezery měla odpovídat třetině čtverčíku, v sazbě z verzálek se doporučuje použít jako základní mezislovní mezeru půlčtverčík; tuto mezeru je pak podle potřeby možno stáhnout až na třetinu čtverčíku anebo rozpálit až na dvě třetiny čtverčíku. Zde je vhodné upozornit také na to, že PostScriptová písma mívají nastavenu příliš úzkou mezislovní mezeru. Proto pokud chcete při použití PostScriptových písem dosáhnout optimální kvality sazby, měli byste si v TEXu pohrát s parametry fontdimen 2, 3, 4 a 7.

HONOSNĚJŠÍM DOJMEM BUDE působit, když spojíte sazbu iniciály se sazbou prvního řádku z verzálek. Opět se nemusí jednat o celý první řádek, nýbrž postačí, když z verzálek bude první slovo či první slovní spojení.

V TOMTO ODSTAVCI je ještě jedna ukázka spojení iniciály s verzálkami. Na závěr dodejme, že zvýraznění začátku prvního odstavce jednotlivých kapitol se může uplatnit jak v krásné literatuře, tak v literatuře naučné a populárně vědné.

Makro, které si pro tento účel připravíme, načte text celého odstavce do parametru. Nejdříve vysází tento text nanečisto způsobem, jaký jsme si popsali výše. První řádek takto vzniklého odstavce bude vzorem, jak by měl vypadat první řádek výsledného odstavce.

Nyní stojíme před otázkou, jak v textu, který makro načetlo do parametru, určíme hranici mezi prvním a druhým řádkem, abychom mohli první řádek vysázet zvýrazněně a zbytek odstavce normálně. Pro tento účel využijeme informaci o šířce vysázeného textu: Změříme přirozenou šířku materiálu ze vzorového prvního řádku. Pak založíme dva zásobníky. Jeden zásobník bude reprezentovat text prvního řádku odstavce a na počátku bude prázdný; druhý zásobník bude představovat text zbytku odstavce a na počátku bude obsahovat text celého odstavce. Z tohoto zásobníku budeme postupně přesouvat jednotlivá slova do prvního zásobníku. Po každém přesunu slova vysázíme v pokusném `\hbox`u obsah prvního zásobníku a šířku tohoto `\hbox`u porovnáme s přirozenou šířkou materiálu ze vzorového prvního řádku. Budou-li se obě šířky shodovat, znamená to, že jsme nalezli kýženou hranici mezi prvním a druhým řádkem. Pakliže bude šířka pokusného `\hbox`u větší než přirozená šířka materiálu ze vzorového prvního řádku, musíme onu hranici hledat uvnitř posledního slova v prvním zásobníku. Proto toto slovo vrátíme do druhého zásobníku a do prvního zásobníku budeme nyní postupně přesouvat jednotlivá písmena. Po každém přesunu písmene vysázíme v pokusném `\hbox`u obsah prvního zásobníku (nesmíme ovšem zapomenout na konec tohoto `\hbox`u přidat spojovník) a šířku tohoto `\hbox`u porovnáme s přirozenou šířkou materiálu ze vzorového prvního řádku. Kýžená hranice mezi prvním a druhým řádkem bude známa v okamžiku, kdy se budou obě šířky shodovat.

V případě, že hranice mezi prvním a druhým řádkem probíhá uvnitř slova, mohla by naše metoda detekovat tuto hranici na nesprávném místě, pokud by kvůli splynutí písmen v ligaturu nebo kvůli záporným implicitním kernům měla skupina písmen, za níž by měla tato hranice probíhat, stejnou šířku jako počáteční část této skupiny. To nás ovšem nemusí příliš znepokojovat, neboť takový jev je velice nepravděpodobný a navíc dobrý sazeč se nespoléhá na automatické dělení slov, nýbrž pokaždé ve vysázeném dokumentu zkontroluje, zda jsou všechna slova rozdělena správně. Bohužel v dnešní době je dobrých sazečů poskrovnu, a to i mezi TEXisty.

Máme-li v zásobníku `\prvni` uložen text celého prvního řádku a v zásobníku `\zbytek` zbylý text odstavce, stačí už jen obě části textu vysázet náležitým způsobem do jednoho odstavce. Tím činnost našeho makra skončí.

Při přesouvání jednotlivých slov ze zásobníku `\zbytek` do zásobníku `\prvni` se budeme muset vypořádat s tím, že slova mohou být vzájemně oddělena mezerami různého druhu. Pozornost věnujeme jednak obyčejným mezerám, jednak řídicím mezerám. Naproti tomu nás vůbec nebudou zajímat vlnky, neboť v nich není povolen řádkový zlom. Nejdříve ze zásobníku `\zbytek` vyčleníme úsek textu, který sahá od začátku zásobníku až po první řídicí mezeru. Abychom měli jistotu,

že v zásobníku je alespoň jedna řídicí mezera, připojíme jednu řídicí mezeru na samý konec zásobníku; ukáže-li se, že je tato řídicí mezera nadbytečná, hned ji zase odstraníme. Z takto získaného úseku textu vyčleníme podúsek, který sahá od začátku úseku až po první obyčejnou mezeru. Abychom měli jistotu, že v úseku textu je alespoň jedna obyčejná mezera, připojíme jednu obyčejnou mezeru na samý konec úseku; ukáže-li se, že je tato mezera nadbytečná, hned ji zase odstraníme. Podúsek textu, který tímto způsobem získáme, je oním slovem, které máme přesunout do zásobníku `\prvni`. Zbylou část vyčleněného úseku textu vrátíme do zásobníku `\zbytek`.

Nyní si ukážeme kód celého makra, jehož činnost jsme si vysvětlili v předchozích odstavcích.

```
% deklarace registrů
\newdimen\sirka % přirozená šířka materiálu z prvního řádku
\newtoks\prvni % zásobník s textem prvního řádku
\newtoks\zbytek % zásobník s textem zbytku odstavce
\newtoks\oldprvni % předchozí obsah zásobníku \prvni
\newtoks\oldzbytek % předchozí obsah zásobníku \zbytek
\newtoks\mezera % mezera mezi textem v \prvni a \zbytek

% hlavní makro, které vysází první řádek odstavce zvýrazněně
% #1 = způsob sazby (zvýraznění) prvního řádku
% #2 = text odstavce
\def\prvniradek#1#2\par{\par
   \def\zvyrazneni{#1}
   % na zkoušku vysázíme celý odstavec zvýrazněně
   \setbox0=\vbox\bgroup
      \zvyrazneni{#2\looseness=0 \overfullrule=0pt
      % aby fungoval \hangindent, započteme jeho velikost do šířky
      % prvního řádku ve zkušebním odstavci, je-li toho zapotřebí
      \dimen0=\hsize
      \ifnum\hangafter<1
         \advance\dimen0
            by\ifdim\hangindent>0pt -\fi \hangindent
      \fi
      \parshape 2 0pt \dimen0 0pt \maxdimen \endgraf}
      \ifnum\prevgraf<2 % odstavec má pouze jediný řádek
         \egroup {\zvyrazneni{#2\looseness=0 \endgraf}}\par
      \else % odstavec má více řádků
         % rozebereme odstavec, abychom se dostali k prvnímu řádku
         \count255=\prevgraf
         \loop
         \ifnum\count255>0
```

```
            \setbox0=\lastbox \unskip \unpenalty
            \advance\count255 by-1
         \repeat
         % určíme přirozenou šířku materiálu z prvního řádku
         \setbox0=\hbox{\unhbox0}
         \global\sirka=\wd0 \egroup
         % nastavíme počáteční obsah zásobníků
         \prvni={}
         \zbytek={#2}
         % začneme přesouvat jednotlivá slova
         \expandafter \presunslovo
      \fi}

% makro, které přesouvá jednotlivá slova ze zásobníku \zbytek
% do zásobníku \prvni a zároveň porovnává šířku sazby textu ze
% zásobníku \prvni s přirozenou šířkou vzorového prvního řádku
\def\presunslovo{
   % zazálohujeme původní obsah zásobníků \prvni a \zbytek
   \oldprvni=\prvni
   \oldzbytek=\zbytek
   % přesuneme jedno slovo ze zásobníku \zbytek do zás. \prvni
   \expandafter\osetriridicimezeru\the\zbytek\ \end
   % v pokusném \hboxu vysázíme obsah zásobníku \prvni
   \setbox0=\hbox{\hskip\leftskip\indent
      \def\noindent{\setbox0=\lastbox}%
      \expandafter\zvyrazneni\expandafter{\the\prvni}%
      \hskip\rightskip}
   \ifdim\wd0=\sirka % nalezli jsme hranici mezi 1. a 2. řádkem
      \let\next=\vysazejodstavecN
   \else % hranici mezi 1. a 2. řádkem jsme ještě nenalezli
      \ifdim\wd0<\sirka
         \edef\temp{\the\zbytek}
         \ifx\temp\empty % něco je v nepořádku
            \errmessage{Chyba 1}
            \let\next=\vysazejodstavecN
         \else % přesuneme další slovo
            \edef\temp{\prvni={\the\prvni\the\mezera}}\temp
            \let\next=\presunslovo
         \fi
      \else % hranice probíhá uvnitř slova
         % obnovíme předchozí obsah zásobníků
         \prvni=\oldprvni
```

```
         \zbytek=\oldzbytek
         % nyní začneme přesouvat jednotlivá písmena
         \let\next=\presunpismeno
   \fi\fi
   \next}

% makro, které z textu oddělí první část ukončenou řídicí mezerou,
% případný zbytek textu vrátí do zásobníku \zbytek a oddělenou
% část předá ke zpracování makru \osetrimezeru
\def\osetriridicimezeru#1\ #2\end{
   \def\temp{#2}
   \ifx\temp\empty % text neobsahuje řídicí mezeru
      \zbytek={}
      \mezera={}
   \else % text obsahuje řídicí mezeru
      \smazridicimezeru#2\end
      \mezera={\ }
   \fi
   \osetrimezeru#1 \end}
\def\smazridicimezeru#1\ \end{\zbytek={#1}}

% makro, které z textu oddělí první část ukončenou mezerou,
% tuto část přidá do zásobníku \prvni a zbytek textu vrátí
% do zásobníku \zbytek
\def\osetrimezeru#1 #2\end{
   \prvni=\expandafter{\the\prvni#1}
   \def\temp{#2}
   \ifx\temp\empty
   \else % text obsahuje mezeru
      \smazmezeru#2\end
      \mezera={ }
      \def\temp{#1}
      \ifx\temp\empty
         \edef\temp{\the\prvni}
         \ifx\temp\empty
            \mezera={} % počáteční mezera bude ignorována
   \fi\fi\fi}
\def\smazmezeru#1 \end{
   \toks0={#1}
   \edef\temp{\zbytek={\the\toks0 \the\mezera\the\zbytek}}\temp}

% makro, které přesouvá jednotlivá písmena ze zásobníku \zbytek
% do zásobníku \prvni a zároveň porovnává šířku sazby textu ze
% zásobníku \prvni s přirozenou šířkou vzorového prvního řádku
```

```
\def\presunpismeno{
   % přesuneme jedno písmeno ze zásobníku \zbytek do zás. \prvni
   \expandafter\sejmipismeno\the\zbytek\end
   % v pokusném \hboxu vysázíme obsah zásobníku \prvni
   \setbox0=\hbox{\hskip\leftskip\indent
      \def\noindent{\setbox0=\lastbox}%
      \expandafter\zvyrazneni\expandafter{\the\prvni-}%
      \hskip\rightskip}
   \ifdim\wd0=\sirka % nalezli jsme hranici mezi 1. a 2. řádkem
      \let\next=\vysazejodstavecR
   \else
      \edef\temp{\the\zbytek}
      \ifx\temp\empty % něco je v nepořádku
         \errmessage{Chyba 2}
         \let\next=\vysazejodstavecR
      \else
         \ifdim\wd0<\sirka % hranici musíme hledat dále
            \let\next=\presunpismeno
         \else % něco je v nepořádku
            \errmessage{Chyba 3}
            \let\next=\vysazejodstavecR
   \fi\fi\fi
   \next}
\def\sejmipismeno#1#2\end{
   \prvni=\expandafter{\the\prvni#1}
   \zbytek={#2}}

% makra, která provedou konečnou sazbu celého odstavce
\def\vysazejodstavecN{\vysazejodstavec{}{\relax}}
\def\vysazejodstavecR{\vysazejodstavec{-}{\odstranspoj}}
\def\vysazejodstavec#1#2{\setbox0=\hbox{%
      \def\noindent{\setbox0=\lastbox}\indent
      \expandafter\zvyrazneni\expandafter{\the\prvni#1}}
   \noindent\unhbox0\break \expandafter#2\the\zbytek \par}
\def\odstranspoj{\futurelet\temp\xodstranspoj}
\def\xodstranspoj{\ifx-\temp \expandafter\removetoken \fi}
\def\removetoken#1{}
```

Povšimněme si definice `\vysazejodstavec`: Nejprve sestavíme první řádek v `\hboxu` a potom obsah tohoto `\hboxu` vypustíme do horizontálního seznamu v odstavcovém módu. Děláme to proto, abychom na konci prvního řádku neměli `\discretionary`, pokud by první řádek končil spojovníkem. V odstavcovém módu TEX automaticky připojuje prázdné `\discretionary` za každý spojovník,

kdežto ve vnitřním horizontálním módu nikoli. Toto `\discretionary` by nám dělalo neplechu při přetečeném prvním řádku, a to i v případě, že by velikost přetečení byla menší než hodnota registru `\hfuzz`, tedy i tehdy, kdy by nám nepatrné přetečení řádku samo o sobě vůbec nevadilo. Při hodnotě registru `\exhyphenpenalty` menší než 10 000 by TEX přetečený první řádek zalomil v tomto `\discretionary` a hned za ním by provedl ještě jeden zlom, vynucený příkazem `\break`, takže by se pod prvním řádkem objevil jeden prázdný řádek a pak by teprve pokračoval další text odstavce.

Příkaz `\odstranspoj` v případě potřeby odstraní přebytečný spojovník ze začátku druhého řádku. Představme si, že v německém nebo anglickém textu na konci prvního řádku je slovo, které obsahuje spojovník (např. *Drucker-Zeugnis*), rozděleno právě v místě spojovníku. Jak vyplývá z kódu našich maker, tento spojovník zůstane na začátku zásobníku `\zbytek` a na konec prvního řádku bude přidán další spojovník. V němčině a angličtině (na rozdíl od češtiny) je zvykem v takových případech psát spojovník jen na konci řádku, takže by nám zde jeden spojovník přebýval; proto musí být pomocí makra `\odstranspoj` odstraněn. V češtině je zapotřebí dělení slov obsahujících spojovník řešit komplexně v celém dokumentu; obvykle postačí mít nastaveno `\exhyphenpenalty=10000`.

Teď už nám zbývá říci si pár slov o užívání našich maker. Před odstavec, jehož první řádek má být zvýrazněn, zapíšeme příkaz `\prvniradek`. Jako první parametr tohoto příkazu uvedeme příkaz (nebo příkazy), který provede požadované zvýraznění. Nemusí se jednat pouze o zvýraznění verzálkami, také je možno nastavit jiné písmo, třeba kurzivu či tučné písmo, anebo použít verzálky menšího písma. Sázíme-li text písmem o velikosti deseti bodů, můžeme první řádek zvýraznit devítibodovými verzálkami. Nejdříve tedy zavedeme devítibodové písmo:

```
\font\ninerm=csr9
```

a potom už můžeme do zdrojového souboru zapsat celý odstavec, uvozený příkazem `\prvniradek`:

```
\prvniradek{\ninerm\verzalky}
\noindent
Jiným vhodným způsobem zvýraznění začátků
...
řádku, je už tvořena malými písmeny.
```

Všimněte si, že příkaz `\noindent` pro zahájení odstavce bez odstavcové zarážky je zapsán těsně na začátku textu až po řídicí sekvenci `\prvniradek`. Pokud chceme upravit délku odstavce pomocí `\looseness`, musíme příkaz `\looseness` zapsat až na konec odstavce. Má-li ovšem odstavec jen jediný řádek, `\looseness` nemá žádný význam, neboť v takovém případě naše makro automaticky nastaví `\looseness` na nulu. Podobně bezvýsledná by byla snaha o zkrácení dvouřádkového odstavce. Jestliže chceme upravit tvar odstavce pomocí `\hangindent`

a \hangafter, musíme tyto dva příkazy uvést také až na konci odstavce. Úprava tvaru odstavce pomocí \parshape není možná.

Hranice mezi prvním a druhým řádkem nesmí být uzavřena ve skupině. Pro sazbu uvozovek je tedy vhodné v celém odstavci používat příkazy \clqq a \crqq místo obvyklejšího příkazu \uv. Pro náročnějšího čtenáře dodejme, že makro \uv, jak je definováno v csplainu, by mohlo způsobit neplechu, i kdybychom se pomocí tohoto makra snažili dát do uvozovek část textu, která by celá zůstala na prvním řádku. Toto makro potlačuje případné implicitní kerny mezi uvozovkami a uvozovaným textem, neboť mezi uvozovky a uvozovaný text vstupuje příkaz hlavního procesoru. Tímto způsobem potlačené implicitní kerny a ligatury TEX obnovuje při druhém průchodu řádkového zlomu (obnova ovšem probíhá pouze v těch skupinách znaků, v nichž TEX vyhledává potenciální místa dělení). Ve vzorovém prvním řádku by tedy implicitní kerny mohly být obnoveny, kdežto v textu vysázeném v pokusném \hboxu nikoli. V takovém případě by naše makro nejspíše havarovalo s chybou č. 3.

Visící spojovník

V typografii se někdy využívá tzv. visící interpunkce, která spočívá v tom, že interpunkční znaménka vyčnívají z okraje sazby. Jakým způsobem je možno v TEXu udělat visící interpunkci, je dostatečně popsáno v dodatku D Knuthova TEXbooku a v kapitole 6 Olšákova TEXbooku naruby. Zde se zaměříme na jednu zajímavost, kterou ve zmíněných knihách nenaleznete.

Zvláštní problém při tvorbě visící interpunkce představuje spojovník. V originálním TEXbooku i v českém TEXbooku naruby se doporučuje pro visící spojovník použít speciální font, ve kterém by spojovník měl nulovou šířku. Pro tento účel by se nejspíše hodil virtuální font. My si ale ukážeme, že při sazbě visícího spojovníku si TEXista může vystačit s obyčejným fontem, v němž má spojovník svoji přirozenou šířku.

Důležitým předpokladem našeho řešení je, aby použitý font měl alespoň jednu pozici neobsazenu žádným znakem. Tento předpoklad je splněn jak u originálních fontů písma Computer Modern, tak u tzv. CS-fontů s československou variantou tohoto písma. Dále využijeme toho, že TEX přičítá k meziřádkové penaltě hodnotu z registru \brokenpenalty, jestliže řádek, který je bezprostředně nad meziřádkovou penaltou, končí rozděleným slovem.

Do registru \hyphenchar přiřadíme číslo libovolné pozice, která není obsazena žádným znakem; pro originální fonty písma Computer Modern i pro CS-fonty vyhovuje např. pozice "90. Protože tím vlastně dosadíme do funkce spojovacího znaménka znak, který ve skutečnosti neexistuje, nebude za rozdělená slova na konci řádku připojeno vůbec nic. Aby nás TEX nehuboval, že v použitém fontu není na dané pozici přítomen žádný znak, nastavíme \tracinglostchars

na nulu. Budeme-li mít odstavec vysázen bez spojovníků na konci řádků, podle velikosti meziřádkových penalt zjistíme, které řádky končí rozděleným slovem, a na konec takových řádků připojíme spojovník, který bude uzavřen do `\hbox`u s nulovou šířkou, takže tento spojovník bude vyčuhovat napravo ze sazebního obrazce. Mezi každé dva řádky v odstavci TEX vkládá penaltu o hodnotě `\interlinepenalty`. Mezi prvním a druhým řádkem je tato penalta zvětšena o hodnotu `\clubpenalty` a mezi předposledním a posledním řádkem je zvětšena o hodnotu `\widowpenalty`. Končí-li první ze dvojice řádků rozděleným slovem, je k této penaltě přičtena ještě hodnota `\brokenpenalty`. Abychom tedy z velikosti meziřádkové penalty mohli vyčíst, zda je na konci řádku, který této penaltě předchází, rozdělené slovo či nikoli, budeme potřebovat, aby `\brokenpenalty` nebyla rovna nule. Plain i csplain nastavují `\brokenpenalty=100`.

Pro uživatele vytvoříme příkazy `\vystrkuj` a `\nevystrkuj`. Mezi tyto dva příkazy uživatel napíše libovolný počet odstavců, v nichž bude chtít mít visící spojovníky. Makro `\vystrkuj` zahájí skupinu, nastaví registry `\hyphenchar` a `\tracinglostchars` na hodnoty, o nichž jsme se zmínili výše, dále lokálně předefinuje příkaz `\par` a pak otevře pracovní `\vbox`, označený číslem 0. Pro zajištění správné meziřádkové mezery nad jednotlivými odstavci je zapotřebí přenášet `\prevdepth` mezi vnějším vertikálním módem a pracovním `\vbox`em 0. Po sestavení odstavce v pracovním `\vbox`u 0 bude tento odstavec v cyklu `\loop` postupně odspodu rozebrán, přitom bude na konec řádků s rozděleným slovem připojen visící spojovník. Následně budou jednotlivé části odstavce opět poskládány do pracovního `\vbox`u 1. Po rozebrání a opětném složení celého odstavce bude uzavřen pracovní `\vbox` 0 a obsah `\vbox`u 1 bude vložen do vnějšího vertikálního seznamu; před tento materiál bude ještě vložena mezera o velikosti `\parskip`. Makro `\nevystrkuj`, pokud bude v pracovním `\vbox`u 0 ještě rozpracován nějaký odstavec, si vynutí jeho ukončení. Protože během toho ukončování bude pracovní `\vbox` 0 uzavřen a znovu otevřen, makro `\nevystrkuj` ho pomocí `\egroup` definitivně uzavře; potom uzavře skupinu, která byla otevřena makrem `\vystrkuj`. Tím se registr `\tracinglostchars` vrátí ke své původní hodnotě a příkaz `\par` dostane svůj původní význam. Původní hodnota registru `\hyphenchar` musí být ovšem obnovena ze zálohy, neboť přiřazení do tohoto registru je vždy globální. Nakonec je nastaveno správné `\prevdepth`.

```
\newcount\spojovnik % pro uschování hodnoty \hyphenchar
\newdimen\hloubka % pro uschování velikosti \prevdepth
\newcount\radek % číslo právě zpracovávaného řádku
\newcount\vdova % číslo předposledního řádku v odstavci

\def\vystrkuj{\par
   \spojovnik=\hyphenchar\font
   \hyphenchar\font="90 % tento znak nesmí být ve fontu přítomen
   \begingroup
```

```
   \tracinglostchars=0
   \def\par{\ifhmode\ukonciodstavec\fi}
   \global\hloubka=\prevdepth
   \setbox0=\vbox\bgroup
   \prevdepth=\hloubka}

\def\ukonciodstavec{\endgraf
   \global\hloubka=\prevdepth
   \setbox2=\lastbox
   \global\setbox1=\vbox{\box2}
   \radek=\prevgraf \advance\radek by-1
   \vdova=\radek
   \loop
   \ifnum\radek>0
      \skip2=\lastskip \unskip
      \count2=\lastpenalty \unpenalty
      \count4=\count2
      \advance\count4 by-\interlinepenalty
      \ifnum\radek=\vdova \advance\count4 by-\widowpenalty \fi
      \ifnum\radek=1 \advance\count4 by-\clubpenalty \fi
      \setbox2=\lastbox
      \ifnum\count4=0 \else
         % připojíme visící spojovník
         % a ošetříme mezeru podle \rightskip
         \setbox2=\hbox to\wd2{\unhbox2
            \skip0=\lastskip\unskip \rlap{-}\hskip\skip0} \fi
      \global\setbox1=\vbox{\box2
         \ifnum\count2=0 \else \penalty\count2 \fi
         \vskip\skip2 \unvbox1}
      \advance\radek by-1
   \repeat
   \skip2=\lastskip
   \global\setbox1=\vbox{\vskip\skip2\unvbox1}
   \egroup % ukončíme pracovní \vbox
   \vskip\parskip \unvbox1 \endgraf
   \setbox0=\vbox\bgroup
   \prevdepth=\hloubka}

\def\nevystrkuj{\par\egroup\endgroup
   \hyphenchar\font=\spojovnik
   \prevdepth=\hloubka}
```

Při výpočtu demerits řádku s rozděleným slovem TEX normálně použije hodnotu z registru `\hyphenpenalty`. V odstavcích s visícím spojovníkem je to trochu

jiné — TEX většinou použije hodnotu z registru `\exhyphenpenalty`, někdy však také hodnotu z registru `\hyphenpenalty`. Abychom dokázali rozhodnout, zda TEX v daném případě použije registr `\hyphenpenalty` nebo `\exhyphenpenalty`, musíme vědět, co přesně do sazebního materiálu vkládá algoritmus na automatické vyhledávání potenciálních míst dělení slov. Nuže, většinou na příslušné místo vkládá `\discretionary{\char\hyphenchar\font}{}{}`. Pokud se však potenciální místo dělení objeví v implicitním kernu nebo uvnitř ligatury, musí zmíněný algoritmus takový případ pečlivě ošetřit, aby kern či ligatura zůstala zachována, kdyby v daném místě k řádkovému zlomu zrovna nedošlo; když se např. v anglickém textu vyskytne slovo *difficult,* zmíněný algoritmus ho upraví tímto způsobem: `di\discretionary{f-}{fi}{ffi}\discretionary{-}{}{}cult`. Poněvadž máme při sazbě visících spojovníků nastaven registr `\hyphenchar` na neexistující znak, je příkaz `\discretionary{\char\hyphenchar\font}{}{}` stejný jako příkaz `\discretionary{}{}{}`, takže se při výpočtu demerits použije `\exhyphenpenalty`, neboť parametr ⟨*prebreak*⟩ je prázdný. Je-li však slovo rozděleno v implicitním kernu nebo uvnitř ligatury, použije se `\hyphenpenalty`.

Summary: Some tips and tricks for TEX typesetting

The article presents an original solution of three challenging typographic tasks by means of TEX's macrolanguage. The reader can find here not only a detailed description of the sophisticated macros but also a number of recommendations concerning typography.

1. In Czech typography, the interval dash cannot stand at the end of a line nor can it be moved to the beginning of the following line. If the split between lines is inevitable, the dash is replaced by a word. While writing a macro for the interval dash, it is necessary to overcome a certain disadvantage of the command `\discretionary`, namely that it does not allow to append glue to the current list, as the word which replaces the dash has to be separated from the text by a space, while the dash is not separated.

2. The setting of the whole first line of a paragraph in capitals (or possibly in a different font) can be applied at the beginning of individual chapters.

3. For hanging hyphenation, the TEXbook recommends to use a special font with zero-width `\hyphenchar`. The solution of this typographic task can, however, be achieved using an ordinary font with real-width `\hyphenchar`.

Makro na konverzi textů, PDF záložky

Petr Olšák

Občas se mi při programování maker v TEXu stane, že potřebuji překonvertovat (zhruba řečeno) „řetězec znaků na jiný řetězec znaků“. Například potřebuji z textu sundat háčky a čárky, potřebuji text převést do jiného kódování, potřebuji si v textu všímat některých speciálních znaků a ty převést na jiné znaky... Už dlouhou dobu jsem si říkal, že by nebylo marné udělat na to nějaké univerzální makro. Posledním podnětem k tvorbě tohoto makra byl požadavek pana Kubena vyřešit problém s českými texty v záložkách v PDF při použití `hyperref` a bez použití `inputenc`.

V tomto článku představím makro `cnv.tex`, které umožňuje konverzi podle tabulky, kterou může deklarovat uživatel. Konverze se vyhýbá běžné expanzi a může jí podléhat jakýkoli token jakékoli kategorie.

Motivace

Pan Kuben mě požádal, ať se pokusím vyřešit problém s nefungující češtinou v PDF záložkách při použití balíčku `hyperref` (s volbou `unicode`) v LATEXu. Vysvětlil mi, že pokud navíc použije balíček `inputenc` (tj. akcentované znaky má aktivní), pak vše funguje, jak má. Pokud ale nechce aktivizovat česká písmena s háčky a čárkami a použije jen `\usepackage{czech}` bez balíčku `inputenc`, pak texty v záložkách nefungují. Přitom v CSLATEXu bychom se rádi balíčku `inputenc` vyhnuli, protože pak máme například méně starostí při zpracování rejstříků a máme čitelnější log soubory.

Po kratším průzkumu jsem přišel věci na kloub: makro `hyperref` používá pro texty do záložek 16bitový UNICODE a snaží se text do tohoto kódování poněkud amatérským způsobem konvertovat: před každý „obyčejný znak“ přidá nulový bajt a aktivní znaky nechá expandovat na požadované dva bajty podle deklarace v souboru `puenc.def`. Je tedy nabíledni, že pokud akcentované znaky nemáme aktivní, nebude to fungovat. Protože text pro záložku je už uložen v parametru (například při činnosti LATEXového makra `\section`), kategorie jednotlivým znakům jsou už přiděleny a znaky nelze jednoduše dodatečně „aktivizovat“. Chtělo by to tedy důkladnější makro na konverzi, které nespoléhá na žádnou expanzi ani na aktivní kategorie znaků. Tak jsem se konečně pustil do psaní dlouho odkládaného makra `cnv.tex`. Na „zakázce“ pro pana Kubena jsem pak makro odladil. Nakonec jsem makro zveřejnil k volnému použití [1].

Použití makra

Použití makra `cnv.tex` se pokusím ukázat právě na konverzi českého textu do PDF záložek. V předchozím odstavci jsem nebyl úplně přesný, když jsem psal, že text v záložkách je v 16bitovém UNICODE. Přesněji tento text obsahuje vesměs oktalovou notaci jednotlivých bajtů. V nezměněném tvaru zůstávají jen běžné ASCII znaky. Takže text „Záložka“ se konvertuje na:

```
\376\377\000Z\000\341\000l\000o\001\176\000k\000a
```

Prefix `\376\377` říká PDF prohlížeči, že text záložky je v UNICODE.

Chceme-li takovou konverzi použít, pak v csplainu můžeme psát:

```
\input cnv-pu
\newcount\numlink
\def\zalozka#1{\global\advance\numlink by1
   \pdfdest num\numlink fith\relax % deklarace cíle záložky
   \def\cnvtable{pu}\cnvin{#1}%    % konverze textu pro záložku
   \pdfoutline goto num\numlink{\string\376\string\377\cnvout}}
\zalozka{Tento text bude v záložce}
```

Ukázka makra je poněkud zjednodušená: například cíl záložky se většinou hodí „vysunout“ poněkud výše, aby to pěkně vypadalo. Nicméně způsob použití makra `cnv.tex` je zde zřejmý: nejprve načteme pomocí `\input` konverzní tabulku `pu` (PdfUnicode). Pokud ještě nebylo načteno samotné makro `cnv.tex`, pak se v tomto okamžiku načte automaticky také. Makro umí pracovat s libovolným množstvím konverzních tabulek. Je tedy potřeba před vlastní konverzí deklarovat, podle jaké tabulky konverze proběhne (`\def\cnvtable{pu}`). Samozřejmě, pokud makro používáme jen k jedinému účelu a s jedinou tabulkou, stačí napsat `\def\cnvtable{pu}` někam mezi globální deklarace. Výsledek konverze máme po provedení makra `\cnvin` připraven v makru `\cnvout`. Pomocí primitivu pdfTEXu `\pdfoutline` tento konvertovaný text umístíme do záložky včetně prefixu `\376\377`.

Možnosti deklarace konverzních tabulek jsou patrné po pohledu do souboru `cnv-pu.tex`. Konverze je dvouprůchodová: v prvním průchodu probíhá expanze těch tokenů, které jsou deklarovány jako makra příkazem `\predef`. Například úsek tabulky:

```
\predef \TeX    {TeX}
\predef \uv #1{\clqq #1\crqq}
```

způsobí, že token `\TeX` se v prvním průchodu expanduje na tři tokeny `TeX` a `\uv{text}` se expanduje na `\clqq␣text\crqq`. Pointa je ale v tom, že tyto deklarace vůbec nekolidují s běžným významem deklarovaných tokenů (např. sekvence `\TeX` je v TEXu definována poněkud jinak) a navíc můžeme takto deklarovat jakýkoli token jakékoli kategorie. Takže je možné:

```
\predef X{aha}
```

což způsobí, že každý výskyt znaku X se promění na aha. K tomuto výsledku „expanze“ se TEX při prvním průchodu vrací.

V druhém průchodu probíhá finální konverze, která se deklaruje pomocí `\findef` a má stejná pravidla, jako `\predef`. Rozdíl je v tom, že `\findef` nemůže pracovat s parametry a dále k výsledku expanze se už konverzní proces nevrací a považuje ho za definitivní. Takže `\predef␣X{(X)}` způsobí chybu (nekonečná rekurze), zatímco `\findef␣X{(X)}` je v pořádku a způsobí, že každý výskyt znaku `X` bude obalen závorkami.

V souboru `cnv-pu.tex` je tedy psáno:

```
\findef a{\string\000a}
\findef b{\string\000b}
\findef c{\string\000c}
... atd.
\findef \clqq {\string\000\string\214}
\findef \crqq {\string\000\string\215}
```

K dispozici je ještě další deklarační makro `\cnvaccent`. V `cnv-pu.tex` například můžeme najít:

```
\cnvaccent  \'A{\string\000\string\301}
```

Tento zápis způsobí, že každý výskyt `\'A` nebo `\'␣A` nebo `\'{A}` se v druhém průchodu konverze promění na `\000\301`. Navíc se makro `\cnvaccent` už v době deklarace pokusí expandovat `\'{A}` a pokud je výsledkem jediný token (například Á), pak výše uvedená deklarace provede interně ještě:

```
\findef Á{\string\000\string\301}
```

kde Á je výsledek expanze `\'{A}`. Je tedy vhodné před načtením příkazu `\cnvaccent` zařídit správnou expanzi maker `\'`, `\v` atd. To je v tabulce `cnv-pu.tex` provedeno takto:

```
\ifx\fontencoding\undefined
   \csname csaccents\endcsname
\else
   \fontencoding{\encodingdefault}\selectfont
\fi
```

Je zde pamatováno nejen na csplain (provede se příkaz `\csaccents`), ale i na LATEX (provedou se příslušné příkazy z NFSS). Tabulka je čtena celá uvnitř skupiny, takže tyto změny neohrozí případné jiné nastavení uživatele. Přitom vlastní deklarace jsou globální, takže po ukončení skupiny se neztratí.

Proč je to tak uděláno? Tabulka `cnv-pu.tex` konvertuje z interního 8bitového kódování TEXu na oktalový zápis podle UNICODE, ale toto interní kódování

může být jakékoli a není o něm v tabulce explicitní zmínka. Nemusíme tedy mít desítky tabulek typu `cork-pu`, `iso2-pu` atd.

Nebudu zde podrobně rozepisovat další vlastnosti makra `cnv.tex` a možnosti deklarací konverzních tabulek. Vše je dokumentováno přímo v souboru `cnv.tex`. Stručně jen uvedu některé vlastnosti:

- Můžeme deklarovat implicitní konverzi pro nedeklarované tokeny (viz `\cnvdefault`, `\cnvadefault`).
- Je možno nastavit, zda deklarační makra pracují jako `\def`, `\edef`, `\gdef` nebo `\xdef` (viz `\predefmethod`, `\findefmethod`).
- Makro se vyrovná s konverzemi mezer i svorek jakoby se jednalo o obyčejné znaky (kdo psal nějaké složitější TEXové makro, ten ví, že to zcela obyčejné znaky nejsou).
- Lze během konverze spustit některá makra běžným způsobem a nechat je zpracovat třeba i hlavním procesorem TEXu (`\cnvexec`, `\cnvnext`, `\cnvexpand`, `\cnvexpandtext`).
- Je možné deklarovat a používat libovolné množství tabulek (`\cnvtable`).

Původní „zakázka“ souvisela s balíčkem `hyperref` v LATEXu. Nevytvořil jsem sice samotný balíček, který problém řeší, ale pokusil jsem se aspoň zveřejnit „radu“ pro LATEXové uživatele, co mají napsat na začátek svého dokumentu:

```
\usepackage{czech}
\input cnv-pu
\usepackage{hyperref}  % volba [unicode] není nutná
\def\pdfstringdef #1#2{% \pdfstringdef předefinujeme
    \bgroup \escapechar=‘\\%
    \def\cnvtable{pu}\cnvin{#2}%
    \xdef #1{\string\376\string\377\cnvout}\egroup }
```

Vidíme, že je zde předefinováno interní makro `\pdfstringdef` použitého balíčku `hyperref`. Proto je samozřejmě nutné příkaz `\def\pdfstringdef` umístit až po volání balíčku `hyperref`. Dále je potřeba provést `\input cnv-pu` až po `\usepackage{czech}`, aby bylo správně nastaveno makro `\encodingdefault` a deklarace `\cnvaccent` proběhly podle zvoleného kódování uživatele.

Pan Kuben se pokusil navázat spojení s autorem balíčku `hyperref` a požádal ho, aby tam využití makra `cnv.tex` zapracoval například jako další volbu balíčku `hyperref`. Zdá se, že tato iniciativa vyšla naprázdno.

Rozčarování

Tušil jsem, že převod záložek do UNICODE výše popsaným způsobem nebude všelék na všechny neduhy a problémy s češtinou v záložkách. Tušení mě, bohužel,

nezklamalo. Existence neportabilních záložek v Portable Document Formátu zůstává a zůstane nadále, protože v tomto případě vše závisí na prohlížeči a hlavně na použitém systémovém fontu pro záložky, který nemůžeme jako autoři dokumentu ovlivnit.

Panu Kubenovi to zřejmě na jeho instalaci fungovalo (v Acroreaderu 6 a 7 ve W2000 a WinXP). Mně ale Acroreader zobrazil češtinu v záložkách i při UTF8 kódování chybně. V Linuxovém Acroreaderu 5 (ve Slackware 8 i 9) se mi místo všech akcentovaných znaků objevily jen puntíky. V takovém případě je čitelnější text, při kterém nechám v záložkách kódování ISO-8859-2, protože pak se mi zmrví jen znaky, které se v ISO-2 liší od ISO-1. Zkusil jsem také Linuxový Acroreader 7. Tam byly české znaky v UNICODE záložkách zobrazeny správně, ale ne všechny. Místo „Č“ jsem viděl takový podivný čtvereček a místo uvozovek taky. Zato „ž“ se zobrazovalo správně. Prohlížeč má možnost měnit velikost písma v záložkách. Při této změně začnou (podle očekávání) zlobit zase jiná písmena ve stejných textech. Budeme se tedy asi muset smířit s tím, že zobrazování češtiny v záložkách při použití Acroreaderu je nekonečný problém. Myslet si, že když já to vidím ve svém PDF prohlížeči dobře, že to tak uvidí celý svět, je poněkud naivní. Pan Kuben generuje PDF soubory s českými texty v záložkách asi pro svou soukromou potřebu nebo s vědomím, že to není dobře čitelné všude. Ani jemu to nefunguje všude, například píše, že v Acroreaderu 4 a 5 v OS/2 a eCS je to hodně špatné.

Na základě těchto zkušeností jsem se rozhodl, že zatím nebudu nově generovat například dokument `tbn.pdf` tak, aby měl texty záložek v UNICODE. Doba k tomu ještě zdaleka nenazrála. Nechám své PDF dokumenty tak, jak jsou (v ISO-8859-2), s vědomím, že ne všichni vidí texty záložek dobře.

Další možnosti

Koverze textů PDF záložek je jen jedna z možností využití makra `cnv.tex`. Tušíme, že díky nastavovatelným konverzním tabulkám můžeme makro použít pro mnoho jiných účelů. Bohužel ale makro pracuje jen s jednotlivými znaky, které v prvním průchodu mohou expandovat a v druhém průchodu se mohou proměnit na konečný výsledek. Napadlo mě vytvořit nadstavbu nad tímto makrem, které si všímá nejen jednotlivých znaků, ale celých úseků znaků (slov) a tyto úseky mění podle požadavku uživatele na případně jiná slova. Tak vzniklo makro `cnv-word.tex`.

Konverzi slov zajišťuje deklarační makro `\stringdef`, které se používá následovně:

```
\stringdef {slovo}   {nono}
\stringdef {slož}    {odhod}
\stringdef {sl}      {SL}
```

```
\wconvert {Vysloužilý voják složil zbraně a neřekl ani slovo.}
```

V makru `\cnvout` pak máme: „VySLoužilý voják odhodil zbraně a neřekl ani nono.“.

Na pořadí deklarací záleží. Kdybychom použili `\stringdef␣{sl}␣{SL}` jako první, pak by se na další slova začínající na `sl` už nedostalo a výsledek by byl následující: „VySLoužilý voják SLožil zbraně a neřekl ani SLovo.“.

Jak je toto makro uděláno je podrobně popsáno v souboru `cnv-word.tex`. Rád uvítám další náměty na vylepšení. Bohužel, možnosti regulárních výrazů pomocí makrojazyka TEXu asi nenaprogramujeme. Na určité věci je asi potřeba použít jiný nástroj.

Odkaz

[1] `ftp://math.feld.cvut.cz/pub/olsak/makra/*`:
`cnv.tex` – vlastní makro včetně dokumentace,
`cnv-pu.tex` – konverzní tabulka pro PDF záložky,
`cnv-word.tex` – experiment s konverzí celých slov.

Summary: Macro for conversion of texts, PDF outlines

A TEX macro programmer sometimes needs to convert a string of tokens to another string of tokens with defined rules. For example, we need to remove accents from a Czech text or to recode this text to another encoding or to transform some special characters to something else or... For this purpose, the `cnv.tex` macro was designed. Last but not least motivation was the problem of my TEX colleague Jaromír Kuben: he needed to convert Czech text to PDF outlines (conversion to UNICODE) using `hyperref.sty` but without activating Czech characters by `inputenc.sty`.

This article presents the `cnv.tex` macro for string conversion by a user defined table. The conversion process is out of normal expansion and each token (independent on category code) can be converted to a single token or a group of tokens. The macro is available including documentation on `ftp://math.feld.cvut.cz/pub/olsak/macros/` in files `cnv.tex`, `cnv-pu.tex`, and `cnv-word.tex`.

Redakční poznámka

Regulární výrazy jsou velmi mocným nástrojem a uživatelům TEXu (nebo alespoň tvůrcům makrobalíků) často scházejí k efektivní práci. Proto současný vývojářský tým pdfTEXu zakomponoval od verze 1.30 POSIX regex knihovnu. Zatím

jde o experimentální kód, jehož uživatelské rozhraní se ještě může měnit. Cituji za základního použití:

```
\pdfmatch [icase] [subcount <number>}] {<pattern>}{<string>}
     It returns the same values as \pdfstrcmp, but with the
     following semantics:
       -1: error case (invalid pattern, ...)
        0: no match
        1: match found
     Options:
     * icase: case insensitive matching
     * subcount: it sets the table size for found subpatterns.
       A number -1 resets the table size to the start default.
\pdflastmatch <number>
     The result of \pdfmatch is stored in an array.
     The entry 0  contains the match, the following entries
     submatches. The positions of the matches are also available.
     They are encoded:
       <position> -> <match string>
     The position -1 with an empty string indicates
     that this entry is not set.
```

Zároveň se testuje verze 1.40, do níž je integrován jednoduchý, ale plnohodnotný skriptovací jazyk lua (http://www.lua.org). Rozhraní může číst a zapisovat do registrů TEXu. TEXtak získá regulární výrazy, aritmetiku v plovoucí desetinné čárce, snazší programování cyklů ap.

Vít Zýka

TrueType fonty, TEX a čeština

Ondřej Jakubčík

Uživatelé TEXu používají v drtivé většině fonty ve formátu Adobe Type 1, případně fonty v METAFONTu. Bohužel, ne vždy je ten správný font v jednom z těchto formátů k dispozici. TrueType fonty nabízejí poměrně zajímavou alternativu k těmto zavedeným formátům.

Úvod

Přestože TEX není zdaleka jen záležitostí Unixových systémů, používá i ve Windows hlavně Postscriptové fonty. Jejich výhodou je, že každá laserová tiskárna dokáže Type 1 fonty vykreslit, a tím je výsledný Postscriptový soubor velmi snadno přenositelný. Nevýhodou je, že firma Adobe nebyla ve svém Adobe Standard Encoding k češtině právě štědrá, a pokud některé fonty obsahují i akcentovaná písmena, pak jsou to většinou kompozity, někdy i generované automaticky (typickým příkladem jsou potom písmena vypadající asi takto: t̆, d̆, l̆).

Program `a2ac` do značné míry kompenzuje tuto nevýhodu, ovšem uživatelé používající stále rozšířenější formát PDF se musí smířit s tím, že z jejich dokumentů nepůjde kopírovat text. To v případě dokumentů určených jen pro tisk nemusí být na škodu, ale v případě, že je s nimi třeba dále pracovat, jsou akcenty separované od písmen samotných poměrně nepraktické. I přes značnou konfigurovatelnost programu `a2ac` přeci jen není nad to mít odpovídající akcentovaná písmena spíše než kompozity. Toto je obzvláště důležité pro specifická česká (a slovenská) písmena ť, ď a ľ, neboť u těchto písmen se prostě opravdová klička dá nahradit apostrofem jen velmi obtížně.

S rozmachem TrueType a OpenType fontů se postupně od Type 1 fontů upouští, a některé fonty v tomto formátu již nejsou ani v katalozích. Pro TEXovou komunitu by bylo nebezpečné, kdyby zaspala a nevyužila možností, které TrueType a OpenType fonty nabízejí.

Abych nemohl být nařčen z toho, že níže popsané metody jsou již známé (viz např. [5], [6], [7]), uvedu, že jsou specifické pro kódování Cork, které trpí podobnými neduhy jako Adobe Standard Encoding. V Čechách se mi kódování XL2, popřípadě XT2 jeví mnohem vhodnější.

Do nedávné doby byl pojem TrueType spojen s Windows. Fonty ve Windows měly jisté neduhy, co se týče ligatur, kerningu a akcentovaných písmen obzvlášť. Pokud vím, není mnoho programů pro Windows, které by z písem ligatury a kerning využívaly (snad až na sázecí programy jako je PageMaker, QuarkXPress a InDesign), proto některé fonty vůbec tyto informace neobsahují.

Ovšem pod vlivem TEXu a fontu Computer Modern jsem došel k názoru, že teprve správný kerning dělá font fontem, a že sebekrásnější písmo bez odpovídající metriky je v podstatě pro vážnější práci nepoužitelné.

Dostupná písma

Uživatel Windows 2000/XP má v systému nainstalována písma od firem Monotype a Linotype, která může použít. Protože systém Windows nepoužívám, jal jsem se hledat jinde. Na webových stránkách firmy Artifex [1] jsou volně ke stažení fonty od společnosti URW++. Tyto fonty obsahují všechna akcentovaná písmena (alespoň pro češtinu), ligatury a dostatek kerningových párů. Nemohu se nezmínit také o písmu Lido ze Střešovické písmolijny [2], které je nyní dostupné ve formátu jak TrueType, tak OpenType.

Unicode

TrueType fonty se vyznačují mimo jiné tím, že každý znak má svoje označení v Unicode. To je výhodné, neboť fonty nemusí nést tabulku Postscriptových jmen (a také ji někdy nenesou).

Předpoklady

Pro úspěšné provedení níže popsané procedury je základním předpokladem aktuální distribuce TEXu a nějaký Unixový systém. Já jsem použil Linux a teTEX 3.0. Popsané postupy každopádně vyžadují pdfTEX 1.21a. Příkazy jsou psané v `bash`i, což je, jak doufám, standardní Unix–kompatibilní shell.

Postup asi ve Windows použitelný nebude, možná bude fungovat, pokud máte nainstalované prostředí Cygwin.

Příprava

Nejprve budeme potřebovat kódovací soubory a skript ze stránky [3]. Také budeme potřebovat nějaké TrueTypové fonty, jako příklad použijeme právě fonty od URW. Kódovací soubory uložíme do adresáře, který TEX (resp. dvips a pdftex) prohledává, a aktualizujeme databázi příkazem `mktexlsr`. Skript `uni-names.sed` uložíme např. do `/usr/local/bin`.

Příprava adresářů

V dočasném adresáři vytvoříme adresáře `afm`, `tfm`, `truetype`, a `vf`. V každém z nich vytvoříme složky `<písmolijna>/<rodina>`. V našem případě použijeme písmo URW Garamond No. 8, takže v Linuxu napíšeme například:

```
cd /tmp
mkdir -p {afm,tfm,truetype,vf}/urw/garamond
```

Do adresáře `/tmp` rozbalíme soubor `urwfonts-1.41.tar.bz2` a zkopírujeme soubory do odpovídajícího adresáře a pojmenujeme je trochu výhodněji:

```
tar xvfj /<cesta>/urwfonts-1.41.tar.bz2
cp urwfonts-1.41/GaramondNo8-Reg.ttf \
   truetype/urw/garamond/ugar.ttf
cp urwfonts-1.41/GaramondNo8-Ita.ttf \
   truetype/urw/garamond/ugari.ttf
cp urwfonts-1.41/GaramondNo8-Med.ttf \
   truetype/urw/garamond/ugam.ttf
cp urwfonts-1.41/GaramondNo8-MedIta.ttf \
   truetype/urw/garamond/ugami.ttf
```

Příprava metrik AFM

Nyní je na čase vygenerovat metriky fontu ve formátu AFM. V TEXu je na to utilita `ttf2afm`, kterou použijeme:

```
for i in uga{r,m}{,i}
do
  ttf2afm -o afm/urw/garamond/$i.afm truetype/urw/garamond/$i.ttf
done
```

Pokud se neobjevilo žádné varování, pak je vše v pořádku a můžeme postoupit k dalšímu kroku. Pokud si však program `ttf2afm` postěžuje

```
Warning: ttf2afm (file font.ttf): no names available in the
'post' table, print glyph names as indices
```

pak je nutné příkaz upravit na:

```
for i in uga{r,m}{,i}
do
  ttf2afm -u truetype/urw/garamond/$i.ttf | \
  uni-names.sed > afm/urw/garamond/$i.afm
done
```

Příprava metrik TFM

Aby byl TEX schopen s fonty pracovat, potřebuje k nim mít metriku TFM. Pro TrueType fonty je dále nezbytné mít i virtuální font, neboť ten obsahuje ligační tabulky.

Pro převod metrik AFM na metriky TFM slouží program `afm2tfm`. Pomocí následujícího příkazu převedeme metriky na soubory TFM a VF:

```
for i in uga{r,m}{,i}
do
  afm2tfm afm/urw/garamond/$i.afm -T ntt-xl2.enc -v ${i}8z.vpl \
  l2$i.tfm >> garamond.map
  vptovf ${i}8z.vpl
  mv ${i}8z.vf vf/urw/garamond
  mv ${i}8z.tfm tfm/urw/garamond
  mv l2${i}.tfm tfm/urw/garamond
  rm ${i}8z.vpl
done
```

Nyní, když máme virtuální fonty a metriky normálních fontů, může se stát, že požadujeme ještě například malé kapitálky. Ty vygenerujeme takto:

```
afm2tfm afm/urw/garamond/ugar.afm -T ntt-xl2.enc -c 0.67 \
  -V ugarc8z.vpl l2ugar.tfm
afm2tfm afm/urw/garamond/ugam.afm -T ntt-xl2.enc -c 0.73 \
  -V ugamc8z.vpl l2ugam.tfm
vptovf ugarc8z.vpl
vptovf ugamc8z.vpl
mv *.vf vf/urw/garamond
mv uga*.tfm tfm/urw/garamond
rm l2ugam.tfm l2ugar.tfm
rm ugarc8z.vpl ugamc8z.vpl
```

Konstanty jsou zjištěny spočítáním poměru nejvyššího velkého písmena a nejvyššího malého písmena ve skutečném kapitálkovém fontu. Obrázek, ze kterého je možno toto spočítat, je dostupný např. na `www.myfonts.com`, kde vybereme skutečný kapitálkový font, zadáme zkušební text Ix (velké I a malé x) a v libovolném bitmapovém editoru změříme výšky těchto písmen a z nich poměr vypočítáme.

Úprava map

Než budeme moci písma použít, musíme upravit soubor `garamond.map` a přidat na každý řádek odpovídající jméno TrueTypového souboru, tj. změnit řádek

```
l2ugar GaramondNo8-Reg " XL2encoding ReEncodeFont " \
```

```
<ntt-xl2.enc
```
na
```
l2ugar GaramondNo8-Reg " XL2encoding ReEncodeFont " \
<ntt-xl2.enc <ugar.ttf
```
případně, pokud jsme použili skript `uni-names.sed`,
```
l2ugar GaramondNo8-Reg " XL2encoding ReEncodeFont " \
<tt-xl2.enc <ugar.ttf
```
Řádky jsou zde rozdělené z důvodu formátování, ale správně mají být vcelku.

Finální úpravy

Nyní přesuneme adresáře `afm`, `tfm`, `truetype`, `vf` a soubor `garamond.map` do adresářové struktury TEXu (obvykle `<něco>/texmf/fonts`) a zajistíme, aby pdftex načetl soubor `garamond.map`, takže např. v teTEXu přidáme řádek
```
Map garamond.map
```
na konec souboru `updmap.cfg` a aktualizujeme jmennou databázi příkazem `mktexlsr` a aktualizujeme mapy příkazem `updmap`. Tím máme písma nainstalovaná.

Formáty písem

Pro srovnání bych rád uvedl několik faktů o čtyřech formátech fontů, které připadají v úvahu pro použití v TEXu. Existují ještě jiné formáty, které se ale v Čechách nepoužívají, ať již proto, že jsou pro češtinu zbytečné (jako třeba CID fonty s čínštinou), nebo se jedná o interní technologie firem (např. IntelliFont).

METAFONTové fonty jsou známy každému, kdo používá TEX. U rodiny Computer Modern byla poprvé prezentována myšlenka generování široké škály fontů, lišících se od sebou šířkou, sklonem, výškou atp. Tato myšlenka byla v jistém slova smyslu znovuobjevena v Multiple Master fontech od firmy Adobe [12], které ovšem již také nejsou oficiálně podporovány. Výhodou je, že libovolný matematický výraz může být použit jako vstup pro vykreslení. Ve zdrojových souborech jsou rovnou uvedeny informace o kerningu, ligaturách, akcentech a kódování.

Adobe Type 1 [10] je formát vyvinutý pro Postscriptové tiskárny. Je podporován všemi tiskárnami, které splňují standard PostScript Level 1. Jsou popsány Bézierovými křivkami, částmi kruhových oblouků a úsečkami. Podporují hinting, který umožňuje vykreslit fonty při nižším rozlišení tak, aby vypadaly přijatelně. Vyznačují se tím, že metriky jsou samostatně v souborech AFM nebo PFM.

TrueType [15] fonty byly vyvinuty firmou Apple ve spolupráci s firmou Microsoft. Pro popis znaků používá Bézierovy křivky a úsečky. Používá pro hinting specifické instrukce, které jsou patentovány firmou Apple Computer, Inc. Metriky jsou součástí souboru s fontem.

Zvláštní odnoží písem TrueType je písmo *Type 42* [18], které obsahuje celý soubor s fontem TrueType ve speciální Postscriptové obálce, která umožňuje načíst tento font do Postscriptové tiskárny. Bohužel podpora fontů Type 42 je součástí teprve poměrně nových tiskáren, takže u dokumentů není zaručena stoprocentní přenositelnost.

Několikrát jsem zmínil formát *OpenType* [16]. Je to výsledek společné práce firmy Adobe a Microsoft, a jako takový spojuje výhody TrueType fontů a Type 1 fontů. Může obsahovat buď TrueTypový font nebo font Type 2 či CFF. *CFF* [14] je způsob, který umožňuje uložit více fontů Type 1 do jednoho souboru efektivnějším způsobem, *Type 2* [13] zase umožňuje kompaktnější popis fontu. Výsledkem je formát, který umožňuje snadnou konverzi z předešlých dvou formátu do jednoho univerzálního. Metriky jsou opět součástí souboru s fontem.

Podpora ze strany TEXu

TEX podporuje všechna písma, ke kterým je schopen načíst metriky ve formátu TFM. Když získáme metriky z písem (u formátu OpenType je toto zatím složitější, podpora je připravována pro projekt Ω [17], ale neměl jsem tu možnost jej důkladně vyzkoušet), dalším problémem je přidání těchto fontů do dokumentu.

Program `dvips` podporuje přidání (i částečné) fontů Type 1 a Type 42. Pro TrueType a OpenType zatím podpora není.

O něco lépe je na tom pdfTEX, který umožňuje přidat do dokumentu jak Type 1 a Type 42, tak přímo TrueTypový soubor. Tyto fonty umí přidat i částečně. Pokud chceme použít font OpenType, pak musíme fontový soubor přidat celý, což neúměrně zvětšuje soubor PDF. Úplné vložení fontu do dokumentu může způsobit i právní problém, neboť některé písmolijny nepovolují úplně vložení (tvrdí, že soubor s fontem je možné z PDF souboru extrahovat). Pro zájemce o použití fontů OpenType jsou potřebné nástroje k dispozici na CTANu [19].

Pár slov závěrem

V ukázkách byl použit font URW Garamond No. 8, nicméně můžeme použít jakýkoli font. Uživatelům Linuxu bych doporučil zaměřit se například na fonty Luxi, které jsou dodávány společně s X.Org [4]. Soubory `ntt-xl2.enc`, `ntt-xt2.enc`, `tt-xl2.enc` a `tt-xt2.enc` jsou upravené soubory `xl2.enc` a `xt2.enc` pana Zdeňka Wagnera. Od originálu se liší absencí znaku ȷ (dotlessj), který, bohužel, Unicode neobsahuje, a taktéž znak Togonek byl nahrazen znakem Tcedilla. Soubory `ntt*` obsahují normální Postscriptová jména, soubory `tt*` obsahují znaky ve formátu `uni0000`, které je možno použít ve stavu nouze, nebo pokud nechceme generovat malé kapitálky a upravovat metriky skriptem. Nevylučuji použití ani

pod Windows, kde se za normálních okolností program `sed` nevyskytuje, a tudíž by skript `uni-names.sed` nefungoval.

Právní doslov

V textu byly použity pojmy, které mohou být registrovanými, obchodními nebo jinými značkami či známkami.

Reference

[1] `http://www.artifex.com/downloads/`
[2] `http://www.pismolijna.cz`
[3] `zatím nic`
[4] `http://www.x.org`
[5] `http://www.radamir.com/tex/ttf-tex.htm`
[6] `http://www.csit.fsu.edu/ mimi/tex/doc/guides/truetype/`
[7] `http://www.tug.org/tex-archive/info/TrueType/`
[8] `http://ipe.compgeom.org/pdftex.html`
[9] Adobe Systems, Inc.: *PostScript language reference manual*, 3rd edition, Addison–Wesley Publishing, 1999
[10] Adobe Systems, Inc.: *Adobe Type 1 Font Format*, 3rd printing, Addison–Wesley Publishing, 1993
[11] Adobe Systems, Inc.: *Adobe Font Metrics File Format Specification*, version 4.1, `http://partners.adobe.com/`
[12] Adobe Systems, Inc.: *Designing Multiple Master Typefaces*, `http://partners.adobe.com/`, 1995–1997
[13] Adobe Systems, Inc.: *The Type 2 Charstring Format*, Technical Note #5177, `http://partners.adobe.com/`, 2000
[14] Adobe Systems, Inc.: *The Compact Font Format Specification*, Technical Note #5176, `http://partners.adobe.com/`, 2003
[15] Microsoft Corp.: *TrueType 1.0 Font Files Technical Specification*, revision 1.66, `http://www.microsoft.com/typography`, 1995
[16] Microsoft Corp.: *OpenType specification*, version 1.4, `http://www.microsoft.com/typography`, 2002
[17] Mehta, A., Bella, G., Haralambous, Y.: *Adapting Ω to OpenType Fonts*, TUGBoat, Volume 24, 2003
[18] Adobe Systems, Inc.: *The Type 42 Font Format Specification*, Technical Note #5012, `http://partners.adobe.com/`, 1998
[19] `CTAN: /fonts/utilities/fontools`

Summary: TrueType fonts, TEX and Czech language

TrueType and OpenType fonts have several advantages compared to the well-known Type 1 fonts, which have been used for years by TEX users. In this article a way how to use TrueType fonts with TEX for typesetting in Czech and Slovak language is presented. While instructions are specific for encoding XL2 and XT2 (which are compatible with ISO 8859-2), the encoding vectors can be easily modified to support other encodings, like Cork.

Ondřej Jakubčík
e-mail: ojakubcik@seznam.cz

Zpracování pomocných TEXových souborů pomocí XSLT 2.0

Zdeněk Wagner

Článek nastiňuje možnosti zpracování pomocných TEXových souborů procesorem XSLT 2.0. Koncept je demonstrován reimplementací programu MakeIndex. Autor se též zamýšlí nad možností reimplementace BibTEXu pouze pomocí XSLT.

1. Úvod

V moderních operačních systémech se začíná prosazovat jako standard kódování Unicode, resp. UTF-8. V programech se tak projevuje obdoba problému Y2K. Zatímco dosavadní kódování vystačila pro téměř každý jazyk s osmibitovými znaky, nyní potřebujeme šestnáctibitové. Úprava programů však není zcela triviální. Kódování UTF-8 umožňuje úsporu místa na disku tím, že znaky US abecedy zabírají pouze jeden bajt, zatímco akcentované znaky a znaky východoasijských jazyků jsou vícebajtové. Navíc může být program provázán s dalšími pomocnými soubory, např. s fonty. Programátor tak musí řešit komplexní problém.

Stejná potíž postihla též TEX a jeho podpůrné programy. Zpracování vstupu v UTF-8 lze v TEXu řešit různými metodami, z nichž nejlepší je encTEX [1], neboť z hlediska programátora maker se znak stále chová jako znak. Makro nikdy nevidí polovinu znaku. Některé úlohy se však řeší pomocí externích programů a některé z nich ještě na zpracování UTF-8 upraveny nejsou.

UTF-8 musí být podporováno ve všech programech, jež splňují standardy XML. Nabízí se tedy možnost, že pomocné soubory budou překonvertovány do XML, zpracovýny nějakým vhodným nástrojem a opět převedeny do formátu, který TEX umí načíst. Tuto úlohu nám usnadňuje poměrně nový standard XSLT 2.0 [2], který obsahuje funkci pro zpracování obyčejných textových souborů. Tento jazyk implementuje XPath 2.0 [3]. Ukázku takového zpracování předvedl Michael Kay na konferenci XML Prague 2005 [4].

2. Reimplementace programu MakeIndex pomocí XSLT 2.0

Cílem této ukázky není vytvoření programu s naprosto stejnou funkcí. Chyby vstupu mohou být vyřešeny jiným způsobem, ale na druhé straně bude zde vytvořený program obohacen o několik nových vlastností. Systém byl testován s procesorem XSLT Saxon-B 8.5 [5] v operačním systému OS/2 Warp 4 s Javou 1.4.1 od GoldenCode [6] a 1.4.2_05 od firmy Innotek [7]. Zde popisované transformační styly jsou volně dostupné [8]. Nebudeme proto uvádět kompletní kód. Budeme rekonstruovat postup jejich vývoje a vysvětlíme si vybrané zajímavosti.

V ukázkách budeme používat prefixy několika jmenných prostorů. Především `xsl` je jmenným prostorem XSLT, jenž je svázán s URI `http://www.w3.org/1999/XSL/Transform`. Prefixem `xs` označíme jmenný prostor XML Schema, jehož URI je `http://www.w3.org/2001/XMLSchema`. Nakonec budeme vytvářet vlastní funkci, která musí mít neprázdné URI jmenného prostoru. URI může být zcela libovolné. Zde použijeme `http://icebearsoft.euweb.cz` a svážeme je s prefixem `zw`.

2.1. Načtení TEXového souboru

Po prvním zpracování vstupu LATEXem vznikne soubor s příponou `idx`. Tento pomocný soubor je vstupem pro MakeIndex. My jej ovšem budeme načítat procesorem XSLT. K tomu je určena funkce `unparsed-text()`. Jejím prvním parametrem je URI souboru, jejž chceme načíst. Předpokládá se, že soubor je uložen v kódování UTF-8. Má-li soubor jiné kódování, zadáme jej v druhém parametru. Chceme uživatelům umožnit, aby si kódování mohli zadat sami, přičemž UTF-8 ponecháme jako default. V transformačním stylu `parse-index.xsl` proto nadefinujeme parametr:

```
<xsl:param name='enc' as='xs:NMTOKEN'>utf-8</xsl:param>
```

Celý soubor pak lze vložit do jedné řetězcové proměnné příkazem:

```
<xsl:variable name='idx-content' as='xs:string'
              select='unparsed-text($index, $enc)'/>
```

přičemž `$index` je jméno požadovaného souboru. My to však uděláme trošku jinak. MakeIndex je schopen vzít vstup z několika souborů. Tuto vlastnost lze implementovat velmi snadno. Nadefinujeme si parametry transformačního stylu:

```
<xsl:param name='index' as='xs:string' required='yes'/>
<xsl:param name='file-separator' as='xs:string',</xsl:param>
```

Jména souborů, zadaná ve tvaru `index=seznam`, nyní rozdělíme použitím funkce

```
tokenize($index, $file-separator)
```

Získáme tím posloupnost tokenů, z nichž každý reprezentuje URI jednoho souboru. Načteme je tedy v cyklu:

```
for $fn in (tokenize($index, $file-separator))
    return unparsed-text($fn, $enc)
```

Na závěr vše slepíme dohromady funkcí `string-join()`. Obsah všech vstupních souborů v jednom řetězci však není to, co by se nám pro další zpracování hodilo. Víme, že jednotlivé položky jsou zapsány pomocí maker, obvykle `\indexentry`. Využijeme tedy další funkce, již nám XSLT 2.0 nabízí, a to zpracování řetězce pomocí regulárních výrazů v perlovské syntaxi. Každé makro s parametry tedy převedeme na element uložený pouze v paměti procesoru XSLT. Kostra příkazu vypadá takto:

```
<xsl:variable name='index-items' as='item()*'>
  <xsl:analyze-string
       select='string-join(for $fn in
               (tokenize($index, $file-separator))
               return unparsed-text($fn, $enc), "")'
     regex='\\(\S+?)\s*\{{(.+?)\}}\{{(-?\d+?)\}}\r?\n' flags='s'>
    <xsl:matching-substring>
      <xsl:variable name='entry' as='xs:string'
           select='regex-group(1)'/>
      <xsl:variable name='page' as='xs:integer'
           select='xs:integer(regex-group(3))'/>
      ... <!-- Kód pro analýzu argumentů -->
    </xsl:matching-substring>
    <xsl:non-matching-substring>
      <xsl:message>
The input file contains illegal entry:
<xsl:value-of select='normalize-space(.)'/>
      </xsl:message>
    </xsl:non-matching-substring>
  </xsl:analyze-string>
</xsl:variable>
```

Položky musí vyhovovat zadanému regulárnímu výrazu. Nevyhovující řetězce zapomeneme a na terminál zobrazíme chybovou zprávu.

Všimněte si, že se vůbec nestaráme o jméno makra, v němž je položka uložena, pouze si toto jméno uložíme. Uschováme si též číslo stránky. Vlastní položka může mít komplikovanější syntaxi. Text proto znovu rozebereme pomocí regulárních výrazů. MakeIndex umožňuje definovat různé druhy oddělovačů ve stylovém souboru. My je nadefinujeme pomocí parametrů transformačního stylu, jimž ponecháme analogická jména a totožné defaultní hodnoty. Některé parametry budou využívány uvnitř regulárních výrazů jako *attribute value template*. Speciální znaky proto musí být uvozeny zpětným lomítkem:

```
<xsl:param name='encap' as='xs:string'>\|</xsl:param>
<xsl:param name='actual' as='xs:string'>@</xsl:param>
<xsl:param name='escape' as='xs:string'>\\</xsl:param>
<xsl:param name='level' as='xs:string'>!</xsl:param>
<xsl:param name='quote' as='xs:string'>"</xsl:param>
```

Další znaky budou užity jen v textových řetězcích, takže zpětné lomítko se použít nesmí:

```
<xsl:param name='range-open' as='xs:string'>(</xsl:param>
<xsl:param name='range-close' as='xs:string'>)</xsl:param>
```

Vlastní text položky zpracujeme tímto kódem:

```
<xsl:analyze-string select='normalize-space(regex-group(2))'
     regex='(.+[^{$escape}{$quote}]){$encap}\s*(.+)'>
  <xsl:matching-substring>
    <xsl:call-template name='make-index-item'>
      <xsl:with-param name='entry' tunnel='yes' select='$entry'/>
      <xsl:with-param name='page' tunnel='yes' select='$page'/>
      <xsl:with-param name='style' tunnel='yes'
           select='regex-group(2)'/>
      <xsl:with-param name='text' select='regex-group(1)'/>
    </xsl:call-template>
  </xsl:matching-substring>
  <xsl:non-matching-substring>
    <xsl:call-template name='make-index-item'>
      <xsl:with-param name='entry' tunnel='yes' select='$entry'/>
      <xsl:with-param name='page' tunnel='yes' select='$page'/>
      <xsl:with-param name='text' select='.'/>
    </xsl:call-template>
  </xsl:non-matching-substring>
</xsl:analyze-string>
```

V kódu normalizujeme mezery, takže `brouk␣␣␣Pytlík` bude zkomprimován do formy `brouk␣Pytlík`. Odstraní se i případné mezery na začátku a konci řetězce a řádkové zlomy se též nahradí mezerou. Regulární výraz odpovídá po expanzi proměnných (v perlovském zápisu) `/(.+[^\\"])\|\s*(.+)/`. Tímto výrazem oddělíme text položky od označení stylu zobrazení stránkové číslice. Při předávání parametrů šabloně `make-index-item` nastavíme stribut `tunnel='yes'`. Šablonu totiž budeme volat rekurzivně. Její kostra vypadá takto:

```
<xsl:template name='make-index-item'>
  <xsl:param name='entry' as='xs:string' required='yes'
                          tunnel='yes'/>
  <xsl:param name='page' as='xs:integer' required='yes'
                         tunnel='yes'/>
  <xsl:param name='style' as='xs:string' tunnel='yes'
       select='$default-page-style'/>
  <xsl:param name='text' as='xs:string' required='yes'/>
  <xsl:element name='{$entry}'>
    <xsl:analyze-string select='$text'
         regex='(.+?[^{$escape}{$quote}]){$level}\s*(.+)'>
    ...
    </xsl:analyze-string>
  </xsl:element>
</xsl:template>
```

Účelem této šablony a šablon z ní volaných je oddělení víceúrovňových rejstříkových položek. Proto zpracujeme text rekurzivně, přičemž na rozdíl od programu MakeIndex není zde hloubka vnoření omezena. Atribut `tunnel='yes'` způsobí, že parametry se stejnou hodnotou „protunelují“ do rekurzivně volané šablony, aniž bychom je museli v elementu `<xsl:call-template>` explicitně uvádět. Studium kódu ponecháme čtenářům za domácí úkol.

Nyní nastal čas k tomu, abychom ověřili funkčnost vytvořeného stylu. Text byl přetransformován do sekvence elementů, jejichž jména odpovídají jménům maker ve vstupních souborech. Vše si můžeme vypsat do XML souboru šablonou:

```
<xsl:template name='parsed-index'>
  <parsed-index>
    <xsl:copy-of select='$index-items'/>
  </parsed-index>
</xsl:template>
```

Aby to nebylo tak jednoduché, předpokládejme, že soubory nazvané `file1.idx` a `file2.idx` pro nás někdo vytvořil v Linuxu v kódování ISO-8859-2, je nutno je zpracovat současně, ale výsledný `file.xml` si chceme prohlédnout v OS/2

v kódování CP852. Zařídíme to následujícím příkazem (celý text musíte zapsat na jeden řádek):

```
saxon8 -o file.xml -it parsed-index parse-index.xsl
  index=file1.idx,file2.idx enc=iso-8859-2 !encoding=cp852
```

2.2. Abecední řazení pomocného souboru

Řadicí algoritmus je implementován v samostatném souboru `sort-inxex.xsl`. Důvod takové modularizace si vysvětlíme později. Nevýhodou je, že tento styl již nelze použít samostatně, ale musíme oba soubory importovat do jednoho stylu např. takto:

```
<?xml version='1.0' encoding='utf-8'?>
<xsl:stylesheet version='2.0'
     xmlns:xsl='http://www.w3.org/1999/XSL/Transform'
     xmlns:xs='http://www.w3.org/2001/XMLSchema'
     exclude-result-prefixes='#all'>
<xsl:import href='parse-index.xsl'/>
<xsl:import href='sort-index.xsl'/>
</xsl:stylesheet>
```

Uživateli dáme šanci, aby si zvolil jazyk, podle jehož pravidel se má rejstřík řadit. Použijeme k tomu parametry stylu s vhodnou defaultní hodnotou:

```
<xsl:param name='case-order'
           as='xs:NMTOKEN'>lower-first</xsl:param>
<xsl:param name='lang' as='xs:NMTOKEN'>en</xsl:param>
```

Nadefinujeme si též parametr pro zadání jména elementu, jejichž abecedně seřazený seznam se má na výstupu objevit:

```
<xsl:param name='keyword' as='xs:NMTOKEN'>indexentry</xsl:param>
```

Podobně jako v minulém případě si budeme chtít výstup otestovat. Bude nám k tomu sloužit šablona pojmenovaná `sorted-index`. Předpokládáme, že ve finální verzi bude tento styl spolupracovat s dalšími styly a možná bude užitečné, abychom jinými prostředky dokázali ovlivnit metodu řazení. Testovací šablonu tedy definujeme s využitím tunelových parametrů:

```
<xsl:template name='sorted-index'>
  <xsl:param name='entry' as='xs:string' select='$keyword'/>
  <xsl:param name='x-lang' as='xs:NMTOKEN' select='$lang'/>
  <xsl:param name='x-case-order' as='xs:NMTOKEN'
             select='$case-order'/>
```

```
  <xsl:param name='items' as='item()*' select='$index-items'/>
  <sorted-index>
    <xsl:call-template name='sort-index-items'>
      <xsl:with-param name='entry' tunnel='yes' select='$entry'/>
      <xsl:with-param name='x-lang' tunnel='yes'
           select='$x-lang'/>
      <xsl:with-param name='x-case-order' tunnel='yes'
           select='$x-case-order'/>
      <xsl:with-param name='items' as='item()*' select='$items'/>
    </xsl:call-template>
  </sorted-index>
</xsl:template>
```

Samostudiem jste si mohli zjistit, že elementy, jejichž sekvence je uložena v proměnné `$index-items`, obsahují vnořený element `<key>` s řadicím klíčem, element `<text>` s vlastním textem položky (může se shodovat s klíčem a buď vnořenou položku, nebo element `<page>` s číslem stránky. Řazení tedy provedeme víceúrovňově touto rekurzivní šablonou:

```
<xsl:template name='sort-index-items'>
  <xsl:param name='entry' as='xs:string' required='yes'
             tunnel='yes'/>
  <xsl:param name='x-lang' as='xs:string' required='yes'
             tunnel='yes'/>
  <xsl:param name='x-case-order' as='xs:string' required='yes'
             tunnel='yes'/>
  <xsl:param name='items' as='item()*' select='$index-items'/>
  <xsl:for-each-group select='$items[name() = $entry]'
                      group-by='key'>
    <xsl:sort select='current-grouping-key()' data-type='text'
              case-order='{$x-case-order}' lang='{$x-lang}'/>
    <xsl:variable name='key' as='xs:string'
                  select='current-grouping-key()'/>
    <xsl:for-each-group select='current-group()' group-by='text'>
      <xsl:sort select='current-grouping-key()' data-type='text'
                case-order='{$x-case-order}' lang='{$x-lang}'/>
      <xsl:element name='{$entry}'>
        <xsl:attribute name='id'>
          <xsl:value-of select='$key'/>
        </xsl:attribute>
        <xsl:attribute name='text'>
          <xsl:value-of select='current-grouping-key()'/>
        </xsl:attribute>
```

```
        <xsl:call-template name='sort-index-items'>
          <xsl:with-param name='items'
               select='current-group()/element()[name()=$entry]'/>
        </xsl:call-template>
        <xsl:call-template name='process-page-numbers'>
          <xsl:with-param name='items'
               select='current-group()/page'/>
          <xsl:with-param name='keytext'
               select='if ($key = current-grouping-key())
               then $key
               else concat($key,$actual,current-grouping-key())'/>
        </xsl:call-template>
      </xsl:element>
    </xsl:for-each-group>
  </xsl:for-each-group>
</xsl:template>
```

Vybrané elementy nejprve seřadíme a seskupíme podle klíče, potom podle textu položky. Následně rekurzivním voláním seřadíme vnořené elementy a nakonec zpracujeme čísla stran. Všimněte si, že jsme neuvedli tunelové parametry, neboť procesor XSLT to provede za nás.

Při zpracování čísel stran se musíme vypořádat jednak se styly, jednak s rozsahy stran. Nejprve expandujeme explicitně uvedené rozsahy a uložíme výsledek do proměnné `$expanded-ranges`. V šabloně `$process-page-numbers` k tomu slouží kód (vynechali jsme chybové zprávy):

```
<xsl:variable name='expanded-ranges' as='item()*'>
  <xsl:for-each-group select='$items[@range]'
       group-starting-with='*[@range = "open"]'>
    <xsl:variable name='style' as='xs:string'
         select='self::*[1]/@style'/>
    <xsl:for-each select='self::*[1]/@number
                  to current-group()[last()]/@number'>
      <page number='{.}' style='{$style}'/>
    </xsl:for-each>
  </xsl:for-each-group>
</xsl:variable>
```

Nyní seřadíme čísla stran, která nepatří do explicitně zadaných rozsahů, společně se sekvencí vzniklou v minulém kroku. Řadíme podle čísla strany a podle stylu zobrazení. Výsledek uložíme do proměnné `$sorted-pages`:

```
<xsl:variable name='sorted-pages' as='item()*'>
  <xsl:for-each select='$items[not(@range)], $expanded-ranges'>
```

```
    <xsl:sort select='@number' data-type='number'/>
    <xsl:sort select='@style'/>
    <xsl:copy-of select='.'/>
  </xsl:for-each>
</xsl:variable>
```

Nyní potřebujeme skupiny sousedních stran seskupit do rozsahů, přičemž se nemusí jednat jen o rozsahy zadané explicitně. Rozsahy musíme seskupovat samostatně pro jednotlivé styly. Opět se nám bude hodit cyklus pro skupiny:

```
<xsl:variable name='collapsed-ranges' as='item()*'>
  <xsl:for-each-group select='$sorted-pages'
                      group-adjacent='@style'>
    <xsl:variable name='style' as='xs:string'
                  select='self::*/@style'/>
    <xsl:variable name='distinct-pages' as='xs:integer*'
         select='distinct-values(current-group()/@number)'/>
    ...
  </xsl:for-each-group>
</xsl:variable>
```

Funkcí `distinct-values()` jsme vyloučili opakované prvky. Hodnota je poslední (nebo jedinou) stránkou v rozsahu, pokud následující hodnota je nejméně o 2 vyšší, nebo se jedná o poslední prvek v sekvenci. Takovým elementům přidáme atribut `range='close'`:

```
<xsl:variable name='pages' as='item()*'>
  <xsl:for-each select='1 to count($distinct-pages)'>
    <page number='{subsequence($distinct-pages, ., 1)}'
          style='{$style}'>
      <xsl:variable name='pg' as='xs:integer*'
           select='subsequence($distinct-pages, ., 2)'/>
      <xsl:if test='. = count($distinct-pages) or
                      min($pg) &lt; max($pg) - 1'>
        <xsl:attribute name='range'>close</xsl:attribute>
      </xsl:if>
    </page>
  </xsl:for-each>
</xsl:variable>
```

Rozsahy více než dvou stran nahradíme jediným elementem, v němž bude konec rozsahu vložen do atributu `rangeTo`. Atribut `range` ze všech elementů odstraníme.

```
<xsl:for-each-group select='$pages'
                    group-ending-with='*[@range="close"]'>
  <xsl:choose>
    <xsl:when test='current-group()[last()]/@number
              - self::*/@number &gt; 1'>
      <page number='{self::*/@number}'
            rangeTo='{current-group()[last()]/@number}'
            style='{self::*/@style}'/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:for-each select='current-group()'>
        <xsl:copy>
          <xsl:copy-of select='@* except @range'/>
        </xsl:copy>
      </xsl:for-each>
    </xsl:otherwise>
  </xsl:choose>
</xsl:for-each-group>
```

Výslednou sekvenci opět seřadíme, a to podle počátečních a koncových čísel rozsahů a podle stylů:

```
<xsl:for-each select='$collapsed-ranges'>
  <xsl:sort select='@number' data-type='number'/>
  <xsl:sort select='if (@rangeTo) then @rangeTo
                    else -99999' data-type='number'/>
  <xsl:sort select='@style'/>
  <xsl:copy-of select='.'/>
</xsl:for-each>
```

Takto napsaný kód si neporadí správně s případem, kdy do rejstříku vkládáme heslo uvedené na některé z úvodních stránek číslovaných římskými číslicemi. Takový přístup není známkou správného stylu psaní knih, takže obtíže spojené s implementací této vlastnosti nestojí za vynaložené úsilí.

2.3. Formátování výstupu

Formátování výstupu je naprogramováno ve stylu `format-index.xsl`. Opět jej nelze použít samostatně, ale jen společně s předchozími dvěma styly. Zde je specifikována výstupní metoda `text`, ale není uvedeno kódování, aby si jej uživatel mohl zadat sám. Atribut `encoding` elementu `<output/>` není *attribute value template*, takže jej nelze převzít z hodnoty parametru `$enc`.

V transformačním stylu nejprve do pomocné proměnné uložíme seřazený seznam:

```
<xsl:variable name='sorted-list' as='element()'>
  <xsl:call-template name='sorted-index'/>
</xsl:variable>
```

Pokud nechceme mít hlavičky s počátečními písmeny, vytvoříme výstup jednoduchou šablonou:

```
<xsl:template name='noheaders'>
  <xsl:param name='sorted-items' as='element()'
             select='$sorted-list'/>
  <xsl:text>\begin{theindex}</xsl:text>
  <xsl:apply-templates select='$sorted-items'/>
  <xsl:text>&#10;\end{theindex}&#10;</xsl:text>
</xsl:template>
```

Šablony pro formátování výstupu si nastudujte za domácí úkol. Opět využijeme rekurzivní volání, abychom zpracovali vnořené položky.

Hlavičky s počátečními symboly představují pouze drobnou komplikaci. K řešení nám poslouží cyklus přes skupiny, kde využijeme seskupení podle prvního znaku konvertovaného na velké písmeno. V češtině a slovenštině musíme ještě ošetřit CH. To však není vše. Zbývají akcentovaná písmena, zejména dlouhé samohlásky, jež nemají primární řadicí platnost. Vytvoříme si proto nejprve funkci, která určí hodnotu klíče při řazení podle českých nebo slovenských pravidel:

```
<xsl:function name='zw:key'>
  <xsl:param name='key' as='xs:string'/>
  <xsl:value-of
       select="if ($lang != 'cs' and $lang != 'sk') then $key
       else translate($key,
           'ÁÄĎÉĚÍĹĽŇÓÖŐŤÚŮÜŰÝ', 'AADEEILLNOOOTUUUY')"/>
</xsl:function>
```

Tuto úlohu nelze řešit pojmenovanou šablonou, neboť funkci `zw:key()` budeme potřebovat ve výrazu XPath.

Vlastní výstup bude proveden šablonou:

```
<xsl:template name='headers'>
  <xsl:param name='sorted-items' as='element()'
             select='$sorted-list'/>
  <xsl:text>\begin{theindex}</xsl:text>
  <xsl:for-each-group select='$sorted-items/*'
       group-adjacent="if (matches(@id, '^[^\p{L}]'))
       then $symbols else
       if ($use-ch and upper-case(substring(@id, 1, 2)) = 'CH')
```

```
      then 'CH'
      else zw:key(upper-case(substring(@id, 1, 1)))">
    <xsl:value-of select="concat('&#10;&#10;', $lethead-prefix,
                  current-grouping-key(), $lethead-suffix)"/>
    <xsl:apply-templates select='current-group()'/>
  </xsl:for-each-group>
  <xsl:text>&#10;\end{theindex}&#10;</xsl:text>
</xsl:template>
```

Formátování rejstříkových položek je dosaženo použitím týchž šablon jako v případě výstupu bez hlaviček.

2.4. Výhody modularizace

Zřejmou výhodou modularizace je možnost snadné změny formátování výstupu analogicky, jak to řeší MakeIndex ve stylových souborech. Můžeme například v kopii stylu `makeindex.xsl` předefinovat šablonu pro zápis položek.

Modularizace nabízí ještě další možnosti. Nemusíme totiž zpracovávat pouze rejstříky LaTeXových dokumentů. Styl `sort-index.xsl` ponecháme beze změny, ale vytvoříme si vlastní styl (či jiný program) pro konverzi do stejného formátu, jaký generuje styl `parse-index.xsl`, a styl pro formátování výstupu. Abyste si tyto programy mohli snadno otestovat, je struktura pomocných souborů popsána schématem Relax NG, jež bylo programem TRANG [9] konvertováno na XML Schema.

Vezměme si ještě jiný příklad. Předpokládejme, že máme knihu o botanice, k níž chceme sestavit rejstřík českých názvů, rejstřík latinských názvů, rejstřík anglických názvů, rejstříky názvů v dalších jazycích. Dále chceme rejstřík citovaných knih s čísly stran, kde se odkaz vyskytuje, rejstřík citovaných autorů – a pokud budeme takto pokračovat, můžeme překročit povolený počet otevřených souborů. Při použití implementace v XSLT vše vložíme do jediného souboru, pouze různé druhy položek označíme odlišnými makry. Abychom nemuseli tento soubor opakovaně načítat, přepíšeme si výstupní styl `format-index.xsl` tak, aby se každý typ setříděného rejstříku generoval vlastním elementem `<xsl:result-document>`, v němž zavoláme šablonu `sorted-index` se správnými parametry.

3. Náměty k reimplementaci BibTeXu

Myšlenka reimplementace BibTeXu pomocí nástrojů XML není nová (např. Widmann [10], Dagnat et al. [11], Hufflen [12], Beebe [13]). Popisované implementace jsou však založeny na specializovaných nástrojích, případně na jazyku, který

pouze připomíná XSLT. Reimplementace založená čistě na XSLT s využitím modulárních transformačních stylů by otevřela nové možnosti. Správa rozsáhlé bibliografické databáze v obyčejném souboru, ať už ve formátu `.bib`, nebo v XML, je dosti nepohodlná. Tyto údaje by mohly být uloženy v nějaké SQL databázi s pohodlným uživatelským rozhraním, k níž by se z XSLT přistupovalo pomocí XQuery [14], případně by údaje mohly být uloženy v databázi XIndice [15], jež pracuje přímo s XML. Taková implementace by jistě pomohla i projektu Biblet [16].

4. Závěr

České pořekadlo říká: „Kolik jazyků umíš, tolikrát jsi člověkem.“ V obecné rovině je pravdivé, ale z programátorského hlediska lze o jeho platnosti polemizovat. Zatímco s některými cizinci se domluvíme výhradně jejich mateřštinou, téhož výsledku lze dosáhnout programem napsaným v různých jazycích.

Ukázali jsme si, že program MakeIndex lze reimplementovat snadno použitím XSLT. Nepotřebujeme tedy specializovaný nástroj. Místo toho, abychom se učili řadu různých jazyků a jednoúčelových nástrojů, s využitím spolupráce TEXu či LATEXu s procesorem XSLT bychom více času mohli věnovat sazbě věcně správných a typograficky kvalitních dokumentů.

Reference

[1] P. Olšák: *Nový encTEX – kódování UTF-8 v TEXu*. Zpravodaj Československého sdružení uživatelů TEXu, **13** (2), 98–106 (2003).
[2] XSLT 2.0 – `http://www.w3.org/TR/xslt20/`
[3] XPath 2.0 – `http://www.w3.org/TR/xpath20/`
[4] M. Kay: *Schema-aware XSLT Processing*. Konference XML Prague, červen 2005. `http://www.xmlprague.cz`
[5] `http://saxon.sf.net/`
[6] `http://www.goldencode.com`
[7] `http://www.innotek.de`
[8] `http://icebearsoft.euweb.cz/xslt-indexing/`
[9] Trang: Multi-format schema converter based on RELAX NG – `http://www.thaiopensource.com/relaxng/trang.html`
[10] T. Widmann: *Bibulus—a Perl/XML replacement for BibTEX*. TUGboat 24 (3), 468–471 (2003).
[11] F. Dagnat, R. Keryell, L. B. Sastre, E. Donin de Rosière, N. Torneri: *BibTEX++: Toward higher-order BibTEXing*. TUGboat 24 (3), 472–488 (2003).

[12] J.-M. Hufflen: *European bibliography styles and MlBibTEX*. TUGboat 24 (3), 489–498 (2003).
[13] N. Beebe: *A bibliographer's toolbox*. TUGboat 25 (1), 89–104 (2004).
[14] XQuery 1.0 – `http://www.w3.org/TR/xquery/`
[15] Apache XIndice – `http://xml.apache.org/xindice/`
[16] T. Miller: *Biblet: A portable BibTEX bibliography style for generating highly customizable XHTML*. TUGboat 26 (1), 85–96 (2005).

Summary: Processing auxiliary TEX files with XSLT 2.0

The article shows possibilities of processing auxiliary TEX files with XSTL 2.0. The idea is demonstrated on reimplementation of MakeIndex in XSLT. Some thoughts concerning the possibilities of reimplementation of BibTEX purely in XSLT are also presented.

Schválené grantové projekty

Výbor CSTUGu schválil tři grantové projekty, které podle jeho mínění budou užitečné pro českou a slovenskou komunitu uživatelů TEXu. Návrhy grantů jsou zveřejněny v následujícím textu a podrobné informace lze nalézt na webových stránkách sdružení `http://www.cstug.cz`.

Návrh grantu — česká a slovenská podpora pro balík babel LaTeXu

Cíl:
V základní distribuci LaTeXu mít kvalitní podporu češtiny a slovenštiny v balíku babel a pro sazbu fonty v kódování T1, s vhodnou (ne nezbytně stoprocentní) mírou zpětné kompatibility se současnými způsoby sazby českých dokumentů (např. czech.sty z cstexu, csquote ap.).

Požadavky:
1) Upravit stávající verzi podpory české a slovenské sazby v distribuci babelu Johannese Braamse ve dvou fázích tak, aby se minimalizovaly problémy s užitím stylu.

2) Přidat makra ("uv), na která jsou zvyklí uživatelé cslatexu a přidat možnosti stylu csquote.
3) Použití nejnovější verze balíku babel.
4) Použití samodokumentujících maker (.dtx).
5) Akceptovat osmibitové fonty zejména v kódování T1, alternativně i latin2 (to ne za každou cenu).
6) Licence LPPL.

Výstup:
1) Distribuce babelu obsahující dobře dokumentované styly.
2) Dokumentace (uživatelská, ke zdrojovému kódu).
3) Závěrečná zpráva a publikace do Zpravodaje $\mathcal{CS}$TUGu.

Odměna:
10 000 Kč podle úrovně řešení a dodržení termínů, vyplacená poté, co bude vytvořený styl přijat LaTeX teamem do základní distribuce.

Řešitel:
Petr Tesařík

Termín:
První verze do 15. dubna 2006, alfa verze do 15. května, beta verze do 15. června 2006, finální do 15. července 2006.

Převzetí a doplňující informace za zadavatele poskytne:
Petr Sojka.

Návrh grantu — Užitečné Latin Modern pro Čechy, Moraváky a Slováky

Cíl:
V základní distribuci LM mít kvalitně zvládnutá specifika češtiny a slovenštiny tak, aby se rodina mohla stát CSTUGem doporučovaným implicitním fontem LaTeXu pro sazbu fonty v kódování T1.

Požadavky:

1) Bude ustavena pracovní skupina a kompetence členů skupiny.
2) Budou průběžně shromažďovány požadavky komunity uživatelů TEXu v ČR a SR na úpravy LM fontů, ty budou transparentně přeloženy a tlumočeny autorům LM (např. přes WWW stránku na webu CSTUGu). Zde budou také odpovědi autorů LM.
3) Budou připraveny testovací sady slov a vzorníků verifikujících design rysů potřebných pro sazbu českých a slovenských textů.
4) Primární bude kvalita, zpětná kompatibilita s CSfonty bude až sekundární. Srovnání s CS primárně jako test chyb, ne jako požadavek na změny LM.
5) Použití nejnovějších verzí LM fontů.
6) Testování LATEXové podpory LM, mapovacích souborů pro CS včetně vypracování doporučení jejich užití.
7) Licence GPL či LPPL při vývoji testovacího software.

Výstup:

1) Webová stránka dokladující vznesené připomínky a odpovědi autorů.
2) Dokumentace verifikačních a testovacích nástrojů (uživatelská, ke zdrojovému kódu).
3) Závěrečná zpráva a publikace do Zpravodaje $\mathcal{CS}$TUGu.

Navrhovaná odměna:
10 000 Kč včetně cestovného podle úrovně řešení a dodržení termínů, vyplacená poté, co budou změny realizovány ve verzi LM prezentované na BachoTeXu 2006 resp. do termínu dokončení projektu.

Řešitelé:
Karel Píška, Karel Horák

Termín:
Finální verze přípomínek a realizace kerningů do BachoTeXu 2006, alfa verze do 15. 4., beta verze do 15. 5. 2006, dokončení projektu do 30. 6. 2006.

Převzetí a doplňující informace za zadavatele poskytne:
výbor CSTUGu.

Návrh grantu — Český překlad manuálu ConTEXt pro začátečníky

Cíl:
Představit českým a slovenským uživatelům základní možnosti moderního sázecího systému ConTEXt (`www.pragma-ade.com`) a usnadnit jeho použití. Text seznámí čtenáře se syntaxí strukturního značkování ConTEXtu i s možnostmi, jak uživatelsky nastavit formátování jeho značek. Umožnit zveřejnění tohoto překladu ve Zpravodaji.

Požadavky:

1) Přeložit anglický text manuálu ConTEXt *on Excursion* autorů Tona Ottena a Hanse Hagena (`svn://ctx.pragma-ade.nl/manuals/start`). Jde o 124 stran textu A4.
2) Zařadit kapitolu věnující se psaní českých textů.
3) Naformátovat text do formátu Zpravodaje.

Výstup:

1) Podklad pro článek ve Zpravodaji.
2) Zdrojové texty překladu.

Odměna:
10 000 Kč podle úrovně řešení a dodržení termínů, vyplacená poté, co bude překlad schválen redakční radou a přijat ke zveřejnění.

Řešitel:
Ján Buša, Tomáš Hála, David Jež, Martin Kolařík, Libor Škarvada, Petr Tesařík, Vít Zýka

Termín:
Finální verze do konce roku 2006, verze pro korektury, opravy a testování do konce října 2006.

Převzetí a doplňující informace za zadavatele poskytne:
Zdeněk Wagner a Jaromír Kuben.

Zpravodaj Československého sdružení uživatelů TEXu
ISSN 1211-6661 (tištěná verze), ISSN 1213-8185 (online verze)

Vydalo:	Československé sdružení uživatelů TEXu vlastním nákladem jako interní publikaci
Počet výtisků:	700
Uzávěrka:	17. března 2006
Odpovědný redaktor:	Zdeněk Wagner
Redakční rada:	Petr Aubrecht, Jiří Demel, Jaromír Kuben, Petr Sojka, Martin Tkadlčík, Vít Zýka
Tisk a distribuce:	KONVOJ, spol. s r. o., Berkova 22, 612 00 Brno, tel. +420 549 240 233
Adresa:	CSTUG, c/o FEL ČVUT, Technická 2, 166 27 Praha 6
Tel:	+420 224 353 611
Fax:	+420 233 332 938
Email:	`cstug@cstug.cz`

Zřízené poštovní aliasy sdružení CSTUG:

`bulletin@cstug.cz`, `zpravodaj@cstug.cz`
korespondence ohledně Zpravodaje sdružení

`board@cstug.cz`
korespondence členům výboru

`cstug@cstug.cz`, `president@cstug.cz`
korespondence předsedovi sdružení

`gacstug@cstug.cz`
grantová agentura CSTUGu

`secretary@cstug.cz`, `orders@cstug.cz`
korespondence administrativní síle sdružení, objednávky CD-ROM

`cstug-members@cstug.cz`
korespondence členům sdružení

`cstug-faq@cstug.cz`
řešené otázky s odpověďmi navrhované k zařazení do dokumentu CSFAQ

`bookorders@cstug.cz`
objednávky tištěné TEXové literatury na dobírku

ftp server sdružení:
`ftp://ftp.cstug.cz/`

www server sdružení:
`http://www.cstug.cz/`

Uzávěrka příštího čísla: 15. června 2006