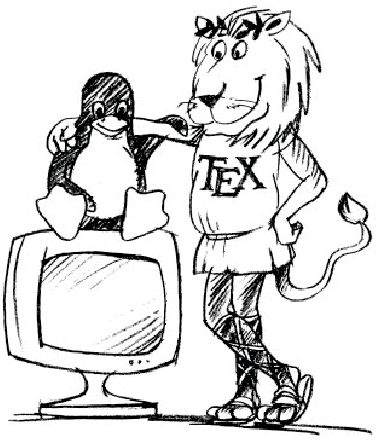
CSTUG



ZPRAVODAJ

Československého sdružení uživatelů TEXu

3–4
2004

ISSN 1211-6661 Tištěná verze
ISSN 1213-8185 Online verze
Ročník 14

OBSAH

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)

Dopis nového předsedy CSTUGu

Jaromír Kuben

Milí přátelé TEXu a příbuzných programů.

V sobotu 27. 11. 2004 proběhlo na MFF UK v Praze valné shromáždění sdružení CSTUGu. Na tomto shromáždění byl zvolen nový čtrnáctičlenný výbor. Tento výbor mne poté na svém prvním zasedání zvolil za předsedu. I když jde o funkci, na niž se kandidáti příliš nehrnuli, potěšilo mne, že lidé, kteří se o rozvoj TEXu u nás velmi zasloužili a kterých si vážím, mne považovali za vhodnou osobu, která bude v následujících třech letech nejvíce zodpovědná za úspěšný chod sdružení. Chtěl bych poděkovat Petru Olšákovi, po němž tuto funkci přebírám, za to, že nesl statečně po tři roky toto nelehké břímě.

V posledních týdnech se rozproudila diskuse, zda nenastal čas ukončit činnost CSTUGu. Jsem přesvědčen, že toto sdružení sehrálo významnou roli v rozšíření TEXu v našich zemích. Sám používám TEX a příbuzné programy již skoro patnáct let. Vím, že zde TEX věčně nebude. Zatím ale neznám vhodný software, který by ho nahradil. Navíc mám kolem sebe řadu lidí, kteří si zvykli TEX používat pro přípravu článků, skript, knih a nejrůznějších dalších tiskovin jako řadoví uživatelé, kteří nehodlají hluboce bádat nad definicemi složitých maker, otázkami instalace TEXovských distribucí apod., ale chtějí jen tento nástroj rutinně používat. Kdyby CSTUG zanikl, určitě by nepřestali okamžitě TEX používat, protože řada časopisů, zejména v oblasti matematiky, vyžaduje příspěvky v TEXu, mnohá velká světová nakladatelství používají pro přípravu určitých publikací výlučně tento sázecí systém. Ale do budoucna by se u nás určitě zhoršil přístup např. k novým instalacím a k literatuře o TEXu a počet uživatelů by klesal rychleji, než by bylo nutné, což by byla podle mého mínění škoda. Domnívám se proto, že čas na zánik CSTUGu ještě zdaleka nenastal.

To vše spolu se skutečností, že se lidí s podobným míněním našlo více, mne vedlo k rozhodnutí opět kandidovat do výboru a k přijetí nabídnuté funkce předsedy. Složení nového výboru, v němž se kromě zkušených matadorů objevilo několik nových ambiciozních tváří, mi dává záruku, že činnost našeho sdružení se může i nadále úspěšně rozvíjet. Za hlavní úkol jsme si vytyčili udržet činnost v té podobě, jak tomu bylo doposud. Tedy v první řadě vydávání Zpravodaje, což vyžaduje zajistit přísun vhodných příspěvků, nákup a distribuci TEXovských instalací na CD a DVD a podporu vydávání literatury o TEXu. Dále podle okolností, zájmu a možností pořádání seminářů a přednášek a vyvíjení další činnosti tak, jak je popsána ve stanovách CSTUGu.

Přeji vám do budoucna hodně radostných chvil s TEXem při tvorbě publikací krásných vzhledem a hodnotných obsahem. Nezapomínejte ve svém okolí TEX

propagovat a snažte se získávat další členy do našeho sdružení. Jen tím bude zajištěna jeho budoucnost a dostatek prostředků na výraznější podporu a rozvoj tohoto skvělého nástroje. Když se vám podaří vytvořit něco pěkného v problematice týkající se TEXu a příbuzných programů, neváhejte se o to podělit s ostatními formou příspěvku do Zpravodaje.

Protože se zdá, že byste toto povídání mohli číst ještě v letošním roce 2004, přeji vám hezké Vánoce a hodně zdraví a nejen TEXovských úspěchů v novém roce.

S pozdravem

Jaromír Kuben

Členové nového výboru CSTUGu a sekretářka CSTUGu. Zleva zadní řada: Petr Tesařík, Jiří Veselý, Jiří Rybička, David Jež, Ján Buša, Petr Březina, Vít Zýka, Jiří Demel, Helena Holovská (sekretářka), zleva přední řada: Libor Škarvada, Petr Sojka, Zdeněk Wagner, Jaromír Kuben (nově zvolený předseda).

Zápis z valného shromáždění ze dne 27. 11. 2004

1. Schválení programu

Navržený program byl schválen.

2. Volba zapisovatele

Zapisovatelem byl zvolen Jiří Kosek. [pro: 47, proti: 0, zdrželo se: 1]

3. Volba členů volební komise

Do volební komise byli zvoleni Miroslav Dont, Jaromír Antoch a Štěpán Kasal. [pro: 45, proti: 0, zdrželo se: 3]

4. Zprávu o činnosti sdružení

Zprávu výboru o činnosti sdružení přednesl Petr Sojka (viz připojený dokument).

Na dotaz z pléna bylo vysvětleno, že LPZ = LaTeX pro začátečníky.

5. Zpráva o hospodaření

Zprávu o hospodaření vyhotovenou panem Vlčkem prezentoval předseda sdružení Petr Olšák.

Petr Sojka požaduje, aby do zprávy o hospodaření byly zahrnuty i pohledávky vůči nakladatelství Konvoj.

Zdeněk Wagner upozorňuje na chybu ve zprávě za prvních 10 měsíců roku 2003 a za celý rok 2003. Tyto nesrovnatelnosti budou odstraněny po konzultacích s firmou Vlček a opravená zpráva bude publikována ve Zpravodaji. [pro: 48, proti: 0, zdrželo se: 0]

6. Zpráva revizora

Revizní zprávu Pavla Sekaniny přednesl Petr Olšák (viz připojený dokument).

Revizní zprávu Tomáše Hály přednesl Petr Olšák (viz připojený dokument).

Výbor vyjasní připomínky ke zprávám po konzultaci s panem Vlčkem. [pro: 48, proti: 0, zdrželo se: 0]

7. Volby nového výboru

Návrh volebních pravidel podle stanov přednesl Miroslav Dont.

Do výboru se dostane ten kandidát, který bude mít alespoň 50 % hlasů.

Výbor musí mít alespoň 10 členů.

Individuální člen má jeden volební lístek, zástupce kolektivního člena má tři volební lístky.

Valné shromáždění schválilo tato volební pravidla. [pro: 48, proti: 0, zdrželo se: 0]

8. Seznámení s kandidáty

Ján Buša, Košice, zpracování a šíření informací o TEXu (překlady apod.).

Petr Březina, Plzeň, sazba řečtiny a latiny, šíření TEXové osvěty mezi klasickými filology, tvorba nástrojů pro sazbu řečtiny a latiny (vzory dělení slov a fonty).

Jiří Demel, FSV ČVUT Praha, práce technického charakteru, e-mail.

David Jež, student VUT Brno, doposud nebyl členem CSTUGu, nová krev do sdružení.

Jaromír Kuben, Univerzita obrany Brno, kandidát na předsedu sdružení, organizace chodu a propagace sdružení, český bibtex, instalace TEXu pod OS/2 postavená na web2c, příprava pokročilejšího manuálu LATEX2e.

Jiří Rybička, MZLU Brno, podpora pro začátečníky (publikace, výuka, vedení DP apod.).

Rudolf Schwarz, Univerzita obrany Brno, pomoc předsedovi s chodem sdružení, dlouholetý uživatel TEXu.

Petr Sojka, FI MU Brno, koordinace aktivit s ostatními LUG i TUG, aktivní činnost proti rušení sdružení, příp. převzetí funkce šedé eminence sdruení :-).

Libor Škarvada, FI MU Brno, údržba a aktualizace dokumentu CSTUG-FAQ - základního informačního zdroje CSTEXisty, organizační a technická výpomoc. Chce podpořit zachování aktivit sdružení aspoň na současné úrovni.

Marcel Takáč, Fakulta prírodných vied Žilinskej univerzity, Žilina, kontaktní osoba pro slovenské členy, slovenská podpora do CSTEXu, WWW stránky www.cstug.sk, vybírání členských příspěvků na Slovensku.

Petr Tesařík, student FF UK Praha.

Jiří Veselý, MFF UK Praha, pomoc mladším členům výboru.

Zdeněk Wagner, AV ČR Praha, editor Zpravodaje CSTUGu, organizace interního grantového systému sdružení.

Vít Zýka, TYPOkvítek Praha, podpora používání pdfTEXu, ConTEXtu a metapostu a zachování této podpory v rámci současných aktivit sdružení (diskusní list, Zpravodaj, distribuce TEXLive).

9. Volby

10. Různé

Petr Sojka poděkoval Petrovi Olšákovi za vedení sdružení v posledních třech letech.

Petr Olšák vysvětlil, proč dál nekandiduje do výboru sdružení.

Petr Sojka novému výboru doporučuje pozitivně motivovat členy sdružení ke spolupráci.

Zdeněk Wagner vyzval ostatní k psaní článků do Zpravodaje, včetně článků pro začínající uživatele.

11. Vyhlášení výsledků voleb

Miroslav Dont seznámil s výsledky voleb.

48 hlasovacích lístků vydáno, 48 odevzdáno.

47 hlasů získali Ján Buša, Petr Březina, Jiří Demel, David Jež, Jaromír Kuben, Jiří Rybička, Petr Sojka, Petr Tesařík.

48 hlasů získali Rudolf Schwarz, Libor Škvarda, Marcel Takáč, Jiří Veselý, Zdeněk Wagner, Vít Zýka.

Všichni kandidáti získali více jak 50 % hlasů a stávají se členy nového výboru.

Zapsal: Jiří Kosek, `jirka@kosek.cz`

Ověřil: Petr Olšák, `olsak@math.feld.cvut.cz`
odstupující předseda

Zpráva o činnosti CSTUGu

SLT 2004

24.–27. června 2004 proběhl ve Znojmě opět ve spolupráci s CZLUGem seminář SLT 2004. CSTUG přispěl tvorbou sborníku (Petr Olšák si dva večery

zaTEXoval) a částečnou pomocí s organizací (Zdeněk Wagner). Na tvorbě webových stránek semináře se významně podílel webmaster Alexandr Babič (člen CSTUG i CZLUG). Přes nevhodně vybraný termín, téměř nulový a opožděný marketing a úzce odborný program se akce zúčastnilo 56 účastníků, což je méně, než organizátoři očekávali. V TEXové sekci bylo prezentováno 11 odborných přednášek, což je poměrně slibná indikace toho, co se v našich krajích kolem TEXu děje.

Zpravodaj

Redakční tým okolo Zdeňka Wagnera pokračoval v tradici vydávání Zpravodaje ve značném předstihu před TUGboatem, a tak mají členové CSTUGu ve svých schránkách čísla 1/2004 a 2/2004. Zpravodajové dvojčíslo 3–4/2004 by mohlo být rozesíláno ještě letos, spolu s disky TEXlive. Zdeněk Wagner stále udržuje webové stránky Zpravodaje, digitalizuje a zveřejňuje starší čísla. S pomocí Josefa Svobody a Jiřího Koska byla zprovozněna experimentální verze vyhledávání podle jmen autorů a klíčových slov a je téměř dokončen, opět v experimentální verzi, převod obsahů Zpravodajů do XML tak, aby bylo možno vytvářet výpisy nejrůznějších typů. Zatím nejsou v této verzi zveřejněny abstrakty, to bude provedeno pravděpodobně ještě letos nebo začátkem roku 2005.

TEXlive

Tým TEXlive po usilovném půlročním snažení přichází s distribucí TEXlive 2004 na DVD a CD (DVD je live s CTANem, na CD je instalační verze a zároveň se vydává další CD s distribucí proTEXt pro Windows, neboť se nestihl dodělat instalační program pro Windows.) Jan Buša s Petrem Sojkou připravili pro projekt českou a slovenskou verzi dokumentace (instalační příručku), aby rozšíření distribuce v ČR a SR bylo co nejsnazší. Bohužel, betatestování se kromě Petra Olšáka, který si ohlídal, aby TEXlive obsahoval jeho modifikaci TEXu s názvem encTEX, nikdo nezúčastnil, takže se můžeme ještě dočkat nepříjemných překvapení. V jejich odstranění snad pomůže nová verze CSTUG FAQ, kterou připravili Libor Škarvada a Radovan Panák, a která se objeví také na zmíněných discích. Disky CSTUG objednal v počtu 466 ks DVD (TEXlive+CTAN), 372 ks CD 1 (proTEXt) a 372 CD 2 (inst) a členové je dostanou spolu se Zpravodajem 3–4.

Doména `cstug.cz`, www, CTAN

Díky panu Wagnerovi a panu Sojkovi se podařilo nepřijít o doménu `cstug.cz` a převést ji pod registrátora Globe.cz. Na `ftp.cstug.cz` je zrcadlen archív CTAN a udržuje se lokální archív sdružení. Správu webu na jaře převzal Alexander Babič a je současným webmasterem CSTUGu. Diskusní list cstexu a

členů CSTUGu administruje Martin Bílý, diskusní list výboru CSTUGu a filtrování pošty sdružení prováděl Jiří Demel. Výbor za celý rok nezasedal, vše se řešilo prostřednictvím elektronické komunikace.

Podpora vydávání literatury

CSTUG pokračuje v podpoře vydávání TEXové literatury formou bezúročné půjčky nakladatelství KONVOJ, spol. s r. o. (půjčky se průběžně splácejí letos jedna třetina z TST a TBN, a dvě třetiny z 3. vydání LPZ). Rychlost splácení LPZ ukazuje na rozšíření a hlad po podpoře literatury pro začátečníky.

Členská základna, administrativa

Členská základna se po úbytku v předchozích letech stabilizovala na cca 300 individuálních a 50 kolektivních členech. Úbytek členů se podařilo zastavit například důsledným dosíláním faktur kolektivním členům. Za tyto aktivity a podobné aktivity vděčí výbor paní Holovské, která nesla břemeno nevděčných administrativních úkonů, ke kterým se nikdo nehlásí. Účetnictví CSTUG vede bez obtíží firma Vlček.

Ostatní (CSTUGem nedotované) aktivity členů

Petr Olšák se zúčastnil setkání polských TEXistů BachoTEX 2004 a domluvil spolupráci na vývoji LM fontů. Napsal o tom zprávu do Zpravodaje. Petr Sojka, Jan Holeček a Karel Piška se zúčastnili konference EuroTEX 2004 v řeckém Xanthi.

Zprávu z pověření předsedy vypracoval Petr Sojka

V Brně dne 26. 11. 2004

Zpráva o hospodaření za rok 2003

Zprávu o hospodaření ve formě excelové tabulky (převedené do TEXu níže) vypracoval pan Vlček. Na valném zhromáždění byla kromě této zprávy zveřejněna i zpráva za část roku 2004 a za část roku 2003. Bylo konstatováno, že částka „příspěvky – individuální členové“ se při přechodu z části roku 2003 na celý rok 2003 nelogicky snížila a že to vyžaduje vysvětlení. O vysvětlení jsme požádali účetního pana Vlčka. Ten podal následující vysvětlení:

Ve zprávě za část roku byly v kolonce „příspěvky – individuální členové“ zahrnuty příjmy, které byly později při kontrole příjmů z členských příspěvků vyhodnoceny jako příspěvky za jiný rok než rok 2003. Tyto příjmy byly tedy

přesunuty do odpovídajícího jiného roku. Zpráva za část roku je vždy jen orientační a rozhodující je zpráva za celý kalendářní rok.

Kromě toho valné shromáždění žádalo upřesnit pohledávky vůči firmě Konvoj. Účetní navázal v této věci kontakt s panem Hálou z firmy Konvoj s požadavkem, aby zkontroloval, zda údaje o našich pohledávkách jsou v souladu s jeho účetními doklady. Do uzávěrky tohoto čísla Zpravodaje se bohužel nepodařilo tuto kontrolu dokončit.

Petr Olšák

Zpráva o hospodaření společnosti „Československé sdružení uživatelů TEXu“ ROK 2003

	stav k 1.1.03	stav k 31.12.03	poznámka
Finanční hotovost	**628 235,34 Kč**	**614 927,71 Kč**	
pokladna Kč	5 198,30 Kč	7 294,00 Kč	
pokladna SK	1 889,34 Kč	6 427,89 Kč	
běžný účet	41 147,70 Kč	101 205,82 Kč	
termínovaný vklad	580 000,00 Kč	500 000,00 Kč	
Výnosy		**168 280,50 Kč**	
příspěvky – kolektivní členové		93 150,00 Kč	
příspěvky – individuální členové		64 989,52 Kč	
úroky z běžného účtu		166,98 Kč	
úroky z termínovaného vkladu		6 351,18 Kč	
ostatní výnosy		3 475,32 Kč	
kursový výnos		147,50 Kč	z úhrady došlé faktury v dolarech na členský příspěvek
Náklady		**175 124,48 Kč**	
kancelářské potřeby		1 766,60 Kč	
materiál operativně evid.		41 086,62 Kč	Lehman + clo a DPH
cestovné		15 078,00 Kč	
výroba zpravodaje		50 977,70 Kč	
účetnictví		12 600,00 Kč	
poštovné		6 484,04 Kč	
poštovné – dosílky		6 373,30 Kč	
poštovné – poukázečné		12,00 Kč	z platby složenkou
mzdové náklady		19 000,00 Kč	dohody o provedení práce
ostatní náklady		376,00 Kč	vyhotovení JCD
finanční náklady		5 564,47 Kč	poplatky za vedení účtu a transakce
kursovní ztráta		678,25 Kč	
příspěvky		15 127,50 Kč	500 USD

Dlouhodobý majetek společnost nevlastní.

Zpracoval dne 20.11.2004 Pavel Vlček – jednatel společnosti ECQUO s.r.o., pověřené zpracováním účetnictví.

DocBook a generování rejstříků

Jiří Kosek

DocBook je dnes již považován za zcela standardní formát dokumentace. Dobrá dokumentace se ovšem neobejde bez dobrého rejstříku. Příspěvek posluchače seznámí s možnostmi DocBooku a XSL stylů pro generování rejstříku. Pozornost bude věnována i dodržování českých specifik při řazení a seskupování rejstříkových hesel. Budou ukázány techniky, jak s využitím standardních nástrojů generovat několik rejstříků v dokumentu a jak rejstříky automaticky vytvářet na základě sémantického značkování.

DocBook se stává stále populárnějším formátem pro vytváření dokumentace k mnoha softwarovým projektům, psaní článků, knih, učebních textů, skript apod. Obliba roste zejména díky aplikacím podporujícím DocBook. Editory XML dokumentů jsou stále pohodlnější a dokonce již existují WYSIWYG editory, které jsou zdarma[1]. Nástroje pro zpracování DocBooku a zejména nejpoužívanější XSL styly[2] umožňují dokument převádět do mnoha výstupních formátů a podobu výstupu přitom ovlivnit řadou parametrů.

Užitnou hodnotu dokumentů, zvláště tištěných, výrazně zvyšuje kvalitní rejstřík. Vytvoření dobrého rejstříku je velmi pracné a zodpovědné, a proto bývá často svěřeno do rukou specialisty. Bohužel v podmínkách dokumentace k open-source projektům, samizdatových publikací a dokonce i knih určených pro malý český trh se finanční a časové zdroje na práci rejstříkového specialisty většinou nedostávají. Proto se v tomto příspěvku podíváme na možnosti tvorby rejstříků v DocBooku tak, aby si každý autor dokumentu mohl sestavit rejstřík sám. Ukážeme si i pokročilejší možnosti, jako více rejstříků k jednomu dokumentu. Nakonec si ukážeme, jak snadno a automaticky doplnit rejstřík do dokumentů, které používají sémantické značkování k vyznačení objektů jako jsou názvy funkcí, jména souborů apod.

Vyznačování rejstříkových hesel

Rejstříková hesla se zapisují přímo do dokumentu pomocí elementu `indexterm`. Jeho obsah se v dokumentu nezobrazuje, ale slouží jako podklad pro generování rejstříku.

```
<para>Bohatství moderních společností je založeno na
informacích<indexterm><primary>informace</primary></indexterm>.</para>
```

[1] `http://xmlmind.com/xmleditor/`

[2] `http://docbook.sf.net`

Do elementu `indexterm` se zapisují hesla a to i víceúrovňová:

```
<indexterm>
<primary>informace</primary>
</indexterm>

<indexterm>
<primary>informace</primary>
<secondary>získání</secondary>
</indexterm>

<indexterm>
<primary>informace</primary>
<secondary>šíření</secondary>
</indexterm>

<indexterm>
<primary>informace</primary>
<secondary>šíření</secondary>
<tertiary>ústní</tertiary>
</indexterm>
```

V rejstříku se pak takto definovaná hesla objeví například jako:

```
informace, 13
  šíření, 17
    ústní, 25
  získání, 15
```

Pokud nějakému termínu odpovídá větší úsek dokumentu, můžeme ho celý pokrýt jako rozsah. Použijí se dva elementy `indexterm`, které označují začátek a konec platnosti určitého hesla.

```
<indexterm class="startofrange" id="ix.xml.historie">
<primary>XML</primary>
<secondary>historie</secondary>
</indexterm>
  ...
<indexterm class="endofrange" startref="ix.xml.historie"/>
```

Ve vygenerovaném rejstříku pak dostaneme interval:

```
XML
  historie, 27--42
```

Pokud chceme, aby se položka řadila nestandardním způsobem, použijeme atribut `sortas`. Třídění se pak provede podle jeho obsahu, ne podle skutečného textu hesla. To je výhodné v případech, kdy heslo obsahuje nějaké speciální znaky apod. Následující příklad v rejstříku zobrazí písmeno Ω, ale bude se řadit jako text „Omega“.

```
<indexterm>
<primary sortas="Omega">&Omega;</primary>
</indexterm>
```

Chceme-li některé výskyty hesla v rejstříku zvýraznit (například mít stránku s primární definicí hesla tučně), můžeme u hesla nastavit jeho důležitost.

```
<indexterm significance="preferred">
<primary>informace</primary>
</indexterm>
```

Nechceme-li, aby rejstříkové heslo obsahovalo odkaz na konkrétní číslo strany, ale odkaz na jiné heslo, můžeme k tomu využít elementy `see` a `seealso`.

```
<indexterm>
<primary>DTD</primary>
</indexterm>

<indexterm>
<primary>definice typu dokumentu</primary>
<see>DTD</see>
</indexterm>

<indexterm>
<primary>XML schéma</primary>
<seealso>DTD</seealso>
</indexterm>
```

Ve výsledném rejstříku bychom dostali:

```
- D -
definice typu dokumentu, viz DTD
DTD, 42

- X -
XML schéma, 81, viz též DTD
```

Tímto jsme se seznámili skoro se všemi možnostmi zápisu rejstříkových hesel v DocBooku. Nezmínili jsme pouze možnost uložit definici rejstříkových hesel zcela mimo místo jejich výskytu s využitím atributu `zone`. Tato metoda není dle mého názoru příliš praktická, více se o ní můžete dočíst v [4].

Generování rejstříku

XSL styly pro DocBook generují rejstřík zcela automaticky. Na místě, kde chceme mít rejstřík, stačí uvést element `index`. Ten je při zpracování automaticky nahrazen rejstříkem. Pro generování rejstříku tak není potřeba dokument zpracovávat opakovaně, jak to známe např. ze systému TEX a makeindex, nebo z DSSSL stylů.

Při generování rejstříku se přitom samozřejmě respektují možnosti použitého výstupního formátu. Rejstřík v HTML stránce tak neobsahuje čísla stran, ale názvy sekcí či kapitol, ve kterých se heslo vyskytuje. Názvy zároveň slouží jako hypertextové odkazy, které dokáží skočit na místo výskytu rejstříkového hesla.

Při výstupu do HTML Helpu se z rejstříkových hesel vytvoří přímo rejstřík na úrovni HTML Helpu.

Lehce problematický je však tištěný výstup prováděný přes formátovací objekty do PDF. Princip generování rejstříku dokumentu XML v XSL je dvoufázový proces [2]. V první fázi se pomocí XSLT dokument přetransformuje na formátovací objekty, které abstraktním způsobem popisují vzhled tištěného dokumentu. Čísla stran v rejstříku v tuto chvíli nejsou a nemohou být vyhodnocena. K samotnému zalomení textu do stránek a vyhodnocení čísel stránek dojde až během následující fáze formátování. Stene-li se však, že jedno rejstříkové heslo se vyskytuje na jedné stránce dvakrát, objeví se číslo této stránky ve výstupu duplicitně. Tento velmi nepříjemný nedostatek lze naštěstí řešit několika způsoby, o kterých se zmíníme v další části článku.

Dalším problémem je generování rejstříků pro jiné jazyky, než je angličtina. Generování rejstříku odpovídá seskupení všech rejstříkových hesel do skupin podle jejich prvního písmena, seřazení skupin podle abecedy a seřazení hesel v jedné skupině podle abecedy. Přesně tento algoritmus implementují i XSL styly. Pro češtinu je však nedostatečný. Písmeno „ch“ je složeno ze dvou znaků, ale přitom tvoří samostatnou skupinu. Slova začínající na „e“ a „é“ patří do jedné skupiny, kdežto slova začínající na „c“ a „č“ patří do dvou různých skupin.

Implementace korektního českého rejstříku není jednoduchá. Nástroje nabízené standardem XSLT pro seskupování jsou velmi slabé, a i řazení pro jednoduché jazyky jako angličtina je výzva. Navíc jsou styly pro DocBook psány tak, aby podporovaly dokumenty v různých jazycích. Každý jazyk má odlišná pravidla pro řazení a seskupování rejstříkových hesel. Jazyk XSLT však nenabízí dostatečné možnosti pro parametrizaci seskupovacího kódu na základě jazyka. Lze však využít rozšíření, které některé implementace XSLT nabízejí a dosáhnout tak požadovaného výsledku. K problematice generování rejstříku podle českých zvyklostí se také ještě vrátíme.

Odstranění duplicitních čísel stran v rejstříku

Jak jsme již zmínili, současné verze jazyků XSLT a XSL-FO nenabízejí podporu pro eliminování duplicitních čísel stran v rejstříku. Problém jde obejít dvěma způsoby. Prvním z nich je využití procesoru FO, které implementuje rozšíření XSL-FO pro generování rejstříků. Druhá možnost spočívá ve víceprůchodovém zpracování dokumentu, během kterého se detekují a následně odstraní duplicity.

Rozšíření pro generování rejstříků obsahují dva nejpoužívanější a nejlepší komerční procesory FO – XEP[3] a XSL Formatter[4]. Styly pro DocBook tato rozšíření podporují, stačí pomocí parametru říci, že se mají použít. Např. XEP by se pak spouštěl pomocí parametrů:

```
xep -xml dokument.xml -xsl .../fo/docbook.xsl -param xep.extensions=1
```

V praxi však většinou parametrů nastavujeme více, a proto je praktické vytvořit si styl s úpravami (podrobněji viz např. [3] a [1]). Tento styl nejprve naimportuje standardní styly a pak provede potřebné změny v nastavení parametrů.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

<xsl:import
  href="http://docbook.sourceforge.net/release/xsl/current/fo/docbook.xsl"/>

<xsl:param name="paper.type" select="'A4'"/>
<xsl:param name="xep.extensions" select="1"/>

</xsl:stylesheet>
```

Pro XSL Formatter se odpovídající parametr jmenuje `axf.extensions`. Nastavení parametrů způsobí odstranění duplicitních čísel stran a vytvoření intervalů ze sekvencí po sobě jdoucích stránek. Vyskytuje-li se jedno rejstříkové heslo na stranách

```
5, 5, 8, 9, 10, 37
```

dostaneme na výstupu mnohem lepší podobu

```
5, 8--10, 37
```

V budoucnu nebude nutné přizpůsobovat výstup stylů použitému procesoru, protože příští verze XSL-FO 1.1 bude již přímo obsahovat podporu pro generování rejstříků[5].

Používáme-li jiný procesor než XEP nebo XSL Formatter, musíme pro odstranění duplicit podstoupit mnohem složitější proces. Tento postup je nutné využít například společně s procesorem FOP[6], který je jednou z mála alespoň částečně použitelných open-source implementací XSL-FO. Nejprve dokument zpracujeme se zapnutým parametrem `make.index.markup`. To způsobí, že rejstřík bude

[3] `http://www.renderx.com/`

[4] `http://www.antennahouse.com/`

[5] `http://www.w3.org/TR/2003/WD-xsl11-20031217/#d0e12534`

[6] `http://xml.apache.org/fop/`

v PDF dokumentu obsahovat okolo hesel a čísel stran značkování. PDF dokument pak můžeme převést na čistý text, ze značek extrahovat čísla stran, odstranit duplicity a nově vzniklý rejstřík natvrdo začlenit do dokumentu. Postup je to nepohodlný a pro češtinu asi nebude fungovat zcela spolehlivě, protože FOP neumí do PDF vkládat korektní mapování z použitého písma do Unicode. Při extrakci čistého textu z PDF tak dostaneme poškozený text. Není to však velké omezení, protože pro kvalitní výstup je FOP stejně nepoužitelný.

Seskupování hesel podle českých zvyklostí

DocBookové styly přizpůsobují svůj výstup jazyku, ve kterém je dokument napsán. Aktivní jazyk je přitom možné určit pomocí atributu `lang`.

```
<?xml version='1.0' encoding='utf-8'?>
<!DOCTYPE book PUBLIC '-//OASIS//DTD DocBook XML V4.2//EN'
                      'http://www.oasis-open.org/docbook/xml/4.2/docbookx.dtd'>
<book lang="cs">
  ...
</book>
```

Díky již dříve zmíněným omezením jazyka XSLT však nemohou styly vzít aktivní jazyk v úvahu při generování rejstříku. Naštěstí některé z XSLT procesorů umožňují definice uživatelských funkcí, pomocí kterých již lze implementovat seskupování závislé na jazyku. Protože tato část stylů používá nestandardní instrukce XSLT a mohla by způsobit problémy s kompatibilitou, není zahrnuta do standardního stylu.

Chceme-li český rejstřík generovat, musíme používat XSLT procesor, který podporuje definování uživatelských funkcí podle EXSLT[7] a tyto funkce lze použít v definici klíče (`xsl:key`). Těmto kritériím vyhovuje například Saxon. V době psaní článku však ještě nešlo použít xsltproc, protože obsahoval nějaké chyby spojené právě s inicializací klíčů obsahujících uživatelsky definované funkce.

Stačí si pak vytvořit styl s úpravami, který ve standardních stylech předefinuje některé šablony generující rejstřík. Kód již je hotový v souboru `autoidx-ng.xsl` a stačí jej proto sloučit se standardními styly.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

<xsl:import
  href="http://docbook.sourceforge.net/release/xsl/current/fo/docbook.xsl"/>
<xsl:include
  href="http://docbook.sourceforge.net/release/xsl/current/fo/autoidx-ng.xsl"/>

<!-- Další úpravy, nastavení parametrů -->
</xsl:stylesheet>
```

[7] `http://www.exslt.org`

Úpravy kódu pro generování rejstříku jsou dostupné i pro HTML výstup, opět v souboru `autoidx-ng.xsl` v odpovídajícím adresáři.

Takto upravené styly přiřadí hesla správně do skupiny, a skupiny seřadí správně podle české abecedy. Je ošetřena i problematika písmena „ch“. Řazení hesel v jedné skupině je ponecháno na XSLT procesoru. Saxon[8] běžně české řazení nezvládá, ale můžeme jej o podporu českého řazení snadno rozšířit. Stačí, když do javové cesty (`CLASSPATH`) přidáme zkompilovanou třídu `Compare_cs`, jejíž kód je velmi jednoduchý.

```
package com.icl.saxon.sort;

import java.text.Collator;
import java.util.Locale;

public class Compare_cs extends TextComparer
{

    int caseOrder = UPPERCASE_FIRST;

    public int compare(Object a, Object b)
    {

        Collator csCollator = Collator.getInstance(new Locale("cs", "cz"));
        return csCollator.compare(a, b);
    }

    public Comparer setCaseOrder(int caseOrder)
    {
        this.caseOrder = caseOrder;
        return this;
    }

}
```

Více rejstříků v jednom dokumentu

V některých typech publikací je zapotřebí několika samostatných rejstříků – např. předmětného a jmenného. XSL styly si s tímto požadavkem snadno poradí. Při zápisu rejstříkového hesla stačí v atributu `role` uvést námi zvolený identifikátor rejstříku.

```
<para>Bohatství moderních společností je založeno na
informacích.<indexterm role="subj"><primary>informace</primary></indexterm>
O rozvoj informační teorie se ve 40. letech zasloužil Claude Shannon.
<indexterm role="name"><primary>Shannon, Claude</primary></indexterm></para>
```

[8] Máme teď na mysli verzi 6.5.3, která se používá s XSL styly pro DocBook, ne verzi 7.x, která implementuje návrh XSLT 2.0.

Na konec dokumentu pak vložíme dva elementy `index`, u kterých určíme jaká hesla do nich mají spadat.

```
<index role="subj"/>

<index role="name">
<title>Jmenný rejstřík</title>
</index>
```

Zda se bude generovat několik rejstříků podle obsahu atributu `role` ovlivňuje parametr `index.on.role`, který je standardně zapnutý.

Ze sémantiky až do rejstříku

Výhoda DocBooku a XML obecně oproti jiným technologiím pro přípravu dokumentů je možnost velmi jemného přiřazovaní významu jednotlivým částem textu. DocBook obsahuje několik desítek sémantických elementů, které umožňují odlišit jména souborů, od názvů funkcí, příkazů atd. Podívejme se na ukázku odstavce, který takto sémantické značkování používá.

```
<para>Příkaz <command>rm</command> je užitečný, ale používejte
jej opatrně. Některé soubory jako například
<filename>/etc/passwd</filename> jsou pro váš systém poměrně důležité.</para>
```

U větších tištěných příruček je velmi užitečné, pokud se všechny důležité termíny vyskytují v rejstříku. Mezi tyto termíny mohou v některých příručkách patřit i názvy příkazů nebo souborů. Pro zařazení hesel z předchozího příkladu do rejstříku bychom museli ručně doplnit odpovídající značkování.

```
<para>Příkaz <command>rm</command><indexterm><primary>rm</primary></indexterm>
<indexterm><primary>příkaz</primary><secondary>rm</secondary></indexterm>
je užitečný, ale používejte jej opatrně. Některé soubory jako například
<filename>/etc/passwd</filename>
<indexterm><primary>/etc/passwd</primary></indexterm>
jsou pro váš systém poměrně důležité.</para>
```

V rejstříku bychom pak dostali následující hesla

```
- Symboly -
/etc/passwd, 42

- P -
příkaz,
  rm, 42

- R -
rm, 42
```

Výsledek je sice užitečný, ale po pravdě řečeno, kdo by chtěl do vstupního dokumentu zapisovat mnoho redundantní informace. V tomto případě je naštěstí mapování sémantických značek na rejstříková hesla velmi jednoduché a

jednoznačné, takže je můžeme snadno algoritmizovat. Není problém napsat jednoduchý XSLT styl, který do dokumentu obsahujícího pouze sémantické značky doplní rejstříková hesla podle našich požadavků.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

<!-- Zkopírování celého dokumentu -->
<xsl:template match="node()|@*">
  <xsl:copy>
    <xsl:apply-templates select="node()|@*"/>
  </xsl:copy>
</xsl:template>

<!-- Příkazy se dají do rejstříku na dvě místa -->
<xsl:template match="command">
  <!-- Zkopírování původního elementu -->
  <xsl:copy-of select="."/>
  <!-- Vytvoření rejstříkových hesel -->
  <indexterm>
    <primary><xsl:value-of select="."/></primary>
  </indexterm>
  <indexterm>
    <primary>příkaz</primary>
    <secondary><xsl:value-of select="."/></secondary>
  </indexterm>
</xsl:template>

<!-- Každé jméno souboru se také přidá do rejstříku -->
<xsl:template match="filename">
  <!-- Zkopírování původního elementu -->
  <xsl:copy-of select="."/>
  <!-- Vytvoření rejstříkového hesla -->
  <indexterm>
    <primary><xsl:value-of select="."/></primary>
  </indexterm>
</xsl:template>

</xsl:stylesheet>
```

Zpracujeme-li nyní náš dokument tímto stylem, dostaneme dočasný dokument, kde budou pro všechny příkazy a jména souborů doplněná rejstříková hesla. Dočasný soubor pak můžeme zpracovat běžnými XSL styly pro DocBook. Celý proces generování dočasného dokumentu a jeho následného zpracování si můžeme zautomatizovat pomocí makefile, dávkových souborů nebo podobné techniky.

Docbookové styly nám však nabízejí možnost výše zmíněného automatického doplnění rejstříkových hesel přímo během zpracování dokumentu styly. Není

proto potřeba vůbec využívat dočasný soubor a zpracovávat dokument dvěma průchody.

Řešení využívá vlastnosti stylů, které se říká profilování. Profilování umožňuje podmíněně zpracovávat jen určité části dokumentu na základě hodnot uložených v atributech. V dokumentu tak můžeme např. označit, že některé kapitoly jsou určené pro začátečníky používající náš program v Linuxu a některé zase pro pokročilé uživatele Windows. Styly pro DocBook při zpracování dokumentu nejprve provedou odfiltrování nepotřebných částí dokumentu a pak provedou klasické zpracování. Filtrování je přitom interně implementováno jako speciální režim, ve kterém se provádí podmíněné kopírování dokumentu. Protože se profilování obvykle provádí pro elementy jako kapitola a podkapitola a málokdy pro sémantické inline elementy, můžeme v tomto režimu přidat šablony, které se kromě profilování postarají o doplnění rejstříkových hesel právě pro sémantické elementy.

```
<?xml version="1.0" encoding="utf-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                version="1.0">

<!-- Naimportování původního stylu -->
<xsl:import
href="http://docbook.sourceforge.net/release/xsl/current/fo/profile-docbook.xsl"/>

<!-- Příkazy se dají do rejstříku na dvě místa -->
<xsl:template match="command" mode="profile">
  <!-- Zkopírování původního elementu -->
  <xsl:copy-of select="."/>
  <!-- Vytvoření rejstříkových hesel -->
  <indexterm>
    <primary><xsl:value-of select="."/></primary>
  </indexterm>
  <indexterm>
    <primary>příkaz</primary>
    <secondary><xsl:value-of select="."/></secondary>
  </indexterm>
</xsl:template>

<!-- Každé jméno souboru se také přidá do rejstříku -->
<xsl:template match="filename" mode="profile">
  <!-- Zkopírování původního elementu -->
  <xsl:copy-of select="."/>
  <!-- Vytvoření rejstříkového hesla -->
  <indexterm>
    <primary><xsl:value-of select="."/></primary>
  </indexterm>
</xsl:template>

</xsl:stylesheet>
```

Nově připravený styl pak z pohledu uživatele zvládne během jednoho kroku do dokumentu doplnit rejstříková hesla a ještě jej zpracovat (např. převést do FO nebo HTML).

Závěr

V článku jsme se podrobně seznámili s možnostmi generování rejstříku pro dokumenty v DocBooku za použití standardních XSL stylů. Ukázali jsme si, jak generovat rejstříky vyhovující českým zvyklostem, jak do jednoho dokumentu vložit několik rejstříků a jak ze sémantického značkování automaticky generovat rejstříková hesla.

Odkazy

1. Jiří Kosek: *DocBook. Stručný úvod do tvorby a zpracování dokumentů.* URL: `http://www.kosek.cz/xml/db/`
2. Jiří Kosek: *XSL FO a jeho open-source implementace.* Str. 105-113. In: Jan Kasprzak – Petr Sojka: *SLT 2002.* ISBN 80-7302-043-2.
3. Robert Stayton: *DocBook XSL: The Complete Guide.* 2003. ISBN 0-9741521-1-0.
   Str. 394. URL: `http://www.sagehill.net/docbookxsl/index.html`
4. Norman Walsh – Leonard Muellner: *DocBook. The Definitive Guide.* 1999. ISBN 156592-580-7. Str. 648.
   URL: `http://www.docbook.org/tdg/en/html/docbook.html`

Multimediální publikování na DVD: projekt 10@FI[1]

Petr Vopálenský, Petr Sojka

Publikování a šíření multimediálních dat na nosičích DVD je stále častější. U každého většího publikačního projektu však prefabrikovaná řešení obvykle nestačí – je třeba řešení ušít na míru.

Článek diskutuje technologie, formáty a postupy vybrané a odzkoušené při tvorbě DVD připravovaného k desátému výročí Fakulty informatiky MU s názvem *10@FI*. Čtenář se seznámí s průběhem přípravy DVD od prvotní myšlenky a návrhu až po konečné vytvoření (GNU/Linux) bootovatelného obrazu DVD a jeho vylisování. Celý projekt vznikal převážně

[1] Aktualizovaná verze příspěvku prezentovaného na SLT 2004 [6]. Podporováno výzkumným záměrem MSM 143300003.

pod OS Linux s pomocí programů s otevřeným zdrojovým kódem, při širokém využití XML technologií a standardů W3C (desítky spolupracujících, ke stovce spoluautorů, tisíce provázaných souborů, stovky obrázků a fotek, desítky minut původních filmů).

Klíčová slova: DVD, multimédia, elektronické publikování, XML, XSLT, DocBook

Úvod

Multimediální prezentace na médiích CD a DVD jsou dnes standardním prostředkem šíření informací. Kvalitní příprava velkého multimediálního díla na DVD je obrovský projekt, jehož příprava a realizace zahrnuje široké spektrum činností několika desítek specializovaných profesí.

Po poměrně dobrých zkušenostech s vysoce svépomocnou (studenti a pedagogové FI) přípravou CD *Všech 5 pohromadě* [7] v roce 1999 Fakulta informatiky zvažovala možnosti, jak se zviditelnit u příležitosti svého letošního (2004) desátého výročí. Vývoj informačních technologií pokročil mílovými kroky kupředu a CD nosiče jsou nahrazovány DVD nosiči. Vyvstala otázka, zda si prezentační DVD nechat vyrobit nějakou renomovanou firmou, nebo si akademičtí informatici zase zkusí prakticky zpracovat informace informačními technologiemi sami. První varianta by dozajisté nevypovídala o FI zdaleka tolik, jako druhá. Jelikož byl pokus vypsat zaměření předmětu PV113 *Softwarové elektronické publikace* tematicky zaměřené na praktickou přípravu DVD úspěšný, nestálo realizaci projektu sedmnácti zapsanými studenty pod vedením tří pedagogů nic v cestě: jeden z těchto studentů pod vedením vyučujícího PV113 a navrhovatele projektu DVD tímto článkem o projektu referuje.

Cíle a infrastruktura projektu

Cíle

Úvodem byly vytčeny tyto cíle: DVD má zmapovat desetiletou existenci Fakulty informatiky Masarykovy univerzity, sloužit jako archív studijních materiálů, učebních textů, má prezentovat jednotlivé výzkumné laboratoře i díla studentů, zejména pak dát plastický obrázek o škole zájemcům o studium a začínajícím studentům. Rozhodlo se o nákladu 3000 ks datových DVD typu DVD5 (4,7 GB), začali se shánět sponzoři, ale hlavně se začala řešit technologie přípravy DVD a softwarová podpora redakční práce týmu.

Komunikační prostředky

Je všeobecně známo, že u větších softwarových projektů je klíčové definovat *rozhraní*, a to jak *datová*, tak *komunikační*, a ta pak důsledně používat,

resp. jejich užívání vynucovat. Jako hlavní komunikační nástroj byl kromě elektronické pošty pro kolaborativní sdílení průběžných informací a diskuse přes web zvolen systém Wiki (http://wiki.org). Wiki umožňuje snadno měnit obsah dokumentů, udržovat informace o změnách a uživatelích. Tým se třináct týdnů scházel jednou týdně celý a kromě toho se dle potřeby scházely jednotlivé podtýmy.

Infrastruktura

Jako zázemí pro tvorbu sloužily tyto tři laboratoře FI MU: SITOLA (Laboratoř pokročilých síťových technologií), LEMMA (Laboratoř elektronických a multimediálních aplikací) a hlavně NLP (Laboratoř zpracování přirozeného jazyka), ve které je umístěn hlavní pracovní server. Na projektu pracovala dvacítka stálých členů plus nespočet přispívajících externistů.

Textová část

Konverze z CD *Všech 5 pohromadě*

Vzorem pro naše DVD bylo CD *Všech 5 pohromadě*. Protože jsme na DVD chtěli zachovat původní data a dále je rozšiřovat, museli jsme dokumenty, které byly ve speciálně navrženém metajazyce, konvertovat na námi po rozvaze zvolený DocBook [4, 5] v XML. Díky tomu, že tvůrci CD již v roce 1999 navrhovali primární formát dokumentů rozumně tak, aby byl obsah oddělený od formy, nebylo velkým problémem transformaci značkovaných dat uskutečnit do letos zvoleného formátu plně automatizovaně.

DocBook

Jako nejvhodnější koncový formát dokumentů pro DVD jsme zvolili XHTML. Bylo by však neefektivní vytvářet přímo XHTML dokumenty (např. pro tisk za účelem korektur je vhodnější PDF), proto jsme použili modifikovaný DocBook [8].

Díky DocBooku stačí vytvářet jeden XML soubor, který pak můžeme dále konvertovat do různých formátů (v našem případě XHTML, PDF pro korektory, nebo PostScript). DocBook je formát XML dokumentu a popisuje tedy dokumenty na sémantické úrovni, ne na vizuální. Autor tedy značkuje obsah, ale už neurčuje, jak se má obsah vizualizovat.

Transformace dokumentů

Abychom z XML souboru dostali XHTML, poslouží XSL transformace [2]. XSLT procesor si vezme XML, aplikuje na něj pravidla ze stylu a výsledek uloží v XHTML. Pro vygenerování PDF dokumentu se XML transformuje (XSLT procesorem) na formátovací objekty (FO) [1] soubor, z kterého se FO procesorem vygeneruje požadovaný PDF dokument. Sám XSLT procesor totiž PDF (binární soubor) vygenerovat neumí. Jako FO procesor jsme používali FOP.

K převodu do XHTML posloužil XSLT procesor XsltProc, k validaci XML souborů xmllint.

DocBook také plně podporuje možnosti a konverze do různých výstupních formátů (odkazy, rejstřík...), což je pak „zadarmo“.

Při zpětném pohledu ale zjišťujeme, že modifikace DTD DocBooku nebyla nejlepší volba. Mnohem efektivnější bylo vytvořit vlastní DTD pro dokumenty, vlastní XSLT šablony pro převod do XHTML a DocBooku. Šablona pro převod do DocBooku je důležitá kvůli dalšímu převodu do PDF, jelikož vytvořit si XSLT šablonu pro převod do PDF (přesněji řečeno do FO souboru), není zrovna triviální záležitost.

Metainformace

DVD si zachovalo původní kostru z CD *Všech 5 pohromadě*, tím pádem bylo vhodné vytvořit dva druhy metainformací pro XML soubory – jedny pro kořenové XML kapitol (určovaly strukturu jednotlivých kapitol) a jiné pro samotné dokumenty.

Z kořenových XML souborů jsme generovali obsah kapitol, podkapitol a dokumentů. Díky metainformacím v XML souboru mohli redaktoři lehce zjistit v jakém stavu se dokument (nebo kapitola) nachází, jestli k němu existuje cizojazyčná verze, zda již byla provedena klasická tištěná korektura textu a zda jsou v pořádku a vyřízená autorská práva.

Vzor DTD pro kořenové XML soubory:

```
<!-- metainformace, indexy jednotlivych kapitol -->

<!ELEMENT kapitola (nazev, (kapitola*, dokument*)*)>
  <!ATTLIST kapitola
        video CDATA #IMPLIED
        soubor CDATA #IMPLIED
        prekladat (ano | ne) #IMPLIED
        autorska-prava (ano | ne) #IMPLIED
        korektura (ano | ne) #IMPLIED
        kotva CDATA #IMPLIED
        popis CDATA #IMPLIED>
<!ELEMENT nazev (#PCDATA)>
<!ELEMENT dokument (spatny-odkaz*)>
  <!ATTLIST dokument
        soubor CDATA #REQUIRED
        prekladat (ano | ne) #IMPLIED
        autorska-prava (ano | ne) #IMPLIED
        korektura (ano | ne) #IMPLIED
        kotva CDATA #IMPLIED
        popis CDATA #IMPLIED
        dontshow (ano|ne) #IMPLIED>
<!ELEMENT spatny-odkaz (#PCDATA)>
```

Protože nám možnosti standardního DocBooku nestačily, přidali jsme do DTD DocBooku nové elementy. Tyto elementy sloužily k identifikaci z jakého zdroje byl dokument získán, kdo je autorem dokumentu, kdo za dokument zodpovídá atd.

Ukázka z modifikováného DTD pro dokumenty:

```
<!-- metainformace umistene v~hlavicce dokumentu -->

<!ELEMENT autor EMPTY>
  <!ATTLIST autor id CDATA #REQUIRED>
<!ELEMENT zdroj (#PCDATA)>
<!ELEMENT odpovedna-osoba EMPTY>
  <!ATTLIST odpovedna-osoba id CDATA #REQUIRED>
<!ELEMENT datum EMPTY>
  <!ATTLIST datum od CDATA #REQUIRED
                  do CDATA #IMPLIED>
<!ELEMENT poznamka (#PCDATA)>
```

Pro jednoduchý zápis jmen je vhodné udržovat seznam osob a odkazovat na něj pomocí jednoznačného id osoby. To umožní například odlišit různé Nováky v autorském rejstříku ap. Ukázka DTD `metalide.dtd`:

```
  <!ELEMENT lide (osoba*)>
    <!ELEMENT osoba
              (jmeno, prijmeni, titul?, foto?, popisek?)>
    <!ATTLIST osoba id ID #REQUIRED>
      <!ELEMENT jmeno (#PCDATA)>
      <!ELEMENT prijmeni (#PCDATA)>
      <!ELEMENT titul EMPTY>
      <!ATTLIST titul pred CDATA #IMPLIED
                      za CDATA #IMPLIED>
      <!ELEMENT foto (#PCDATA)>
      <!ELEMENT popisek (#PCDATA)>
```

a takto bude vypadat část XML souboru `lide.xml`, která ho využívá:

```
<?xml version="1.0" encoding="iso-8859-2"?>
<!DOCTYPE lide SYSTEM "metalide.dtd">
<lide>
  <osoba id="LMatyska">
    <jmeno>Luděk</jmeno>
    <prijmeni>Matyska</prijmeni>
    <titul pred="Doc. RNDr." za="CSc." />
    <foto>f10://602342354567687978</foto>
    <popisek>Děkan FI</popisek>
  </osoba>
  ...
<lide/>
```

V podobném duchu jsme metainformace vytvářeli, validovali a používali i pro videa a fotky. Do metainformací se ukládají informace o stavu dokumentu (korektury, autorská práva, jedinečné id objektu...). Díky metadatům se pak snadno vytvářejí rejstříky (osob, videa, fotek a tématický rejstřík).

Kontrola odkazů

Kontrola odkazů je realizována v jazyce Java, která má velmi dobrou podporu jazyka XML. Bohužel velkou nevýhodou Javy je její pomalost. Kontrola odkazů se provádí rekurzivním průchodem stromové struktury XML souborů od kořene. Lze zaznamenat počet odkazovaných souborů, počet přebytečných souborů, počet chybějících souborů a dokonce rozeznávat chybějící a nadbytečné soubory podle typu (obrázky, video, dokumenty…).

Grafická část

Design

Vzhled XHTML stránek zajištují kaskádové styly (CSS) a díky JavaScriptu lze vzhled jednoduše přepínat (pouze se mění soubor s kaskádovými styly) – čtenář má na výběr z několika vzhledů. Jelikož je forma dokumentu skriktně oddělena od obsahu (součástí XHTML kódu nejsou žádné tagy určující vzhled), má úprava jednoho CSS souboru vliv na vzhled všech XHTML stránek na DVD.

Logotyp, obal a potisk

Již od začátku jsme promýšleli ucelený a jednotný vzhled celého DVD, kam samozřejmě spadá i logotyp, obal a potisk. Bylo navrženo více grafických variací od různých autorů a více názvů pro DVD. O vítězném návrhu se demokraticky hlasovalo. Výsledný vzhled logotypu a návrh názvů pro DVD jsou na obrázku 1.

Obrázek 1: Logotyp projektu DVD *10@FI* a návrhy názvu DVD

Multimediální část

Videomedailonky

Díky prostorovým možnostem nosiče DVD mohl multimediální podtým vytvořit videomedailonky jednotlivých laboratoří fakulty v délce několika desítek minut. Medailonků je devět. Jsou to kompletně autorská díla studentů od námětu, scénáře, režie, ozvučení a střihu až po finální produkci. Jeden ze studentů navíc vymodeloval a „oživil“ virtuální hlasatelku, která některé z medailonků uvádí. Dalším zpestřením jsou ukázky z každoročně pořádaného filmového festivalu studentských filmů. Čtenáře Zpravodaje asi zaujme videozáznam udílení titulu *doctor honoris causa* Masarykovy university autorovi TEXu Donaldu Ervinu Knuthovi.

Jako komprimační kodeky byly použity XviD verze 1.0 pro video a MP3 pro zvukovou stopu. XviD kodek je pod licencí GPL a licenční podmínky u nejznámějšího audio kodeku MP3 jsou v projektu splněny. U původně zvažovaného Ogg Vorbis se objevily problémy s přehráváním v komerčních rozšířených přehrávačích videa. Střih videa probíhal převážně na profesionálním střihacím systému Avid XPress Pro. Pro finální zkompletování zvuku s videem, deinterlacing a kompresi dokonale postačoval volně šířený program Virtual Dub.

Na DVD se nachází celkem přes hodinu autorského videa ve vysoké kvalitě.

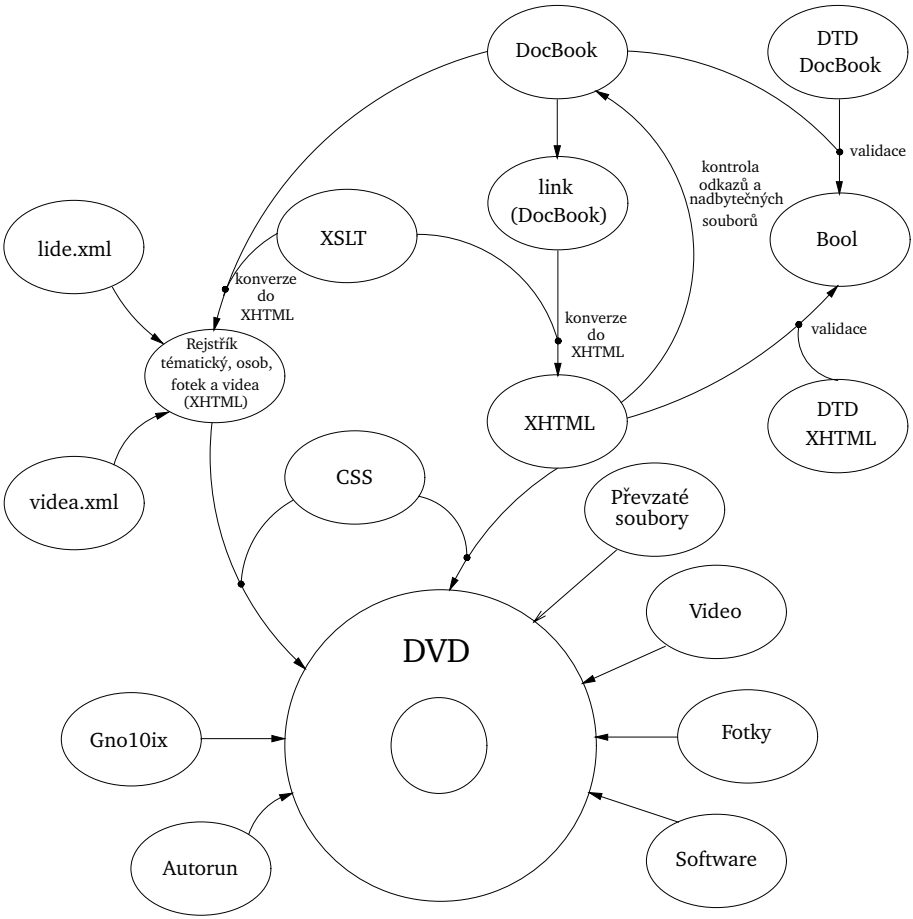
Generování DVD

Výsledný proces vygenerování ISO obrazu DVD se skládá z téměř desítky kroků, které jsou znázorněny na obrázku 2. Zdrojový XML soubor se zvaliduje proti DTD DocBooku. Pokud je validní, nahradí se v něm symbolické odkazy za skutečné a soubor se uloží s příponou link (stále se jedná o validní DocBookové XML). Zároveň se vytvoří rejstříky osob, fotek, videí a rejstřík tématický. Pak se z link-souborů a rejstříků vygenerují XSLT procesorem XHTML soubory, které se opět podrobí validaci (samozřejmě, že již ne oproti DTD DocBooku, ale oproti DTD XHTML). Nad XHTML soubory se provede kontrola odkazů a nadbytečných souborů – chybějící a nadbytečné soubory se vypíší na obrazovku, editor musí doopravit ručně. Výsledné XHTML soubory spolu s fotkami, videi, zrcadly stažených dokumentů a stránek, autorunem, softwarem (viz „Software na DVD“) a Gno10ixem (viz „Bootovatelnost média“) jsou již připraveny k prezentaci na DVD. Vše je automatizováno s pomocí GNU Make.

Ostatní

Správa verzí

Projekty, na nichž pracuje mnoho lidí, je potřeba nějak zabezpečit proti vzájemnému přepisování souborů, zajistit možnost navracení k předchozím verzím



Obrázek 2: Tok dat při generování DVD

atd. My jsme volili z osvědčeného CVS a v té době nového SubVersion. CVS má dva hlavní nedostatky: 1) neumí přejmenovat soubor bez toho, aniž bychom ztratili informaci o logu souboru a 2) binární soubory přidávané do repozitáře musí být ručně označeny speciálním parametrem, jinak může dojít k jejich poškození. My se proto rozhodli použít SubVersion, který plně nahrazuje CVS, a má tyto problémy řešit. Bohužel jsme během nasazení SubVersion narazili na problém s náhodným padáním verzovacího systému (mylně přisuzovaný tomu, že běžel na víceprocesorovém stroji) a jelikož jsme byli v časové tísni a nebyl čas problém řešit, vrátili jsme se k CVS.

SubVersion nám padalo z důvodu špatně nastavené masky (022 místo 002), proto se musí binární soubory svn a svnserve nahradit skripty, které nejdřív masku nastaví a pak spustí samotné binární svn či svnserve.

Systém správy verzí neřešil pouze výše zmiňované, ale zajišťoval přes commit skripty antivirovou kontrolu posílaných souborů a kontrolu validity XML souborů. Více o verzovacích systémech čtenář nalezne v [9].

Bootovatelnost média

Pro použití DVD není zapotřebí mít nainstalován operační systém, DVD je totiž bootovatelné. Je založeno na linuxové live distribuci Gnoppix (a ta dále na Debian GNU/Linux), pojmenované Gno10ix. V distribuci je předinstalované grafické pracovní prostředí GNOME a veškeré potřebné programové vybavení. Naše live distribuce zabírá cca 400 MB prostoru, to není málo, což vyvažuje uživatelskou přítulností a nejnovějšími verzemi programů v ní obsažených. Více čtenář nalezne v samostatném článku ve sborníku [3]. Zkušenosti z téměř půlročního života DVD mezi čtenáři ukazují, že podpora bootování z DVD na různém hardwade je dosud nízká a i z tohoto úhlu pohledu projekt mírně předběhl dobu. Na novém hardware jsou zase občas reportovány problémy s autodetekcí grafických karet ap.

Betatestování

Při takto rozsáhlém projektu je důležité celý projekt řádně otestovat, nejlépe někým, kdo nemá s výrobou samotného DVD nic společného – takový člověk najde často nejvíce chyb. Proto byly průběžně páleny betaverze DVD na DVD-RW a půjčovány dobrovolníkům k testování. Tímto způsobem bylo zjištěno mnoho chyb, na které bychom přicházeli jen velice stěží nebo vůbec. Navíc byla vždy aktuální verze DVD dostupná i přes WWW, takže se na testování mohlo podílet mnohem více lidí, než jen těch několik, kteří měli půjčenou betaverzi DVD.

Software na DVD

Protože ne každý má ve svém počítači nainstalovaný potřebný software, umístili jsme na DVD vše, co je k práci s DVD třeba. Každý software je na DVD uložen ve verzi pro MS Windows a GNU/Linux, ať už jde o oblíbený prohlížeč Mozilla Firefox, Adobe Reader, nebo kodeky XviD pro video a Ogg Vorbis pro zvuk. Tyto programy jsou volně šiřitelné a s volně dostupnými zdrojovými kódy (vyjma Adobe Reader).

Uživatelům systému MS Windows automatizuje detekci software na DVD obsažený autorun, který se snaží zjistit nainstalované součásti systému a pokud nějaký potřebný software chybí, nebo je v systému zastaralá verze, upozorní čtenáře a nabídne mu instalaci software potřebného pro plnohodnotnou práci s obsahem DVD.

Shrnutí a zkušenosti

Příprava finálního produktu jako součást výuky u takovýchto velkých projektů v akademickém prostředí s sebou přináší problém s dodržováním termínů, na druhou stranu se ovšem výsledný produkt téměř rovná komerčním s využitím minimálního rozpočtu. Proto bylo důležité dělat časté týmové schůze, kontroly kvality. Jelikož realizační tým byl značně různorodý, adekvátní přidělování práce a zodpovědností studentům podle jejich znalostí, možností, chuti se učit nové věci a objevovat dosud nepoznané bylo netriviálním a důležitým úkolem vedoucích projektu. Nezbytné bylo vytvoření prostředí a atmosféry pro komunikaci mezi jádrem realizačního týmu a akademickou obcí FI. Pro zajištění součinnosti některých laboratoří a osob byla nezbytná podpora vedení FI. Informovanost veřejnosti a akademické obce byla zajišťována vydáváním tiskových zpráv zveřejňovaných na portálu projektu `http://10.fi.muni.cz`.

Bylo by nemyslitelné DVD vytvořit bez vysokého stupně automatizace pomocí Makefile. Dosažení potřebné kvality bylo možné díky speciálně vyvinutým co nejvíce automatizovaným i ručním procedurám (kontroly odkazů, konverze dokumentů,...).

Celé DVD jsme se snažili vytvářet za pomoci open source programů, a až na výjimky se nám to povedlo. Pro projekt se využilo vybavení stávajících a existujících laboratoří FI, nebyl nakupován speciálně žádný software ani hardware.

Výrobní cena jednoho kusu DVD včetně DVD krabičky a tisku obalu byla 35 Kč. S podporou vedení FI MU se podařilo celý projekt financovat *pouze* z peněz vybraných od sponzorů projektu a tedy všichni zájemci o [studium na] FI dostávají DVD *zdarma*.

O kvalitách výsledného produktu svědčí i udělení ceny České společnosti pro systémovou integraci 4. ročníku soutěže eLearning, konané v rámci 42. ročníku mezinárodního festivalu Techfilm 2004 v Hradci Králové.

Poděkování

Jak naznačeno, na projektu se podílelo autorsky kolem stovky osob, produkčně přes dvacet. Je mimo možnosti tohoto reportu vyjmenovat zásluhy každého zvlášť. Proto aspoň paušálně děkujeme všem, kteří jakoukoli měrou a svou vstřícností přispěli k realizaci projektu.

Odkazy

1. Sharon Adler, Anders Berglund, Jeff Caruso, Stephen Deach, Tony Graham, Paul Grosso, Eduardo Gutentag, Alex Milowski, Scott Parnell, Jeremy Richman, a Steve Zilles.
   Extensible Stylesheet Language (XSL), 2001.
   `http://www.w3.org/TR/xsl`.

2. James Clark.
XSL Transformations (XSLT), 1999.
`http://www.w3.org/TR/xslt`.
3. Marcel Kolaja.
Live distribuce aneb Linux na CD/DVD.
V *Sborník SLT 2004*, strany 203–212, Znojmo, 2004. Konvoj.
4. Jiří Kosek.
DocBook – stručný úvod do tvorby a zpracování dokumentů.
`http://www.kosek.cz/xml/db/`.
5. Jiří Kosek.
XML pro každého – podrobný průvodce.
Grada Publishing, 2000.
6. Petr Vopálenský a Petr Sojka.
Multimediální publikování na DVD: projekt 10@FI.
V *Sborník SLT 2004*, strany 21–29, Znojmo, 2004. Konvoj.
7. Všech pět pohromadě.
CD-ROM, `http://nlp.fi.muni.cz/projekty/vsech5/`, září 1999.
8. Norman Walsh a Leonard Muellner.
DocBook: The Definitive Guide.
O'Reilly & Associates, 1999.
9. Milan Zamazal.
GNU arch, systém pro správu verzí.
V *Sborník SLT 2004*, strany 127–139, Znojmo, 2004. Konvoj.

Novinky v OFS

Petr Olšák

OFS (Olšákův fontový systém) byl již na SLT prezentován. Během roku 2004 byl ale OFS pro plain významě přepracován a byly do něj přidány nové vlastnosti, které umožňují vytvářet on-line katalogy fontů nebo důkladné testy fontových rodin včetně kombinace s matematickou sazbou. Rovněž byly rozšířeny možnosti práce s makry závislými na kódování a implementována podpora TX fontů. Na přednášce budou v krátkosti tyto nové vlastnosti OFS předvedeny posluchačům.

Úvodem

Makro OFS jsem poprvé zveřejnil v roce 2001 a na SLT v Seči jsem toto makro prezentoval [3]. Pak se s OFS delší dobu nic nedělo. Teprve letos jsem se pokusil vytvořit pro OFS jednak interaktivní makro na testy celých fontových

rodin a dále jsem začal pracovat na OkTEXu [4, 5], kde jsem se rozhodl OFS pro plain využít.

Tyto dva cíle mě donutily udělat v OFS několik docela zásadních změn. Změny se týkají výhradně verze OFS pro plain. Na LATEXovou verzi jsem vůbec nesáhl. Neznamená to ale, že by LATEXový uživatel neměl z této práce žádný užitek. Může si pomocí příkazu `csplain ofstest` interaktivně vytvářet vzorníky fontů. Myslím si, že v tomto případě mu může být jedno, jakým formátem je ten vzorník zpracován.

Interaktivní makro `ofstest.tex`

Po spuštění `csplain ofstest` nebo `tex ofstest` si program vyžádá vložení názvu definičního souboru. Například `a35` nebo `slido`. Pokud rovnou na příkazový řádek za slovo `ofstest` a za mezeru zapíšeme definiční soubor(y) do hranaté závorky, pak program už tento údaj nežádá znovu. Dále můžeme vybrat rodinu fontů, kterou chceme testovat. Potom lze příkazem `\help` vypsat další možnosti, co můžeme dělat. Komunikace může vypadat například takto (údaje, které vložil uživatel, jsou vyznačeny rámečkem):

```
$ csplain ofstest [ffonts]
This is TeX, Version 3.14159 (Web2C 7.3.7x)
encTeX v. Feb. 2003, the reencoding enabled.
(/usr/TeX/texmf/tex/ofs/ofstest.tex The format: csplain <Feb. 2000>.
The cs-fonts are preloaded and A4 size implicitly defined.
(/usr/TeX/texmf/tex/ofs/ofs.tex
OFS (Olsak's Font System) based on plain initialized. <May 2004>
(/usr/TeX/texmf/tex/ofs/ofsdef.tex))
Czech hyphenation used (\language=5). \frenchspacing is set on.
(/usr/TeX/texmf/tex/ofs/ofs-8z.tex) (/usr/TeX/texmf/tex/ofs/ofs-8c.tex))
(/usr/TeX/texmf/tex/ofs/ffonts.tex) This is ofstest macro, version <May. 2004>

*** Type family name without brackets (or ? or *): Charter

*** What to do with family Charter ?
    (type command or \help): \help
commands:
 \list ..... List all variants of the family Charter
 \table(s) . Tables of all variants of the family Charter
 \abet ..... One line alphabet/digits sample for each variant
 \chars .... Print list of available characters including TeX sequences
 \text ..... One paragraph in all variants of the family Charter
 \mixed .... Paragraph with fonts combined from \rm, \bf, \it and \bi
 \math ..... Mathematics text combined by fonts from Charter
 \all ...... The same as \list \table \abet \chars \text \mixed \math
 \setsize .. Set size of fonts (current size is "at10pt")
 \cfam ..... Change current family
 FamName ... The same as \cfam FamName
 \rem ...... Remove current family or family given in parameter from \famlist
```

```
 ------------------------------
 \famlist .. Show list of all declared families (the same as \showfonts)
 \decl ..... Input next declaration file
 \remdecl .. Remove all families of given declaration file from famlist
 \help ..... This text
 \morehelp . Show more help information
 \fontusage  The help screen of the OFS
 \end ...... End of this session

*** What to do with family Charter ?
    (type command or \help): \abet \chars \mixed \math \end

 [Charter/at10pt]: \rm abet (/usr/TeX/texmf/tex/ofs/ffonts.tex) \bf abet
\it abet \bi abet
 [Charter/]: \chars (/usr/TeX/texmf/tex/ofs/ofs-8z.tex)
 [Charter/at10pt]: mixed text (/usr/TeX/texmf/tex/ofs/ofs-ps.tex)
 [Charter/at10pt] (\fomenc: PS) [10.0pt/7.0pt/5.0pt]: math text
[1]
Output written on ofstest.dvi (1 page, 7492 bytes).
Transcript written on ofstest.log.
```

V tomto příkladě jsme vybrali rodinu Charter (shodou okolností je touto rodinou fontů dělán sborník SLT) a vypsali jsme si pomocí `\help` možnosti, co můžeme podniknout dále. Z nabídky jsme pak vybrali pouze tisk zkrácené abecedy (`\abet`), výpis dostupných znaků v této rodině včetně jejich TEXových sekvencí (`\chars`), tisk vzorku, kde je kombinován základní řez s tučným a s kurzívou (`\mixed`), a konečně tisk vzorku s matematickou sazbou (`\math`). Výsledek testu je vidět na obrázku 1.

Obrázek je v tomto sborníku mírně zmenšený, takže údaje po stranách (že se jedná o 10bodové písmo) nejsou správné. Samozřejmě, originální výstup z makra `ofstest.tex` je ve správné velikosti a údaje o rozměrech souhlasí.

Funkci všech příkazů, které jsou v interaktivním prostředí k dispozici, nebudu podrobně rozebírat. Stručný význam těchto příkazů je vytištěn na terminál pomocí příkazu `\help` a je zahrnut v předchozí ukázce.

Před tiskem můžeme měnit interaktivně některé parametry. Například pomocí `\def\fotenc{8t}` dáme najevo, že chceme testovat fonty v kódování `8t`. Pomocí příkazu `\setsize` můžeme zadat jiný základní rozměr fontu než `10pt` a příkazem `\baselineskip=...` můžeme měnit velikost řádkování. Konečně příkazem `\def\textlang{en}` dáváme najevo, že chceme vzorky v anglickém textu a nikoli českém.

Výchozí hodnoty makra `ofstest.tex` jsou závislé na tom, zda je použit formát `csplain` nebo formát `plain`. V prvním případě je výchozím jazykem ukázek čeština a výchozí kódování je `8z` (alias IL2). Při formátu `plain` je výchozím jazykem angličtina a kódování je implicitně nastaveno na `8t` (podle Corku).

[Charter/10pt] (enc: 8z), declared in ffonts.tex:

() \rm ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžščěáéťůÚ, acbdefghijklmnopqrstuvwxyz

(Bold) \bf **ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžščěáéťůÚ, acbdefghijklmnopqrstuvwxyz**

(Italic) \it *ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžščěáéťůÚ, acbdefghijklmnopqrstuvwxyz*

(BoldItalic) \bi ***ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžščěáéťůÚ, acbdefghijklmnopqrstuvwxyz***

Charter (8z) \` *: ̀ \' *: ́ \v *: ̌ \u *: ̆ \= *: ̄ \r *: ̊ \^ *: ̂ \. *: ̇ \H *: ̋ \~ *: ̃ \" *: ̈ \c *: ̧ \k *: ̨ \m *: ̛ \b *: ̲ \d *: ̣ \dotlessi: ı \dotlessj: ȷ \" i: ï \" \i: ï \SS: SS \AE: Æ \OE: Œ \O: Ø \ss: ß \ae: æ \oe: œ \o: ø \ellipsis: … \promile: ‰ \varhyphen: - \flqq: « \frqq: » \clqq: „ \crqq: “ \clq: ‚ \crq: ‘ \elqq: “ \erqq: ” \elq: ‘ \erq: ’ \exclamdown: ¡ \questiondown: ¿ \dag: † \ddag: ‡ \section: § \paragraph: ¶ \ellipsis: … \textbullet: • \sterling: £ \currency: ¤ \Lslash: Ł \lslash: ł \Eth: ? \eth: ? \texttimes: ? \textdiv: ? \' A: À \' A: Á \" A: Ä \' a: à \' a: á \" a: ä \v C: Č \v c: č \v D: Ď \v d: ď \' E: É \v E: Ě \' e: é \v e: ě \' I: Í \' i: í \' \i: í \v L: Ľ \' L: Ĺ \v l: ľ \' l: ĺ \v N: Ň \v n: ň \' O: Ó \^ O: Ô \" O: Ö \' o: ó \^ o: ô \" o: ö \' R: Ŕ \v R: Ř \' r: ŕ \v r: ř \v S: Š \v s: š \v T: Ť \v t: ť \r U: Ů \' U: Ú \" U: Ü \r u: ů \' u: ú \" u: ü \' Y: Ý \' y: ý \v Z: Ž \v z: ž \k A: Ą \^ A: Â \u A: Ă \k a: ą \^ a: â \u a: ă \' C: Ć \c C: Ç \' c: ć \c c: ç \k E: Ę \" E: Ë \k e: ÿ \" e: ë \^ I: Î \^ i: î \^ \i: î \' N: Ń \' n: ń \H O: Ő \H o: ő \' S: Ś \c S: Ş \' s: ś \c s: ş \c T: Ţ \c t: ţ \H U: Ű \H u: ű \' Z: Ź \. Z: Ż \' z: ź \. z: ż

Modifications { 8z:charter }

Charter 10pt/12.0pt

Příklad. Nyní zjistíme, zda je možné kombinovat základní řez s tučným (resp. polotučným) řezem a s kurzívou v rodině písma *Charter*. Pro vyznačování je vhodné použít *kurzívu* a už méně **tučnou variantu** a prakticky vůbec se nehodí použít ***tučnou a skloněnou variantu***. Zcela nevhodné je vyznačovat podtržením nebo prostrkáním. To lidé s dobrým vychováním nedělají. *Vyznačení má být takové, aby při čtení bylo vyznačené místo zřetelně odlišné, ale při pohledu z dálky zůstal text odstavce stejnoměrně šedý.* Tomu nejlépe vyhovuje kurzíva, ale ne ve všech rodinách písma je zdařilá kuzíva k dispozici.

Charter (PX) [10.0pt/7.0pt/5.0pt]

Poznámka. Funkci Gamma v bodě x značíme $\Gamma(x)$ a počítáme ji podle vzorce:

$$\Gamma(x) = \int_0^\infty e^{-t} t^{x-1} \mathrm{d}t \quad (x > 0).$$

Speciálně pro $x = n \in \mathbb{N}$ je $\Gamma(n) = (n-1)!$ a pro $\alpha \in (0,1)$ je

$$\Gamma(\alpha)\,\Gamma(1-\alpha) = \frac{\pi}{\sin \pi\alpha}.$$

Definice. Nechť $\boldsymbol{A}$ je čtvercová matice s n sloupci a řádky a s prvky $a_{i,j}$. Pak číslo

$$\det \boldsymbol{A} = |\boldsymbol{A}| = \sum_{\Pi=(j_1,j_2,\ldots,j_n)} \operatorname{sgn} \Pi \cdot a_{1,j_1}\, a_{2,j_2} \cdots a_{n,j_n}$$

nazýváme *determinantem* matice $\boldsymbol{A}$.

Obrázek 1. Výstup z `ofstest.tex`. Charter, příkazy: `\abet`, `\chars`, `\mixed`, `\math`

Makro `ofstest.tex` nemusí testovat jen jedinou rodinu. Napíšeme-li místo jména rodiny hvězdičku, provede se pak zvolený příkaz na všechny rodiny, které jsou načteny v deklaračních souborech. Seznam těchto rodin je dostupný pomocí známého příkazu OFS `\showfonts` nebo příkazem `\famlist`. Makro dále umožní za chodu načítat další deklarační soubory (příkaz `\decl`) a ze seznamu rodin vymazávat některé rodiny. Až nám po takové činnosti zůstanou v seznamu

[AntykwaTorunska/10pt] (enc: 8z), declared in pantyk.tex:

(Regular) \rm ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščéáéřů, acbdefghijklmnopqrstuvwxyz

(Bold) \bf **ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščéáéřů, acbdefghijklmnopqrstuvwxyz**

(Italic) \it *ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščéáéřů, acbdefghijklmnopqrstuvwxyz*

(BoldItalic) \bi ***ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščéáéřů, acbdefghijklmnopqrstuvwxyz***

(Light) \lr ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščéáéřů, acbdefghijklmnopqrstuvwxyz

(LightItalic) \li *ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščéáéřů, acbdefghijklmnopqrstuvwxyz*

(Medium) \mr **ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščéáéřů, acbdefghijklmnopqrstuvwxyz**

(MediumItalic) \mi ***ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščéáéřů, acbdefghijklmnopqrstuvwxyz***

[AntykwaTorunskaCaps/10pt] (enc: 8z), declared in pantyk.tex:

(Regular) \rm ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščÉÁÉŤŮ, ACBDEFGHIJKLMNOPQRSTUVWXYZ

(Bold) \bf **ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščÉÁÉŤŮ, ACBDEFGHIJKLMNOPQRSTUVWXYZ**

(Italic) \it *ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščÉÁÉŤŮ, ACBDEFGHIJKLMNOPQRSTUVWXYZ*

(BoldItalic) \bi ***ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščÉÁÉŤŮ, ACBDEFGHIJKLMNOPQRSTUVWXYZ***

(Light) \lr ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščÉÁÉŤŮ, ACBDEFGHIJKLMNOPQRSTUVWXYZ

(LightItalic) \li *ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščÉÁÉŤŮ, ACBDEFGHIJKLMNOPQRSTUVWXYZ*

(Medium) \mr **ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščÉÁÉŤŮ, ACBDEFGHIJKLMNOPQRSTUVWXYZ**

(MediumItalic) \mi ***ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščÉÁÉŤŮ, ACBDEFGHIJKLMNOPQRSTUVWXYZ***

[AntykwaTorunskaCond/10pt] (enc: 8z), declared in pantyk.tex:

(Regular) \rm ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščéáéřů, acbdefghijklmnopqrstuvwxyz

(Bold) \bf **ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščéáéřů, acbdefghijklmnopqrstuvwxyz**

(Italic) \it *ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščéáéřů, acbdefghijklmnopqrstuvwxyz*

(BoldItalic) \bi ***ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščéáéřů, acbdefghijklmnopqrstuvwxyz***

(Light) \lr ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščéáéřů, acbdefghijklmnopqrstuvwxyz

(LightItalic) \li *ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščéáéřů, acbdefghijklmnopqrstuvwxyz*

(Medium) \mr **ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščéáéřů, acbdefghijklmnopqrstuvwxyz**

(MediumItalic) \mi ***ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščéáéřů, acbdefghijklmnopqrstuvwxyz***

[AntykwaTorunskaCondCaps/10pt] (enc: 8z), declared in pantyk.tex:

(Regular) \rm ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščÉÁÉŤŮ, ACBDEFGHIJKLMNOPQRSTUVWXYZ

(Bold) \bf **ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščÉÁÉŤŮ, ACBDEFGHIJKLMNOPQRSTUVWXYZ**

(Italic) \it *ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščÉÁÉŤŮ, ACBDEFGHIJKLMNOPQRSTUVWXYZ*

(BoldItalic) \bi ***ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščÉÁÉŤŮ, ACBDEFGHIJKLMNOPQRSTUVWXYZ***

(Light) \lr ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščÉÁÉŤŮ, ACBDEFGHIJKLMNOPQRSTUVWXYZ

(LightItalic) \li *ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščÉÁÉŤŮ, ACBDEFGHIJKLMNOPQRSTUVWXYZ*

(Medium) \mr **ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščÉÁÉŤŮ, ACBDEFGHIJKLMNOPQRSTUVWXYZ**

(MediumItalic) \mi ***ABCDEOPQRSVWXYZ 0123456789 &%?!-ŽžŠščÉÁÉŤŮ, ACBDEFGHIJKLMNOPQRSTUVWXYZ***

Obrázek 2. Výstup z `ofstest.tex`. Katalog volně dostupných rodin AntykwaTorunska

`\listfam` jen rodiny, které chceme podrobit testu, pak teprve použijeme příkazy `\abet`, `\text`, `\mixed` a další.

Na obrázku 2 je výstup, který se dá využít jako vzorníček fontů. Zvolil jsem deklarační soubor `pantyk` (Polish Antykwa) a příkazem `\remdecl defaults` jsem odstranil z `\listfam` implicitní rodiny OFS. Zůstaly tam jen rodiny Antykwy Toruńské, z nichž jsem si udělal vzorníček příkazem `\abet`. Takové vzorníčky stovek rodin, které mám na počítači k dispozici, jsem si vytiskl a vybírám z nich písma podle aktuální potřeby (vzorníčky nechám kolovati).

Všimněme si, že OFS pro plainTEX si udržuje v paměti informace o původních názvech jednotlivých variant fontů. V uvedené ukázce vidíme varianty `Light`, `LightItalic`, `Medium` a `MediumItalic`. Ve vzorníčku máme také uvedeny příkazy na přepnutí do zvolené varianty.

Závěrem bych chtěl poznamenat, že interaktivní makro `ofstest.tex` se dá použít i neinteraktivním způsobem. V plainTEXovém dokumentu můžeme napsat například:

```
\input ofstest [a117]
\cfam Bodoni        \tables \text \mixed
\cfam LetterGothic  \tables \text \mixed
⟨...atd.⟩
```

Makro samo pozná, že je zavoláno v neinteraktivním režimu a potlačí nutkání klást jakékoli dotazy.

Výjimky z kódování pro každou rodinu fontů

Ukazuje se, že každá rodina fontů je vybavena mírně odlišnou sadou znaků. Tu chybí netečkované j, tam zas je navíc k dispozici Euro atd. Není rozumné pro každou takovou výjimku vytvářet kódování s novým názvem. OFS raději pracuje s pořád stejným kódováním, ale je nově schopno registrovat všechny výjimky z kódování jednotlivých rodin a udržovat si podle toho představu, které znaky jsou momentálně k dispozici a které ne. Myslím si, že touto vlastností například NFSS z LATEXu nedisponuje.

Za základ pro kódování `8z` jsem vzal encoding vektor `xl2.enc`, který vytvořil pan Wagner. Vzhledem k tomuto kódování pak každá rodina fontů může mít nějaké výjimky. Třeba rodině Charter chybí znaky `\dotlessj`, `\Eth`, `\eth`, `\textimes` a `\texdiv`. Je to názorně vidět na výpisu při testu příkazem `\chars` (obrázek 1). Naopak třeba rodina fontů Lido má navíc (proti základnímu kódování `8z`) znaky `\degree`, `\euro`, `\trademark` atd.

V deklaračních souborech můžeme výjimky seskupit do skupin příkazem `\modifydef`, jednotlivé výjimky pak tam daklarujeme pomocí `\characterdef`, `\characterdel`, `\accentdef` a `\accentdel` a konečně jednotlivým rodinám můžeme přidělit určité skupiny výjimek příkazem `\modifyenc`. Programátor maker se může ptát na existenci znaku ve fontu příkazem `\knowchar`. Podrobněji je o těchto příkazech pojednáno v dokumentaci [2].

Například rodina Charter má pro kódování `8z` přidělenu skupinu výjimek s názvem `8z:charter`. O této skutečnosti nás rovněž informuje výpis `\chars` na obrázku 1 v odstavci `Modifications`. Kdyby tato skupina výjimek obsahovala znaky navíc, byly by v tomto odstavci vykresleny. Výpis `\chars` ovšem zatajuje, že nejsou dosažitelné některé akcentované znaky. Například v rodině Charter chybí znak ű. Proto je pro něj deklarována výjimka pomocí `\accentdel`. Jakmile je výjimka registrována, OFS automaticky začne realizovat sekvenci „`\H u`“ pomocí implicitního akcentu, takže i tento znak nám bude fungovat (ačkoli bude v TEXu složen ze dvou znaků pomocí primitivu `\accent`).

Automatické čtení kódovacích souborů

Aby mohl OFS udržovat přehled o kontrolních sekvencích pro jednotlivé znaky a tím také o výjimkách z kódování, musí mít načteny odpovídající soubory `ofs-`⟨*kódování*⟩`.tex`. Už ve verzi z roku 2001 tyto soubory existovaly, ale uživatel si je načítal podle vlastního uvážení sám příkazem `\input`. Protože se v OkTEXu bude pracovat s množstvím různých kódování a makra mezi těmito kódováními budou sofistikovaně a mnohdy automaticky přepínat, rozhodl jsem se OFS vybavit možností automatického načítání souborů `ofs-`⟨*kódování*⟩`.tex` v okamžiku, kdy to je nutné.

Abych neiritoval přílišnou inteligencí makra OFS ty uživatele, kteří mají rádi konzervativní přístup a raději si soubory načtou sami, je implicitně automatické načítání kódovacích souborů vypnuto. Uživatel si může tuto funkci zapnout pomocí příkazu `\loadingenc=1`. Teprve po takovém příkazu máme zaručeno, že OFS udržuje přehled o TEXových sekvencích vztahujících se ke kódování použitých fontů.

Automatické načtení souboru `ofs-`⟨*kódování*⟩`.tex` může proběhnout ve velmi nepříznivé době: například v horizontálním módu, uvnitř skupiny nebo dokonce ve stavu, kdy má uživatel nastaveny `\catcode` speciálním způsobem. Aby přesto proběhlo načtení souboru bezkonfliktně, rozhodl jsem se toto načtení řešit následujícími kroky:

- Před načtením kódovacího souboru založí OFS novou skupinu.
- Pro jistotu v této skupině nastaví všem znakům standardní kategorie a znaku `@` nastaví kategorii 11.
- Nastaví `\endlinechar=-1`, takže další čtení ze souboru ignoruje prázdné řádky (neprovede se tedy žádné nadbytečné `\par`) a ignoruje mezery, které vznikají na koncích řádků (nedochází k zavlečení mezer v horizontálním módu).
- Provede `\globaldefs=1`, aby všechny následující definice z kódovacího souboru zůstaly zachovány i po opuštění skupiny.
- Načte kódovací soubor.
- Ukončí skupinu, takže `\globaldefs`, `\endlinechar` a kategorie znaků mají zase stejný význam, jaký měly před načtením souboru.

Novinky v deklaračních souborech

Deklarační soubory vytvořené pro staré OFS z roku 2001 jsou a budou stále použitelné, tj. jazyk těchto souborů je zpětně kompatibilní. V nové verzi OFS jsem ale přidal další vlastnosti a je tedy možno použít nové příkazy.

Mezi již zmíněné novinky patří možnost přidělit každé rodině skupinu výjimek z kódování. To ale zdaleka není všechno, co lze nově využít ve verzi OFS 2004.

Každá rodina fontů může být registrována příkazem `\registerenc` k použití jen pro určitá kódování. Máme tak zaručeno, že například po

```
\def\fotenc{aaa} \setfonts [Times/]
```

se nebude OFS marně snažit najít neexistující metriku `ptmraaa.tfm`, ale vynadá nám, že kódování `aaa` není pro rodinu Times registrováno. Tato vlastnost je bohatě využita v OkTEXu, viz [5].

Každý deklarační i kódovací soubor může začínat příkazem

`\protectreading` ⟨*jméno souboru*⟩⟨*mezera*⟩`% This is part of OFS package`

Tím záměrně zkomplikujeme čtení tohoto souboru v balíčcích, kde není příkaz `\protectreading` definován. V nové verzi OFS je tento příkaz definován tak, že si poznamená ⟨*jméno souboru*⟩ do paměti a při příštím čtení stejného souboru se chová jako `\endinput`, tj. zaručí, že níže zapsané deklarace budou přečteny jen jednou. Podobnou vlastnost má LATEXový příkaz `\ProvidesPackage`.

Nově je zavedena též možnost předeklarovat již deklarované rodiny fontů prostým zopakováním příkazu `\ofsdeclarefamily [`⟨*Rodina*⟩`]`, ovšem s jinými parametry. To dává možnost uživateli přizpůsobit i implicitní rodiny fontů vlastním potřebám.

Dvojí čtení stejného souboru

Kam psát výjimky z kódování jednotlivých rodin fontů? Slušelo by se, aby byly tyto výjimky v deklaračních souborech společně s deklarovanými rodinami.

V rámci OkTEXu plánuji načíst pokud možno všechny dostupné deklarační soubory rodin fontů už při generování formátu. Pak by ale byly výjimky přečteny už při generování formátu. Ovšem to není dobré, pokud chceme šetřit paměť TEXu. Rozhodl jsem se tedy použít následující trik:

Výjimky z kódování lze zapisovat do deklaračních souborů za `\endinput`, takže při prvním čtení těchto souborů budou ignorovány. Ovšem v době, kdy bude rodina aktivována příkazem `\setfonts`, může OFS otevřít deklarační soubor ještě jednou, tentokrát přeskočí obvyklé deklarace i `\endinput` a přečte seznam výjimek z kódování. Pro tyto účely jsou nově vytvořeny příkazy `\modifyread` a `\modifytext`. Jak přesně pracují, je popsáno v dokumentaci [2] a příklad jejich použití najde čtenář v novém deklaračním souboru `slido.tex`.

Makro `\ofshexbox`

V plainTEXu je například znak § řešen pomocí makra `\mathhexbox`. Při vší úctě ke Knuthovi si dovolím poznamenat, že to je docela nekoncepční, protože textový znak je deklarován tak, že jeho funkčnost závisí na momentálním nastavení matematických fontů. Rozhodl jsem se tedy vytvořit pro takové účely nové makro `\ofshexbox`, které závisí jen na čtevřici deklarovaných textových

fontů, jež tvoří rodinu. OFS si navíc pohlídá velikost těchto fontů, tj. zaručí, že vždy budou použity ve správné velikosti. Pomocí příkazu

`\ofshexboxdef` ⟨*rodina*⟩`{`⟨*metrika-rm*⟩`}{`⟨*metrika-bf*⟩`}{`⟨*metrika-it*⟩`}{`⟨*metrika-bi*⟩`}`

lze deklarovat ⟨*rodinu*⟩ jako čtveřici metrik, která bude pak při použití makra `\ofshexbox`⟨*rodina*⟩⟨*hexa-kód*⟩ použita. Která konkrétně metrika bude použita, závisí na aktuální variantě fontů. Při `\bf` bude použita ⟨*metrika-bf*⟩, atd. Je-li aktuální jiná varianta než `\bf`, `\it`, `\bi`, bude použita ⟨*metrika-rm*⟩.

Následující ukázka z dokumentace ilustruje, jak se můžeme jednou pro vždy vyrovnat s tím, že některé rodiny znak Euro obsahují a některé ne:

```
\ofshexboxdef {TS1}{tcrm1000}{tcbx1000}{tcti1000}{tcbi1000}
\characterdef \euro * {\ofshexbox{TS1}BF}
```

Použité metriky `tc*1000.tfm` obsahují znak Euro na pozici `BF`. Tyto metriky z balíčku EC fontů jsou kódované podle tzv. Text Companion Encoding, v LaTeXu označované jako TS1. V ukázce jsme deklarovali znak `\euro` *pro všechny rodiny fontů, ve kterých znak není deklarován* pomocí makra `\ofshexbox`. Znak se tedy v rodinách, kde Euro není přítomno, vyloví z metrik `tc*1000.tfm`, a přitom bude záviset na aktuální variantě (tučná/kurzíva/tučná kurzíva) a navíc bude vždy použit v aktuální velikosti fontů. Pokud ale znak Euro je v rodině fontů obsažen, pak se přednostně použije tento znak.

Aby to celé fungovalo, je samozřejmě potřeba deklarovat pro každou rodinu odpovídající skupinu výjimek ze základního kódování. Pak bude mít uživatel záruku, že pokud napíše `\euro`, vždycky se mu vytiskne znak Euro. Přitom se nemusí starat, zda znak v aktuálně použitém fontu je nebo není.

Nové deklarace v OFS

Do nové verze OFS jsem přidal deklarace rodin `CM*` v kódování `8t` (podle Corku). Tyto rodiny odkazují na EC fonty a respektují všechny stupně zvětšení (různé metriky pro různé velikosti).

Dále jsem definoval nové kódování `8c`, které odpovídá tzv. „Text Companion encoding“ v LaTeXu známé jako TS1. Všem implicitním rodinám fontů `CMRoman`, `CMSans`, `CMTypewriter`, `Times`, `Helvetica`, `Courier` jsem přidělil rozšiřující metriku v kódování `8c`. Tím je například vyřešen znak `\euro` v rodinách `CM*`. Rodiny `Times`, `Helvetica` a `Courier` mají sice ve standardních distribucích TeXu připraveny metriky `8c`, ale tyto metriky zdaleka neobsahují všechny znaky podle definice tohoto kódování (například `\euro` chybí).Tyto rodiny mají tedy v OFS přiděleny výjimky z kódování `8c`.

Vzhledem k tomu, že metriky `tc*.tfm` patří rodině `CMRoman` v kódování `8c`, je možné příklad z předchozí sekce s příkazem `\ofshexbox` přepsat takto:

```
\ofshexboxdef {TS1}{cmr8c}{cmbx8c}{cmti8c}{cmbxti8c}
\characterdef \euro * {\ofshexbox{TS1}BF}
```

Čtenář, který zná dokumentaci k OFS, si jako cvičení může zkusit najít rozdíl mezi deklarací `\ofshexboxdef` z předchozí sekce a touto ukázkou. (Odpověď: při práci s různými velikostmi se podle předchozí sekce použije znak euro vždy z metriky `tc*1000.tfm` příslušně zvětšené pomocí `at`. Na druhé straně tato nová deklarace vybere podle požadované velikosti odpovídající metriku, třeba `tcrm1095.tfm`, `tcrm1200.tfm` atd. vždy navíc přesně zvětšenou pomocí `at`.)

Deklarační soubory pro matematické fonty

OFS z roku 2001 deklarovalo jen matematické kódování `PS` (PostSriptové znaky z fontu Symbol) a `CM` (Znaky z Computer Modern fontů). Později přibyl doplňkový soubor, kde byla deklarace pro fonty MathTimes.

V roce 2004 jsem nejprve vylepšil kódování `PS` tak, že znaky $\sum$, $\int$ a $\prod$ mají v display módu zvětšené varianty a dále jsem deklaroval další matematická kódování:

- Po načtení souboru `amsfn.tex` může uživatel použít `\def\fomenc{AMS}` a využít desítky speciálních matematických symbolů známých především z $\mathcal{AMS}$TEXu.
- Po načtení souboru `txfn.tex` může uživatel použít jednu ze dvou možností: `\def\fomenc{TX}` — pro matematiku se použijí TX fonty, tj. volně přístupná úplná sada matematických znaků vizuálně kompatibilních s rodinou Times a tedy i s mnoha dalšími běžnými písmy dynamické antikvy. Druhou možností je `\def\fomenc{PX}` — podobně jako při kódování `PS` se kombinuje kurzíva aktuální rodiny fontů s matematickými znaky z TX fontů.
- Po načtení souboru `mtfn.tex` může uživatel použít `\def\fomenc{MT}`. Pak se použijí metriky komerční sady matematických fontů MathTimes.

V tuto chvíli už mohu prozradit, že jsem na začátku tohoto článku zatajil, že obrázek 1 nekoresponduje zcela přesně s výpisem, který obrázku předchází. Test `\math` byl ve skutečnosti spuštěn až po zadání `\input txfn` a `\def\fomenc{PX}`. Na obrázku 1 tedy vidíme kombinaci volně dostupné rodiny Charter s volně dostupnými TX fonty pro matematickou sazbu. Tyto matematické fonty vypadají hodně dobře a navíc obsahují nadmnožinu všech znaků z Computer Modern i z $\mathcal{AMS}$ fontů dohromady. Doporučuji používat!

Nekompatibilita

Při přechodu na novou verzi jsem byl bohužel nucen v několika bodech ustoupit od exaktní zpětné kompatibility. Tyto body nyní přesně vyjmenuji. Věřím, že se jedná o změny, které většina uživatelů vůbec nepocítí.

Jazyk deklaračních souborů textových fontů byl pouze rozšířen, tj. staré konstrukce jsou stále použitelné, ale přibyly nová slovíčka. Deklarační soubory pro novou verzi OFS nejsou tedy zpracovatelné ve verzi staré. Typickým novým

slovíčkem je `\protectreading`, které se nyní snažím používat ve všech nových deklaračních souborech.

Jazyk deklaračních souborů matematických fontů byl bohužel mírně změněn. Může se tedy stát, že pokud si někdo vytvořil deklaraci matematické rodiny, nebude mu v novém OFS fungovat. Jste-li autory maker, která deklarují matematické rodiny fontů, pak doporučuji přečíst sekci 3.7 v dokumentaci [2], kde jsou změny důkladně popsány.

Další problémy s kompatibilitou mohou nastat na úrovni použití maker pro fonty Štormovy písmolijny. Tam jsem měl v deklaracích rodin docela nekoncepčně zavedeny i kódovací soubory pro kódování `6s`, zatímco při použití standardních rodin (např. ze souboru `a35.tex`) musel uživatel zavést kódovací soubory manuálně. Rozhodl jsem se kódovací soubory implicitně nezavádět nikde, takže jsem příslušný `\input` vymazal i z deklaračních souborů pro Štormovy fonty. Pokud uživatel chce načítat kódovací soubory, může v nové verzi OFS psát `\loadingenc=1`.

V původní deklaraci Štormových fontů bylo navíc aktivováno makro, které rozšiřovalo sadu matematických fontů o aktuální Štormův font, na který pak byly odkazy u těch matematických znaků, které Štorm do svých písem obvykle zahrnul (například ∞). To ale nefungovalo spolehlivě (ne ve všech Štormových fontech byly matematické znaky zastoupeny stejnou měrou), takže jsem tuto vlastnost zrušil úplně. Pro uživatele, kteří to někdy využili, jsem ponechal makro `\stmath`, jehož explicitním spuštěním se matematická sada fontů rozšíří stejně, jako ve starých deklaracích Štormových fontů.

Citelnou změnu v deklaracích mohou pocítit uživatelé, kteří používali makra definovaná na koncích kódovacích souborů a týkající se uvozovek: `\singleuv`, `\doubleuv`, `\doublefuv`, atd. Tato makra jsem z deklarací vyhodil, protože podle mého názoru patří do stylového souboru pro daný jazyk. Zařadil jsem je proto do OkTEXu. Pokud je uživatel potřebuje, může si je ze souborů `ofs-8?.tex` a `ofs-6s.tex` obkreslit. Zůstala tam, jsou ale zakomentovaná.

Šetření pamětí v příkazu `\showfonts` mě vedlo k odstranění interního příkazu `\listfamilies`. Pokud si uživatel udělal makra, která s tímto příkazem pracovala, pak toto přestane fungovat. Například mé makro `ofscatal.tex` přestalo z tohoto důvodu pracovat. Rozhodl jsem se toto makro už nepředělávat a odsunout je na smetiště dějin. Makro `ofstest.tex` totiž umí taky dělat katalogy fontů, viz například obrázek 2.

Obnovení dokumentace včetně anglické

Českou dokumentci jsem upravoval průběžně před provedením změn ve vlastních makrech OFS. Anglickou dokumentaci jsem pak upravil se zpožděním. Vydatně mi při tom pomohl student Tomáš Komárek. Děkuji.

Odkazy

1. `ftp://math.feld.cvut.cz/pub/olsak/ofs`.
2. Petr Olšák. *OFS: Olšákův fontový systém.* 2001, 2004. Dokumentace k balíku je v souborech `ofsdoc.tex`, `ofsdoc.pdf`.
3. Petr Olšák. *Makro OFS.* in: SLT 2002 – sborník semináře o Linuxu a TEXu, str. 79–92.
4. `ftp://math.feld.cvut.cz/pub/olsak/oktex`.
5. Petr Olšák. *Projekt OkTEX.* in: SLT 2004 – sborník semináře o Linuxu a TEXu.

Projekt OkTEX

Petr Olšák

OkTEX je formát TEXu postavený na plainTEXu a balíčcích OFS, LANG a IENC. Je to pokus vytvořit multijazykové prostředí pro plainTEX, které svými vlastnostmi předčí balíček Babel známý především z LATEXu. Pro potřeby projektu byl vytvořen nový balíček LANG, který spolupracuje s OFS pro plain a dovoluje přepínat mezi jazyky včetně možnosti deklarovat pro každý jazyk libovolné množství kódování fontů. Ve fázi vývoje je dále balíček IENC, který umožňuje nastavit konverze ze vstupního kódování na kódování fontů a spolupracuje přitom s balíčkem LANG a případně s encTEXem, pokud je toto rozšíření v TEXové distribuci dosažitelné.

Úvodem

Na rozdíl od ostatních programů a maker, které jsem dosud na Internetu zveřejnil a na TEXových setkáních prezentoval, je OkTEX zatím neukončen. To znamená, že makra jsou neodladěná a podléhají vývoji. Rozhodl jsem se k tomuto kroku proto, že projekt si asi vyžádá více práce a předpokládal jsem, že se na něm zapojí studenti mého předmětu „Publikační systém TEX“. Bohužel, mé předpoklady se nenaplnily, asi si na spolupráci studenti netroufli.

Na [1] tedy můžete vidět rozpracovaný projekt a můžete pozorovat, v jakém pořadí se postupně cíle projektu naplňují. Osobně jsem zastáncem principu: „nejprve dokumentace a potom programování“, ačkoli mnoho jiných projektů to dělá právě v opačném pořadí. Postup od dokumentce k programu resp. implementaci maker mě přijde daleko přirozenější, protože koncept si autor může rozmyslet v době psaní dokumentace. Jakmile je dokumentace přesně sformulována, pak vlastní programování už může zvládnout cvičená opice. Vymýšlení

konceptu je intelektuálně daleko zábavnější činnost než programování podle toho, co je v dokumentaci napsáno. A psát dokumentaci až podle toho, co někdo naprogramoval, mi připadá mírně řečeno nekoncepční.

Počítám, že první použitelná verze maker bude k dispozici na Internetu na konci léta 2004.

Hlavním pilířem OkTEXu je balíček LANG, který přepíná mezi jazyky a opírá se o OFS pro plainTEX. Rozhodl jsem se naprogramovat tuto aplikaci proto, abych předvedl možnosti přepínání jazyků, které je úzce provázáno s přepínáním fontů včetně možnosti volby jejich kódování. To se může stát inspirací pro LATEXový balíček Babel. Sám osobně s LATEXem nekoketuji, a proto neplánuji ani vylepšování Babelu ani portování balíčku LANG pro LATEX. Zkušenosti s OkTEXem by ale mohly být podnětem pro podobný přístup v LATEXu, který tam zatím chybí. Například CSLATEX přežívá v TEXových distribucích jen kvůli neexistující možnosti pracovat v Babelu s větším množstvím kódování fontů pro jeden jazyk. CSLATEX totiž použití více kódování pro češtinu a slovenštinu umožňuje, ale přináší také problémy novopečeným uživatelům, kteří nevědí, že se stylový soubor pro CSLATEX nedá použít s Babelem.

Formát

Protože přepínání jazyků si vynucuje přepínání vzorů dělení slov a tyto vzory se v TEXu dají načíst jen při generování formátu iniTEXem, je potřeba, aby se balíček LANG stal součástí formátu. Není tedy možné jej načítat pomocí `\input` až při zpracování dokumentu. To je důvod, proč jsem zavedl nový formát. Ten jsem nazval OkTEX, protože mě nic lepšího nenapadlo. Následuje ukázka souboru `oktex.ini`, který je použit pro generování formátu:

```
% oktex.ini
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Feb. 2004                          Petr Olsak

%  initex oktex.ini   nebo:  initex -enc oktex.ini

\input csfonts        % předefinuje primitiv \font jako v csplainu
\let\oriinput=\input
\def\input#1 {}       % předefinuje \input, aby nedělal nic
\oriinput plain       % načte plain.tex s pozměněným \font a \input
\restorefont \let\input=\oriinput
                      % vrátí \font a \input do původního stavu
\input ofs [allfonts] % načte OFS včetně všech deklaračních souborů
\input lang           % načte LANG a uloží vzory dělení slov
\everyjob={\message{This is oktex (plain+OFS+LANG), version May 2004}}
\showlangs            % vypíše seznam deklarovaných jazyků
\dump                 % uloží formát oktex.fmt
```

Vidíme, že formát kromě balíčku LANG zahrnuje i balíček OFS. Protože jsou při načítání OFS zavedeny názvy veškerých fontů, co máme v počítači `[allfonts]`, máme tak možnost v OkTEXu rovnou pracovat s fonty prostřednictvím rozhraní OFS a nemusíme kvůli tomu načítat ani OFS samotné, ani deklarační soubory použitých fontů.

Před načtením makra `plain.tex` je pozměněn význam primitivů `\font` a `\input`. Výchozí fonty jsou tedy načteny stejně jako ve formátu `csplain`: fonty `\preloaded` nejsou načteny vůbec a v případě textových fontů jsou místo CM zavedeny CS fonty. Hlavním důvodem tohoto opatření je nezavádět do paměti fonty `\preloaded`, kterých je v makru `plain.tex` docela hodně. S fonty budeme pracovat pomocí OFS a budeme je zavádět až v okamžiku potřeby. Potřebujeme tedy šetřit s pamětí pro fonty. Obrovská sada fontů `\preloaded` v makru `plain.tex` má podle mého názoru jen historické důvody, které dávno pominuly. V pradávných počátcích TEXu asi trvalo nějakou dobu, než se font z metriky `tfm` zavedl do paměti. Bylo tedy účelné tuto činnost provést jen jednou při generování formátu, a nikoli opakovaně při práci s dokumentem. Dnes je toto opatření zcela zbytečné, ba naopak nám překáží, protože `\preloaded` fonty zbytečně zaplňují paměť vyhrazenou pro fonty smetím, které pravděpodobně nikdy nebudeme potřebovat.

Pozměnění `\input` před načtením `plain.tex` způsobí, že nebudou makrem `plain.tex` načteny vzory dělení pro anglický jazyk. V tomto makru je totiž jediný výskyt primitivu `\input` v kontextu: `\input hyphen`⟨*mezera*⟩. O načtení vzorů dělení se postará balíček LANG, jak uvidíme dále. Protože je obvykle angličtina deklarována jako první jazyk, dostane přiděleno `\language=0` a výsledek je tedy kompatibilní s makrem `plain.tex`.

Balíčky OFS ani LANG nemění význam žádných maker plainu ani primitivů TEXu, dokud nejsou použity příkazy `\setfonts` nebo `\setlang`. Znamená to, že formát OkTEX se chová identicky, jako formát plain, pokud uživatel nepoužije výše zmíněné příkazy. Dokumenty, určené pro plainTEX, jsou tedy OkTEXem zprocovány se stejným výsledkem.

Balíček LANG definuje (pouze pro potřebu kompatibility s csplainem) příkaz `\chyph`, který nastaví české vzory dělení a zavede stylový soubor pro češtinu, kde je například definováno makro `\uv`. Dokument, který má v úvodní části použit příkaz `\chyph` se tedy v OkTEXu chová (skoro) stejně jako v csplainu. Slovo „skoro“ v sobě zahrnuje dvě odlišnosti:

- OkTEX zůstává implicitně u hodnot `\hsize`, `\vsize` podle plainTEXu, zatímco csplain tyto hodnoty mění, aby lépe odpovídaly formátu A4.
- Makro `\uv` ze stylového souboru OkTEXu umožní tvorbu kerningových párů se znaky uvozovek, ale neumožní použít verbatim konstrukci uvnitř uvozovek. Chceme-li kompatibilitu s csplainem, měli bychom po příkazu `\chyph` napsat ještě `\let\uv=\verbuv`.

To samé, co zde bylo řečeno o příkazu `\chyph`, platí pro příkaz `\shyph` a slovenský jazyk.

Deklarace jazyků v balíčku LANG

Balíček LANG je podrobně dokumentován v souboru `langdoc.tex`. Mimo jiné se tam dozvíme, jak může vypadat struktura deklaračního souboru pro jazyky. Tento soubor má název `langdef.tex` a zde je jeho ukázka:

```
% langdef.tex
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
% Mar. 2004                                        Petr Olšák

\declareenc  8t  CMRoman  % T1, latin from Cork (EC fonts, DC fonts)
\declareenc  8z  CMRoman  % IL2, similar to ISO-8859-2 (CS fonts)
\declareenc  6a  CMRoman  % T2A, Cyrillic (LH fonts)
\declareenc  6t  CMRoman  % T2B, Cyrillic, another spec. symbols (LH fonts)
\declareenc  6c  CMRoman  % T2C, Cyrillic, another spec. symbols (LH fonts)
\declareenc  6x  CMRoman  % X2, Cyrillic, another spec. symbols (LH fonts)
\declareenc  6k  CMRoman  % KOI-8, Cyrillic
\declareenc  6y  CMRoman  % LCY, Cyrillic (LH fonts)

%              selector      code  style          encodings
 \declarelang  \english      en    lang-en.tex    8t =8z ;
 \declarelang  \czech        cz    lang-cz.tex    8z  8t ;
 \declarelang  \slovak       sk    lang-sk.tex    8z  8t ;
 \declarelang  \german       de    lang-de.tex    8t =8z ;
 \declarelang  \newgerman    de2   lang-de.tex    8t =8z ;
 \declarelang  \polish       pl    lang-pl.tex    8t  8z ;
 \declarelang  \russian      ru    lang-ru.tex    6a =6t =6c =6x 6k 6y ;
```

Vidíme, že nejprve jsou deklarována kódování, která budou pro jednotlivé jazyky použita. Součástí deklarace je informace o rodině fontů, o které víme, že určitě je v daném kódování připravena. Když bude balíček LANG potřebovat přepnout do takového kódování, pak v případě, že aktuální rodina fontů toto kódování nepodporuje, bude místo ní použita rodina fontů z této deklarace. Je to tedy implicitní rodina fontů pro dané kódování. Jména kódování fontů odpovídají dokumentu [2] a jména rodin odpovídají deklaraci v OFS.

Následující část souboru `langdef.tex` obsahuje vlastní deklarace jazyků. Je zde uveden přepínač (`\czech`, `\german`, atd.), pomocí kterého může uživatel balíčku přepnout do zvoleného jazyka. Následuje zkratka jazyka (`cz`, `de`), která se používá v parametru příkazu `\setlang` a na mnoha dalších místech v balíčku LANG. Dále následuje jméno stylového souboru, který balíček LANG načte při prvním přepnutí do daného jazyka. Tam jsou na jazyku závislé definice. Konečně je v deklaraci uveden seznam kódování, která jsou pro daný jazyk použitelná. Jinými slovy, abeceda jazyka je v těchto kódováních plně obsažena. První kódování v seznamu je tzv. „výchozí kódování“. To bude použito např.

při prvním přepnutí jazyka, pokud není kódování specifikováno. Další kódování v deklaraci mohou být uvozena rovnítkem. To znamená, že abeceda jazyka je tam stejně rozložena jako v předchozím kódování a nemá tedy smysl načítat nové vzory dělení slov. Pokud ale rovnítko chybí, LANG načte znovu vzory dělení slov podle požadovaného kódování. Jazyk může tedy mít stejné vzory dělení slov načteny vícekrát pod různými kódováními. To pro nás není nic překvapivého — známe to z csplainu i z CSLaTeXu

Příkaz `\declarelang` sám vzory dělení nenačítá. Předpokládám totiž, že zde budou deklarovány všechny jazyky, se kterými budeme chtít v balíčku LANG pracovat a že těchto jazyků bude třeba tolik, že načtení všech jejich zorů dělení buď nebude vůbec možné nebo by zbytečně zvětšovalo formát. Můžeme tedy načíst vzory dělení jen některých deklarovaných jazyků, ba co dím, jen pro některá kódování daného jazyka. K tomu slouží příkaz `\loadpatterns`, jehož použití může vypadat takto:

```
\loadpatterns en cz sk ru {6a =6t =6c =6x} de de2 ;
```

V tomto příkladě jsme se rozhodli vůbec nenačítat polské vzory dělení a dále omezit načítání ruských vzorů dělení na jedinou tabulku (`6a =6t =6c =6x`), vzory pro kódování `6k` a `6y` nebudou pro ruštinu načteny. Balíček LANG přidělí čísla vzorů dělení počínaje nulou vzestupně. Protože jazyk `en` je uveden jako první (v parametru příkazu `\loadpatterns`), bude mít přiděleno číslo nula.

Balíček LANG umožňuje příkazem `\showlangs` zobrazit seznam deklarovaných jazyků a přidělených čísel vzorů dělení. Pro náš příklad dopadne výstup `\showlang` takto:

```
LANG selector      style-file    encodings<num.of.hyphen.table>
-----------------------------------------------------------------
en   \english      lang-en.tex    8t<0> 8z<0>
cz   \czech        lang-cz.tex    8z<1> 8t<2>
sk   \slovak       lang-sk.tex    8z<3> 8t<4>
de   \german       lang-de.tex    8t<6> 8z<6>
de2  \newgerman    lang-de.tex    8t<7> 8z<7>
pl   \polish       lang-pl.tex    8t<?> 8z<?>
ru   \russian      lang-ru.tex    6a<5> 6t<5> 6c<5> 6x<5> 6k<?> 6y<?>
```

Otazník znamená, že vzory dělení nejsou načteny. Vidíme, že němčina má pro obě přípustná kódování načteny stejné vzory dělení, zatímco čeština a slovenština má pro různá kódování různé vzory dělení. Němčina je svým způsobem specifická, protože má dvě různé varianty vzorů dělení: podle starých pravidel (`\german`) a podle nových pravidel (`\newgerman`). Tento případ je v balíčku LANG deklarován jakoby dva různé jazyky.

Pokud bude muset balíček LANG přepnout do jazyka a kódování, pro které nejsou načteny vzory dělení, vypíše se varování a použijí se vzory dělení podle

tabulky `\defaulthyphentable`. Tento registr je implicitně nastaven na hodnotu 255, což představuje prázdnou tabulku vzorů, při které se žádné dělení slov konat nebude.

Deklarace jazyků a načtení vzorů dělení má ještě další možnosti, které si zájemce může přečíst v dokumentaci.

Pravidla přepínání jazyků, kódování a fontů

Představme si šestirozměrný prostor L, v němž na jednotlivé osy nanášíme tyto údaje:

- jazyk
- nářečí
- kódování fontů
- rodina fontů
- varianta fontu
- velikost fontu

Cílem projektu OkTEX bylo připravit makra, pomocí kterých bude možné se v tomto prostoru pohybovat třeba podél jen některých os. Problém je samozřejmě v tom, že uvedený prostor je poměrně dost děravý: Ne v každém kódování máme k dispozici všechny rodiny fontů, ne všechny jazyky jsou použitelné ve všech kódováních, ne každá varianta fontu se vyskytuje v každé rodině fontů atd. Úplná záruka neměnnosti ostatních souřadnic při změně jedné souřadnice prostoru L tedy není možná. Obecně lze říci, že pokud si uživatel přeje změnit souřadnice takovým způsobem, že se „strefí do díry“ v tomto prostoru, pak LANG a OFS se snaží najít nejbližší vhodný neprázdný uzel prostoru a o této skutečnosti vypíše na terminál a do logu varování. Díky deklaracím jazyků a registracím kódování ke zvoleným rodinám fontů mají makra OFS a LANG přehled o „dírách“ i plných uzlech tohoto prostoru a mohou se podle toho zařídit.

Například OFS udržuje pokud možno stále stejnou variantu fontu, ačkoli přepíná mezi rodinami. Pokud nová rodina požadovanou variantu neobsahuje, použije OFS variantu `\rm`, která musí být deklarována v každé rodině fontů.

Jazyky můžeme měnit pomocí přepínačů (`\czech`, `\german`, atd.) nebo použitím příkazu `\setlang[`⟨*zkratka-jazyka*⟩`/`⟨*kódování*⟩`]`. Jestliže je některý z parametrů tohoto příkazu prázdný, pokusí se LANG zachovat tento parametr beze změny. Při přepnutí do jazyka a kódování se samozřejmě inicializují odpovídající vzory dělení slov, pokud byly příkazem `\loadpatterns` načteny.

Je-li použito takové `\setlang[`⟨*zkratka-jazyka*⟩`/`⟨*kódování*⟩`]`, které vychází do „díry“ prostoru L, snaží se LANG přepnout na požadovaný jazyk a přizpůsobit kódování. V takovém případě použije kódování, které bylo v dokumentu

s daným jazykem použito naposled. Při prvním přepnutí do daného jazyka použije v tomto případě výchozí kódování daného jazyka.

Prázdný parametr ⟨*kódování*⟩ v příkazu `\setlang` znamená, že aktuální kódování nebude měněno, pokud se tím ovšem nedostáváme do „díry“ v prostoru L. Prázdný parametr ⟨*kódování*⟩ při prvním použití příkazu `\setlang` v dokumentu má ale poněkud odlišný význam: přepne do výchozího kódování daného jazyka.

Přepínače `\czech`, `\german`, `\english` atd. jsou v balíčku LANG definovány jako `\setlang[`⟨*zkratka-jazyka*⟩`/]`, tedy s prázdným parametrem ⟨*kódování*⟩. Znamená to, že první použití takového přepínače (není-li před ním použito žádné `\setlang`) určí i kódování fontů. LANG zvolí za kódování fontů v tomto případě výchozí kódování tohoto jazyka. Všechna další použití těchto přepínačů udrží nastavené kódování, pokud se tím nedostáváme do „díry“ prostoru L.

Příklady:

```
\czech  % od této chvíle máme zapnuto kódování 8z, tj. podle CSfontů
        % OkTeX se nyní chová stejně jako csplain
```

Jiný dokument:

```
\setlang[/8t]
\czech % fonty jsou nyní kódované podle Corku
```

Ještě jiný dokument:

```
\setlang[cz/8t] % fonty jsou kódovány podle Corku
```

Implicitní jazyk balíčku LANG (má smysl se na něj ptát, pokud se při prvním použití příkazu `\setlang` použije prázdný parametr ⟨*jazyk*⟩) má hodnotu `none`. Tento speciální jazyk nepřepíná žádné vzory dělení, nenačítá žádný stylový soubor a akceptuje jakékoli deklarované kódování fontů. Podrobněji o této možnosti je pojednáno v dokumentaci.

Protože příkaz `\setlang` přepíná kódování fontů, úzce při tom spolupracuje s makrem OFS. Při přechodu na jiné kódování se musí najít vhodná rodina fontů, která dané kódování podporuje. Pokud aktuální rodina tuto vlastnost splňuje, ponechá se beze změny, pouze se nově načtou metriky této rodiny podle požadovaného kódování. Pokud ale aktuální rodina fontů nemá požadované kódování registrováno, spustí makro `\setlang` na závěr své činnosti příkaz `\setfonts[`⟨*Rodina*⟩`/]`, kde ⟨*Rodina*⟩ je implicitní rodina zvoleného kódování. Ta je deklarována v souboru `langdef.tex`. Samozřejmě tento příkaz udrží fonty ve stejné velikosti a pokusí se udržet variantu fontů, pokud ji nová rodina obsahuje. Jinak přechází na `\rm`.

Při jakémkoli pokusu dostat se do „díry“ prostoru L se vypíše varování o změně souřadnic aktuálního bodu prostoru, protože jsou nastaveny jinak, než si uživatel přeje. Uživatel může změnit všech pět zatím zmíněných souřadnic najednou pomocí dvojice příkazů:

`\setlang [`⟨*jazyk*⟩`/`⟨*kódování*⟩`]\setfonts[`⟨*Rodina*⟩`-`⟨*varianta*⟩`/`⟨*velikost*⟩`]`

Libovolný z těchto údajů může chybět. V takovém případě se LANG a OFS pokusí ponechat odpovídající souřadnici beze změny. Pokud se takto těsně za sebou sejdou příkazy `\setlang` a `\setfonts`, pak se makro `\setlang` nesnaží přepínat rodiny ve vlastní režii, ale přenechá tuto práci následujícímu příkazu `\setfonts`. Uživatel tak může zamezit výpisu varování o tom, že dosud používaná rodina fontů není v novém kódování akceptovatelná. Balíček LANG se v tomto případě nesnaží přepnout do implicitní rodiny zvoleného kódování. Udělá to jen tehdy, pokud příkaz `\setfonts` skončil neúspěšně.

Dosud jsem nezmínil poslední souřadnici prostoru L: „nářečí“. Z matematického pohledu se nejedná o plnohodnotnou dimenzi tohoto prostoru, protože každý jazyk může mít (ve stylovém souboru pro jazyk) deklarovánu svou množinu nářečí, která je zcela nezávislá na jiných jazycích. Uživatel může v rámci aktuálního jazyka přepnout do nářečí pomocí `\setdialect[`⟨*nářečí*⟩`]`. To může ovlivnit chování některých na jazyku závislých maker, která jsou definována ve stylových souborech. Typickým příkladem takového makra je `\today`.

Pokud `\setdialect` skončil úspěšně, LANG přepne nářečí. Jinak LANG zůstává u implicitního nářečí daného jazyka. LANG si také pamatuje naposledy použité nářečí každého jazyka a pokud se pomocí `\setlang` uživatel k tomuto jazyku vrátí, automaticky se obnoví naposledy použité nářečí tohoto jazyka.

Používání stylových souborů jazyka

Abych odlišil stylové soubory `*.sty` z LaTeXu od stylových souborů pro balíček LANG, rozhodl jsem se jim dávat příponu `.tex`. Doporučený název stylového souboru pro LANG je `lang-`⟨*zkratka-jazyka*⟩`.tex`.

Při prvním přepnutí do daného jazyka se stylový soubor načte automaticky. Uživatel tedy nemusí nic psát do záhlaví svého dokumentu. Ve stylových souborech jazyka se definují makra specifická pro daný jazyk. Navíc je za `\endinput` ve stylovém souboru rovnou napsána dokumentace, která se dá formátovat plainTeXem pomocí příkazu `\printlangdoc`. Například

```
\czech \printlangdoc \german \printlangdoc
```

vytiskne do `dvi` souboru dokumentaci ke stylovému souboru českého a německého jazyka.

Typická úloha stylových souborů pro LANG je vytvořit makra, která mají společný název, ale pro každý jazyk jsou definována poněkud odlišně. To zařídí příkaz `\langdef`, který definuje makro vázané na zvolený jazyk. Pokud příště definujeme pomocí `\langdef` makro stejného jména, ale pro jiný jazyk, pak se definice nepřekrývají. Jméno makra pak expanduje závisle na aktuálně zvoleném jazyce. Například `\today` je jinak definováno pro angličtinu a jinak pro

čestinu. V obou stylových souborech je uvedeno `\langdef\today`, přičemž definice tohoto makra pro češtinu se jistě liší od definice pro angličtinu. Pokud uživatel napíše `\today` a je zrovna aktivní čeština, uplatní se česká definice a při přepnutí do angličtiny napíše `\today` dnešní datum anglicky.

Uveďme si příklad stylového souboru `lang-cz.tex`:

```
%% Czech lang-style file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%% Mar. 2004                              Petr Olsak

\def\langdeflist {cz}%   %% Czech language

\langdef \langinfo {Czech language}

\langdef \langphrases {%
   \def\abstractname{Abstrakt}%
   \def\appendixname{P\v r\'iloha}%
   \def\bibname{Literatura}%
   \def\ccname{Na v\v edom\'i}%
   ⟨...atd.⟩
   \def\tablename{Tabulka}%
}
\langdef \today {\number\day. \ifcase\month\or ledna\or \'unora\or
   b\v rezna\or dubna\or kv\v etna\or \v cervna\or \v cervence\or
   srpna\or z\'a\v r\'i\or \v r\'ijna\or listopadu\or
   prosince\fi \space\number\year
}
%%% Common parts for Czech+Slovak languages:
\inputonce l-czsk.tex %space
\endinput
```

A v souboru `l-czsk.tex` pokračuje společná část pro češtinu a slovenštinu:

```
%% Czech + Slovak lang-style file
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% Mar. 2004                              Petr Olsak

\def\langdeflist {cz sk} %% Common part for Czech and Slovak

\langdef \langinit {%
   \langmessage{frenchspacing on}\frenchspacing
   \afterlang{\langmessage{frenchspacing off}\nonfrenchspacing}%
}
\langdef \activechar #1{%
   \if\string#1"\langmessage
      {Character " is activated as \string\clqq...\string\crqq.}%
      \activedef "{\bgroup\maybeactive\def"{\crqq\egroup}\clqq}\else
   \if\string#1-\langmessage
      {Character - is activated as hyphen-split character.}%
      \def\minus{-}%
```

```
      \activedef -{\ssdoublechar-}\else
   \if\string#1‘\langmessage
      ⟨..atd.⟩
}
%%% double chars:
\doublechardef -\mathchar{-}
\doublechardef --{\futurelet\tmpa\cz@tryemdash}
\doublechardef -#1{\discretionary{-}{-}{-}#1}
\doublechardef ‘\mathchar{‘}
⟨...atd.⟩

%%% quotes:
\let\langdefprefix=\long \langdef \doubleuv #1{\clqq#1\crqq}
\let\langdefprefix=\long \langdef \singleuv #1{\clq#1\crq}
\let\langdefprefix=\long \langdef \doublefuv #1{\frqq#1\flqq}

\langdef \verbuv {%
   \bgroup\aftergroup\crqq
   \afterassignment\clqq\let\next=}

\ifx\uv\undefined    \let\uv=\doubleuv    \fi
\ifx\fuv\undefined   \let\fuv=\doublefuv  \fi

\let\langdefprefix=\long \langdef \uvinuv #1{%
   \bgroup \let\uv=\singleuv \clqq#1\crqq\egroup}
⟨...atd.⟩
```

Nebudu zde podrobně popisovat vlastnosti použitých příkazů ani koncept stylového souboru, od toho je dokumentace. Zmíním základní strategii jen stručně. Makro `\langdeflist` musí obsahovat seznam zkratek jazyků, pro které příkaz `\langdef` bude definovat makra. Každý stylový soubor by měl definovat pomocí `\langdef` makra `\languinfo` (jméno jazyka), `\languinit` (inicializační makro, které se provede při každém přepnutí do tohoto jazyka), `\langphrases` (definice běžných frází používaných většinou v LaTeXu), `\today` (aktuální datum v daném jazyce), `\activechar` (spouštědlo aktivních znaků s definicemi většinou specifickými pro daný jazyk). Dále je možno ve stylových souborech používat `\doublechardef`, které definuje dvojice znaků, pokud uživatel první znak těchto dvojic zaktivní pomocí `\activechar`. Implicitně stylové soubory nenastavují žádný znak samy o sobě jako aktivní. Volba je ponechána na uživateli (ten musí mít úplnou kontrolu o aktivních znacích) a může znak zaktivnit pomocí `\activechar`⟨*znak*⟩. Například „`\czech \activechar-`“ je analogické LaTeXovému „`\usepackage[split]{czech}`“.

V Babelovských stylových souborech jsou často definovány tzv. „shorthands“, což jsou dvojice znaků se specifickým chováním v daném jazyce. I tuto věc lze deklarovat pomocí `\doublechardef`. Tento deklarátor navíc podobně jako `\langdef` definuje makra závislá na zvoleném jazyce.

Na rozdíl od LaTeXových stylových souborů, které musí uživatel uvést v záhlaví dokumentu a jsou tedy zaručeně čteny ve vertikálním módu na vnější úrovni skupin, tento předpoklad nemusí být splněn při čtení stylových souborů balíčku LANG. Stylový soubor je zde načten při prvním přepnutí do daného jazyka a to se může stát klidně v horizontálním módu nebo dokonce uvnitř skupiny, kdy má uživatel nastaveny kategorie znaků třeba zcela nestandardním způsobem. LANG proto čtení stylového souboru provádí stejně jako OFS načítá kódovací soubory (viz [5]).

Čtenář si může všimnout, že stylové soubory balíčku LANG umějí to samé jako stávající stylové soubory pro LaTeX a Babel, ale jsou delako přehlednější a srozumitelnější. Neřeší se v nich například otázka jednotlivých znaků (například jak vytvořit `\clqq`, `\crqq`), protože tyto záležitosti do stylových souborů pro jazyk nepatří. Tyto otázky mají být řešeny na úrovni maker, která manipulují s kódováním fontů (v našem případě na úrovni OFS). V deklaraci jazyka v souboru `langdef.tex` uvádíme seznam povolených kódování, která se s daným jazykem mohou použít. Průnik znaků ze všech těchto kódování je znaková výbava, se kterou můžeme ve stylových souborech pro jazyk počítat a nemusíme řešit, jak jsou tyto znaky v jednotlivých fontech realizovány. Toto oddělení problematiky kódování od problematiky jazyků ve stavajících stylových souborech LaTeXu chybí a je tam bohužel nesmírný guláš. Balíček LANG nám tedy umožňuje přepsat kompletně stylové soubory pro jazyky znova, a mnohem lépe.

V současné době jsem pro LANG přepsal stylové soubory angličtiny, němčiny, češtiny, slovenštiny a polštiny, částečně ruštiny. Implementace dalších jazyků, zvláště těch více exotických, vyžaduje nejprve rozbor použitelných kódování, implementace těchto kódování do OFS, převzetí vzorů dělení a nakonec napsání stylového souboru pro LANG. OFS je pro implementaci dalších kódování pro nelatinkové abecedy připraveno. Nabízí podle mého názoru vše, co je možné vyždímat z 8bitového TeXu. Samořejmě jazyky s tisíci znaky v abecedě nebudou asi tímto přístupem implementovatelné, ale tady skutečně narážíme na limity TeXu, kde vzory dělení jednoho jazyka musejí pracovat s abecedou s maximálně 240 znaky.

Vzory dělení slov

Vzory dělení jsou načítány při generování formátu příkazem `\loadpatterns` pro všechny jazyky, které jsou uvedeny v parametrech tohoto příkazu. Podporuje-li jazyk více kódování, která různě rozmisťují abecedu jazyka, pak příkaz `\loadpatterns` může načíst stejné vzory dělení slov vícekrát pod různými kódováními.

Vzory dělení slov pro daný jazyk jsou v balíčku LANG uloženy výhradně v souboru `hyph-`⟨*zkratka-jazyka*⟩`.tex`.

Původně jsem si myslel, že konverzi na cílové kódování provedu v době načítání vzorů dělení na úrovni expand procesoru, jako je to uděláno v CSLaTeXu. Ve vzorech dělení bychom zapisovali znaky jazyka pomocí přepisů `\v c`, `\'a` atd., což by bylo nezávislé na kódování. Pro nastavení těchto maker by se dalo využít OFS, které ve svých deklaračních souborech pro jednotlivá kódování tato makra definuje. Tento návrh jsem nakonec nepoužil, protože při načítání vzorů jazyků celého světa bychom zbytečně zatížili paměť TEXu definicemi maker pro velké množství znaků. Pravda, po uzavření skupiny se paměť uvolní, ale ne celá. Všechny „multileter control sequences“ zůstávají v paměti TEXu natrvalo. To je při velkém množství znaků všech možných jazyků zbytečné plýtvání.

Rozhodl jsem se tedy konverze vzorů dělení dělat pomocí `\lccode`. V TEXu totiž platí, že parametry primitivů `\patterns` a `\hyphenation` jsou nejprve konvertovány přes `\lccode` a teprve potom použity a uloženy ve vhodném tvaru do paměti. Vzor dělení daného jazyka může být tedy osmibitový nebo zapsaný v zobákové konvenci (např. `^^ab`). Přitom musíme dát makrům najevo, v jakém kódování vzory dělení jsou. Proto na začátku souboru se vzory dělení musí být uveden příkaz `\declarehyphentable`. Úvodní část souboru `hyph-cz.tex` tedy vypadá takto:

```
%%% Czech hyphen table by Pavel Sevecek, see czhyphen.tex for more details
\declarehyphentable [cz/8t] % for LANG package
\patterns{
.a2
.a4da
.a4de
.a4di
.a4do
.a4d^^e9
.a4kl
.a4ko
.a4kr
⟨...atd⟩
```

Pokud má `\loadpatterns` načíst české vzory v kódování `8t` (tj. podle Corku), pak příkaz `\declarehyphentable[cz/8t]` ponechá všechny hodnoty `\lccode` rovny svým parametrům, tj. provede se konverze jedna ku jedné. Jestliže se ale mají načíst tytéž vzory dělení v jiném kódování než `8t`, pak se `\declarehyphentable[cz/8t]` promění v `\input h8t-`⟨*kódování*⟩`.tex`. Například při čtení našich vzorů v kódování `8z` se přečte soubor `h8t-8z.tex`. Tam je jednoduše řečeno toto:

```
%% The \lccode transformation table from 8t to 8z encoding
%
% For LANG package prepared by Petr Olsak
% Mar. 2004

\lccode‘^^a3=‘^^e8
\lccode‘^^a4=‘^^ef
\lccode‘^^a5=‘^^ec
\lccode‘^^a8=‘^^e5
```
⟨*...atd.*⟩

Tím je konverze kódování vzorů dělení ze vstupního `8t` do cílového `8z` zaručena. Vzory dělení pro každý jazyk a pro každé kódování jsou načítány ve skupině, takže po ukončení skupiny se všechna pomocná makra a přechodná nastavení `\lccode` vrátí do původního stavu. Přitom `\patterns` a `\hyphenation` provádí přiřazení globálně, takže to potřebné se nezapomene.

Zatímco vzory českého a slovenského jazyka jsem kompletně převedl do kódování `8t` (z původního na kódování nezávislého TEXového značení), u mnoha jiných jazyků to není potřeba dělat. Například polština má vzory dělení zapsány pomocí aktivního znaku `/` a uloženy v souboru `plhyph.tex`. Potom stačí do souboru `hyph-pl.tex` pro LANG napsat:

```
\declarehyphentable [pl/8t] % for LANG package
 \catcode‘\/=13
 \def/#1{%
  \ifx#1a^^a1\else\ifx#1c^^a2\else\ifx#1e^^a6\else\ifx#1l^^aa\else
  \ifx#1n^^ab\else\ifx#1o^^f3\else\ifx#1s^^b1\else\ifx#1x^^b9\else
  \ifx#1z^^bb\fi\fi\fi\fi\fi\fi\fi\fi\fi}%
\input plhyph.tex
```

Podobně jsem zatím řešil němčinu, kde je v souborech `hyph-de.tex`, resp. `hyph-de2.tex` psáno `\input dehyphn.tex`, resp. `\input dehypht.tex`. Skutečné vzory dělení jsem tedy do nových souborů nekopíroval. Existují dvě možnosti:

- V nových souborech `hyph-`⟨*jazyk*⟩`.tex` použít `\input` ⟨*originální-vzor*⟩.
- Kompletně zkopírovat originální vzor do nového souboru `hyph-`⟨*jazyk*⟩`.tex`.

Každá možnost má svá pro a proti. První možnost má výhodu v tom, že pokud přijde upgrade vzorů dělení nějakého jazyka, pak tento upgrade můžeme rovnou použít v balíčku LANG, aniž bychom o tom třeba vůbec věděli. Tato možnost má ale nevýhodu, že pokud se autoři originálních vzorů rozhodnou pro změnu názvu souboru nebo přidají do souboru makro, které se nebude snášet s balíčkem LANG, pak nám načítání těchto vzorů přestane fungovat. Zatím nevím, která z uvedených dvou možností je lepší a ke které se nakonec přikloním.

Další dilema v souvislosti se vzory dělení spočívá v tom, zda je lepší ponechat vzory dělení 8bitové nebo raději je psát v zobákové konvenci (např. `^^ab`). Připadá mi, že zobáková konvence je lepší. Vzory dělení načítáme v době iniTEXu a tudíž máme zaručeno, že nebude nějaký uživatel měnit před načtením vzorů kategorii znaku `^`. Na druhé straně omylem zavedené TCX tabulky nebo jinak modifikovaný `xord/xchr` vektor v době iniTEXu může zcela znehodnotit načítání 8bitových vzorů dělení slov.

Balíček IENC

Tento balíček se stará o případné překódování vstupního textu na aktuálně použité kódování fontů. Úzce při tom spolupracuje s balíčkem LANG, který přepíná kódování fontů, a při tomto přepínání samozřejmě spolupracuje s balíčkem OFS.

Na rozdíl od balíčků OFS a LANG není IENC zaveden do formátu. Uživatel si může překódování řešit po svém a nemusí využít IENC. Například si uživatel nastaví TCX tabulky a použije jen jediné kódování fontů.

Pokud chce uživatel využít služeb balíčku IENC, pak na začátek svého dokumentu může napsat:

```
\input ienc
\ienc [cp1250]
...
```

V tomto příkladu dává uživatel najevo, že má vstupní dokument kódovaný podle CP1250. Balíček IENC si ověří, zda je dostupný encTEX a pokud ano, řeší případné překódování pomocí něj. Jinak se sníží k tomu, že některé znaky nastaví jako aktivní (podobně, jako `inputenc`, ale narozdíl od něj nenastavuje nutně všechny znaky jako aktivní. Ty pozice, které mají stejné vstupní i fontové kódování, zůstávají neaktivní.

Kdykoli balíček LANG je nucen změnit kódování fontů, pošle o tom zprávu balíčku IENC a ten podle toho pozmění překódovací strategii. V tom se významě liším od LATEXu, kde rozhraním mezi `inputenc` a `fontenc` jsou na kódování nezávislé TEXové sekvence. Změna kódování fontů je tam tedy nezávislá na činnosti balíčku `inputenc`. Důvod, proč jsem tuto strategii nepřejal, je zřejmý: balíček IENC se snaží, pokud to jde, o přímočaré překódování (například pomocí encTEXu), aby bylo možno například přidělovat znakům použité abecedy odpovídající kategorie.

Balíček IENC rozlišuje dva druhy vstupního kódování: jednobytové a vícebytové. Při jednobytovém kódování se předpokládá 92 ASCII znaků na svých pevných pozicích a zbylé znaky do celkového počtu 256 se mohou v jednotlivých kódováních měnit. Změna se může provést pomocí encTEXu a pokud ten není dosažitelný, pak IENC vypíše varování a nastaví překódování pomocí aktivních znaků.

Přestože není encTEX dosažitelný, IENC si překontroluje stav `xord` vektoru TEXu a překódovací metody naváže na výstup z tohoto vektoru. Kontrolu provede docela jednoduše. Načte vzorek znaků ze souboru `iencdef.tex`. V tomto souboru bude definováno makro `\bytechars`, ve kterém budou všechny znaky s jednotlivými kódy uspořádány vzestupně jeden za druhým. Pokud je `xord` vektor nastaven jinak, než jedna ku jedné, makra balíčku IENC to odhalí: například 240. znak makra `\bytechars` nemá kód 240, ale má třeba kód 190. To znamená, že `xord` vektor transformuje kód 240 na kód 190. Pokud nyní uživatel označí vstupní kódování souboru, ze kterého plyne, že třeba znak s kódem 240 je písmeno Ž, nastaví makro IENC tranformaci znaku s kódem 190 na Ž a nikoli 240 na Ž. Takže IENC ctí toto pořadí překódování:

$$\text{soubor} \xrightarrow{\text{xord}} \xrightarrow{\text{IENC}} \text{TEXové fonty, } \texttt{dvi}$$

Důsledek: napíše-li uživatel např. `\ienc[cp1250]` a jeho vstupní soubor skutečně je v tomto kódování, pak IENC překóduje z tohoto kódování správně na vnitřní kódování fontů *nezávisle* na stavu `xord` vektoru v TEXu (tj. nezávisle na nastavené TCX tabulce ve web2cTEXu). Tato schopnost například současnému balíčku `inputenc` z LATEXu chybí: tento balíček vyžaduje nastavení `xord` vektoru jedna ku jedné a pokud toto není splněno, vede to ke zbytečnému matení uživatelů a ke kódovacímu guláši.

Pokud jde o vícebytové kódování, v úvahu přichází asi jen UTF-8. V tomto kódování můžeme zapsat znaky abeced celého světa současně do jednoho souboru. To nás v 8bitovém TEXu vede k dilematu: načíst definice které transformují UTF-8 kódy na TEXové sekvence pro všechny znaky (je jich mnoho desítek tisíc!) nebo raději šetřit pamětí TEXu a načítat „úseky“ těchto definic podle potřeby a podle použitého jazyka? Po delším váhání jsem se rozhodl ke druhé variantě řešení. Popíšu stručně, jak to je uděláno.

Pro zpracování UTF-8 kódování předpokládám použití encTEXu Narazí-li TEX na zatím nedeklarovaný UTF-8 kód na svém vstupu, vypíše o tom varování na terminál a do logu. Tomu může uživatel předejít tak, že přepne v balíčku LANG do odpovídajícího jazyka. To způsobí nejen nastavení odpovídajích fontů, ale také případné načtení dalšího úseku deklarací UTF-8 kódů, které souvisí s uvedeným jazykem.

Balíček IENC mám zatím jen ve fázi „ideového návrhu“. Plno věcí ještě zbývá dořešit. Například jak přehledně členit deklarace UTF-8 kódů a jakým softwarem tyto deklarace generovat přímo z UNICODE specifikací na `www.unicode.org`.

Odkazy

1. `ftp://math.feld.cvut.cz/pub/olsak/oktex`.
2. Karl Berry. *Fontname*, `/texmf/doc/fontname/fontname.pdf`

3. Petr Olšák. *OFS: Olšákův fontový systém.* 2001, 2004. Dokumentace k balíku je v souborech `ofsdoc.tex`, `ofsdoc.pdf`.
4. Petr Olšák. *LANG: Multijazykový balíček pro plain.* 2004. Dokumentace k balíku je v souborech `ofslang.tex`, `ofslang.pdf`.
5. Petr Olšák. *Novinky v OFS.* in: SLT 2004 – sborník semináře o Linuxu a TEXu.

Jiné rodiny písem pro sazbu matematiky

Karel Horák

Účelem tohoto příspěvku je poukázat na možnosti, jež se díky tvořivé TEXové komunitě neustále rozšiřují. Osobně mě před časem potěšily dvě propracované kolekce matematických písem a znaků k Timesu a k Palatinu od Younga Ryua, a to jsem netušil, že během přípravy článku narazím na stejně dobře připravenou kolekci fourier doplňující mou oblíbenou rodinu Utopia.

Computer Modern trochu jinak

Začnu krátkým přehledem možností, které se nabízejí při použití Computer Modern. Kromě doplnění dvou znakových sad v $\mathcal{AMS}$-TEXu existuje ještě skupina řezů vzniklých ze spolupráce Donalda Knutha se světoznámým typografem Hermannem Zapfem na stanfordském projektu *Euler*. Sám Knuth pak použil tyto sady při sazbě veskrze nádherné knihy *Concrete Mathematics* s upraveným řezem Computer Modern nazvaným Concrete. Na první pohled nás zaujme svislou kresbou znaku integrálu (ten se v klasické sazbě hojně používal!). Také výběr různých typů velkých závorek je menší. Knuth zveřejnil i svá makra `gkpmac`, lze je tedy pro sazbu využít. Existuje ovšem i jednodušší možnost (poněkud méně poučná), a to použít virtuální fonty. Vše je už uděláno v balíku `eulervm` Waltra Schmidta.

Stejně jako změnou parametrů dostaneme z Computer Modern řezy Concrete, lze analogicky vytvořit i příslušné sady matematické. To před časem udělal Ulrik Vieth (včetně doplňkových znakových sad z $\mathcal{AMS}$). Balíček se najde v archivu pod názvem `concmath`. Obě kombinace lze posoudit z krátké ukázky běžného matematického textu na následující straně.

Za zmínku stojí i pokus o bezpatková písma nazvaná Computer Modern Bright a doplněná analogickými řezy matematickými včetně $\mathcal{AMS}$-TEXových symbolů. Tady ovšem zatím zůstává stejná patková zdvojená latinka jako v `msbm`, která se k jinak lehkému bezpatkovému písmu hodí ještě méně než ke Computer Modern.

Diracova rovnice pro volnou částici:

$$i\hbar\frac{\partial\psi}{\partial t} = \hat{H}_f\psi = \left(\mathbb{C}\hat{\vec{\alpha}}\hat{\vec{p}} + m_0\mathbb{C}^2\hat{\beta}\right)\psi$$

Integrální reprezentace Besselovy funkce $J_\nu(z)$:

$$J_\nu(z) = \frac{1}{\pi}\sum\int_0^\pi \cos(z\sin(\theta) - \nu\theta)\,d\theta - \frac{\sin(\nu\pi)}{\pi}\int_0^\infty e^{z\sinh t - \nu t}\,dt \quad (|\arg z| < \tfrac{1}{2}\pi)$$

Rozvoj Coulombovy vlnové funkce pomocí Besselových-Cliffordových funkcí:

$$F_L(\eta,) = C_L(\eta)\frac{(2L+1)!^{-L}}{(2\eta)^{2L+1}}\sum_{k=2L+1}^{\infty} b_k t^{k/2} I_k\left(2\sqrt{t}\right)$$

with $b_{2L+1} = 1$, $b_{2L+2} = 0$ and $4\eta^2(k-2L)b_{k+1} + kb_{k-1} + b_{k-2} = 0$.
Identity s odmocninami:

$$\left\{\sqrt{\frac{1}{2}}\cdot\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}\cdot\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}}\cdots\right\} = \frac{2}{\pi}$$

Diracova rovnice pro volnou částici:

$$i\hbar\frac{\partial\psi}{\partial t} = \hat{H}_f\psi = \left(\mathbb{C}\hat{\vec{\alpha}}\hat{\vec{p}} + m_0\mathbb{C}^2\hat{\beta}\right)\psi$$

Integrální reprezentace Besselovy funkce $J_\nu(z)$:

$$J_\nu(z) = \frac{1}{\pi}\sum\int_0^\pi \cos(z\sin(\theta) - \nu\theta)\,d\theta - \frac{\sin(\nu\pi)}{\pi}\int_0^\infty e^{z\sinh t - \nu t}\,dt \quad (|\arg z| < \tfrac{1}{2}\pi)$$

Rozvoj Coulombovy vlnové funkce pomocí Besselových-Cliffordových funkcí:

$$F_L(\eta,\varrho) = C_L(\eta)\frac{(2L+1)!}{(2\eta)^{2L+1}}\varrho^{-L}\sum_{k=2L+1}^{\infty} b_k t^{k/2} I_k\left(2\sqrt{t}\right)$$

with $b_{2L+1} = 1$, $b_{2L+2} = 0$ and $4\eta^2(k-2L)b_{k+1} + kb_{k-1} + b_{k-2} = 0$.
Identity s odmocninami:

$$\left\{\sqrt{\frac{1}{2}}\cdot\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}\cdot\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}}\cdots\right\} = \frac{2}{\pi}$$

Výběr řečtiny

Mezi hojně používané symboly patří řečtina. Ta je standardně součástí matematických rodin, ale občas postrádáme její antikvové řezy, chceme-li konstantu π odlišit od proměnné… Také fyzikové občas potřebují pár neskloněných písmenek, když chtějí být důslední v používání kurzívních písmen jen pro proměnné.

Pro výběr vhodné řečtiny existuje několik hledisek. Aby písmo dobře ladilo s textovou antikvou, je potřeba brát zřetel na přibližně stejnou výšku malých písmen, na stejnou váhu řezu i obdobnou kresbu.

Obvykle je výška malých řeckých písmen menší a výška velkých naopak větší než odpovídající antikva. Rodina Computer Modern postrádá antikvovou řečtinu, vybírat však můžeme z několika variant textové řečtiny, jež jsou z Computer Modern odvozeny (já mám v oblibě jednu z prvních takových od Silvia Levyho). Dlouho jsem postrádal verzi ve formátu Type 1, teď se jí dost blíží řečtina nazývaná Ibycus, volně dostupná v archivech.

Nejpoužívanější evropské písmo

K nejčastěji používanému písmu při sazbě matematických knih a časopisů patří bezesporu Times. K základní sazbě matematiky bychom ve velké většině asi vystačili s řezem Symbol (nyní je standardní součástí 35 základních PostScriptových písem; jeho varianta pod názvem StandardSymL z volného balíčku Type 1 písem od URW se od něj liší jen v malých detailech a obsahuje už i symbol pro euro). Nezřídka se ovšem setkáme i s kombinací Times a Computer Modern, nemůže to být ale nikdy dobrá kombinace, protože obě rodiny mají naprosto odlišné vlastnosti jak v poměrech horních a středních dotažnic, tak v kresbě (odborníci obvykle hovoří o dynamické a statické antikvě). Jak to vypadá, vidíme v následující slibně začínající větě

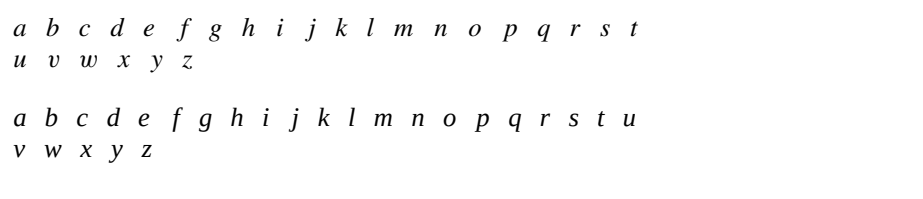
Je dán trojúhelník ABC se stranou a délky 4 cm...

Ten rozdíl nepřehlédneme, ani když se matematika vyskytuje převážně jen ve vzorcích vysazených na samostatném řádku.

Nicméně se taková kombinace snese i u pěkné knihy, když si představíme hrůzy, které vznikají u knih kompletně připravených do tisku nebohými autory při použití některých komerčních textových editorů (v objemné monografii [4] je pro sazbu vzorečků důsledně používáno Computer Modern, přestože v textu se střídá Times s Helvetikou). Oč hůře vypadají některé publikace stejného vydavatele připravené typografickým systémem bez jakéhokoli ohledu na typografii.

Symbolu ovšem chybí skloněná řečtina, což se obvykle řeší elektronickým skloněním liter. Kvalitou asi nejlepší (včetně TEXové podpory s metrikami obsahujícími všechny potřebné informace [fontdimeny] pro sazbu vzorečků) je balíček

MathTime (Micropress, Inc.), dnes doplněný i o další řezy se skriptem a kompletní řečtinou. Pro základní tři matematické řezy MathTime existuje téměř rovnocenná náhrada nazvaná Belleek (R. Kinch, TrueTEX), která je volně dostupná. Předpokládá však, že soubor matematických znaků doplníme normální textovou kurzívou z Times Italic, která má tu vadu, že její v nelze prakticky odlišit od řeckého ν. Tuto vadu ovšem MathTime nemá, jak je vidět z následující ukázky (v dolní polovině obrázku je matematická kurzíva z Mathematiky).



Protože TEX asi nejvíce používají matematici, nemůžeme se nezmínit o písmech dodávaných spolu s věhlasným matematickým softwarem *Mathematica*, zvlášť když jak známo podporuje formátování dokumentů v TEXu. Jeho součástí je dnes pět znakových sad ve verzi vycházející z Timesu v normální a polotučné verzi a dalších pět sad neproporciálních (analogických Courieru), jež dobře korespondují se zápisem vzorečků na obrazovce počítače.

Zmíněný problém s kurzívním v v matematických vzorcích, kdy nepozorný čtenář ztrácí jistotu, zda se jedná o v či ν, nemá ani balíček TX Fonts zmíněný v úvodu. Tedy přesněji řečeno, uživatel si může vybrat, zda chce v matematice v takové či onaké. Také výběr matematických symbolů je veliký, kromě skupin analogických `msam` a `msbm` nabízí ještě třetí, a nejen to. Samozřejmostí je řečtina svislá i skloněná, zdvojená latinka, fraktura i kaligrafická písmenka.

Analogický balíček PX Fonts téhož autora zas umožňuje použít k sazbě matematiky Palatino. Skupiny znaků jsou stejně bohaté a opět máme na výběr ze dvou matematických kurzív podle tvaru kresby písmene *g*. Protože Palatino (vytvořil je v roce 1949 Hermann Zapf) patří mezi opravdu pěkná písma, je možnost použít je pro sazbu matematických textů velkorysým darem. Vzhled sazby s oběma balíčky můžete posoudit na dalších dvou ukázkách.

Ve všech zmíněných případech jsem mlčky předpokládal, že není třeba řešit problém kvalitního textového písma. To ovšem zdaleka není samozřejmost.

Odbočka o problému s českými akcenty

Times je možná jedním z nejvíce používaných písem přinejmenším v západní části zeměkoule, takže je vlastně zvláštní náhoda, že první svazek *Art of Computer Programming* byl vysazen takovým obskurním písmem, z kterého nakonec díky Knuthovi vznikla rodina Computer Modern. Times (New Roman) za necelé století své existence prošel řadou variant. Přesto se ve většině textových

Diracova rovnice pro volnou částici:

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \hat{H}_f\psi = \left(\mathbb{C}\hat{\vec{\alpha}}\hat{\vec{p}} + m_0\mathbb{C}^2\hat{\beta}\right)\psi$$

Integrální reprezentace Besselovy funkce $J_\nu(z)$:

$$J_\nu(z) = \frac{1}{\pi}\sum\int_0^\pi \cos(z\sin(\theta) - \nu\theta)\,\mathrm{d}\theta - \frac{\sin(\nu\pi)}{\pi}\int_0^\infty \mathrm{e}^{z\sinh t - \nu t}\,\mathrm{d}t \quad (|\arg z| < \tfrac{1}{2}\pi)$$

Rozvoj Coulombovy vlnové funkce pomocí Besselových-Cliffordových funkcí:

$$F_L(\eta,\varrho) = C_L(\eta)\frac{(2L+1)!}{(2\eta)^{2L+1}}\varrho^{-L}\sum_{k=2L+1}^{\infty} b_k t^{k/2} I_k\left(2\sqrt{t}\right)$$

with $b_{2L+1} = 1$, $b_{2L+2} = 0$ and $4\eta^2(k-2L)b_{k+1} + kb_{k-1} + b_{k-2} = 0$.

Identity s odmocninami:

$$\left\{\sqrt{\frac{1}{2}}\cdot\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}\cdot\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}}\cdots\right\} = \frac{2}{\pi}$$

Diracova rovnice pro volnou částici:

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \hat{H}_f\psi = \left(\mathbb{C}\hat{\vec{\alpha}}\hat{\vec{p}} + m_0\mathbb{C}^2\hat{\beta}\right)\psi$$

Integrální reprezentace Besselovy funkce $J_\nu(z)$:

$$J_\nu(z) = \frac{1}{\pi}\sum\int_0^\pi \cos(z\sin(\theta) - \nu\theta)\,\mathrm{d}\theta - \frac{\sin(\nu\pi)}{\pi}\int_0^\infty \mathrm{e}^{z\sinh t - \nu t}\,\mathrm{d}t \quad (|\arg z| < \tfrac{1}{2}\pi)$$

Rozvoj Coulombovy vlnové funkce pomocí Besselových-Cliffordových funkcí:

$$F_L(\eta,\varrho) = C_L(\eta)\frac{(2L+1)!}{(2\eta)^{2L+1}}\varrho^{-L}\sum_{k=2L+1}^{\infty} b_k t^{k/2} I_k\left(2\sqrt{t}\right)$$

with $b_{2L+1} = 1$, $b_{2L+2} = 0$ and $4\eta^2(k-2L)b_{k+1} + kb_{k-1} + b_{k-2} = 0$.

Identity s odmocninami:

$$\left\{\sqrt{\frac{1}{2}}\cdot\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}\cdot\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}}\cdots\right\} = \frac{2}{\pi}$$

Diracova rovnice pro volnou částici:

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \hat{H}_f\psi = \left(\mathbb{C}\hat{\vec{\alpha}}\hat{\vec{p}} + m_0\mathbb{C}^2\hat{\beta}\right)\psi$$

Integrální reprezentace Besselovy funkce $J_\nu(z)$:

$$J_\nu(z) = \frac{1}{\pi}\sum\int_0^\pi \cos(z\sin(\theta) - \nu\theta)\,\mathrm{d}\theta - \frac{\sin(\nu\pi)}{\pi}\int_0^\infty \mathrm{e}^{z\sinh t - \nu t}\,\mathrm{d}t \quad (|\arg z| < \tfrac{1}{2}\pi)$$

Rozvoj Coulombovy vlnové funkce pomocí Besselových-Cliffordových funkcí:

$$F_L(\eta,\varrho) = C_L(\eta)\frac{(2L+1)!}{(2\eta)^{2L+1}}\varrho^{-L}\sum_{k=2L+1}^{\infty} b_k t^{k/2} I_k\left(2\sqrt{t}\right)$$

with $b_{2L+1} = 1$, $b_{2L+2} = 0$ and $4\eta^2(k-2L)b_{k+1} + kb_{k-1} + b_{k-2} = 0$.

Identity s odmocninami:

$$\left\{\sqrt{\frac{1}{2}}\cdot\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}\cdot\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}}\cdots\right\} = \frac{2}{\pi}$$

Diracova rovnice pro volnou částici:

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \hat{H}_f\psi = \left(\mathbb{C}\hat{\vec{\alpha}}\hat{\vec{p}} + m_0\mathbb{C}^2\hat{\beta}\right)\psi$$

Integrální reprezentace Besselovy funkce $J_\nu(z)$:

$$J_\nu(z) = \frac{1}{\pi}\sum\int_0^\pi \cos(z\sin(\theta) - \nu\theta)\,\mathrm{d}\theta - \frac{\sin(\nu\pi)}{\pi}\int_0^\infty \mathrm{e}^{z\sinh t - \nu t}\,\mathrm{d}t \quad (|\arg z| < \tfrac{1}{2}\pi)$$

Rozvoj Coulombovy vlnové funkce pomocí Besselových-Cliffordových funkcí:

$$F_L(\eta,\varrho) = C_L(\eta)\frac{(2L+1)!}{(2\eta)^{2L+1}}\varrho^{-L}\sum_{k=2L+1}^{\infty} b_k t^{k/2} I_k\left(2\sqrt{t}\right)$$

with $b_{2L+1} = 1$, $b_{2L+2} = 0$ and $4\eta^2(k-2L)b_{k+1} + kb_{k-1} + b_{k-2} = 0$.

Identity s odmocninami:

$$\left\{\sqrt{\frac{1}{2}}\cdot\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}\cdot\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}}\cdots\right\} = \frac{2}{\pi}$$

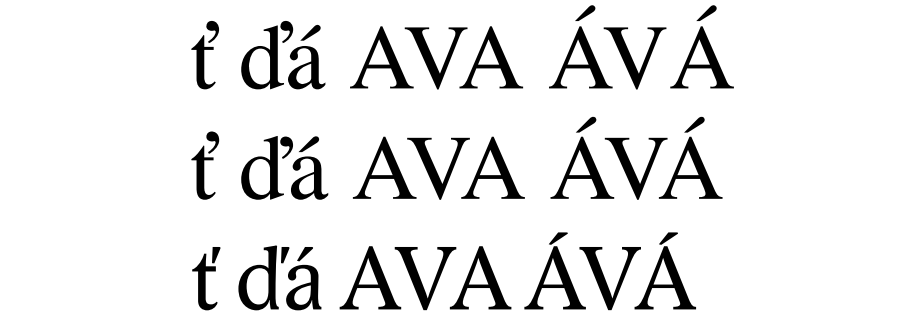
editorů v naší kotlině vyskytuje v podobě, která je mizerně počeštěná a bez odpovídajících kernů mezi písmeny s diakritikou, o chybějících ligaturách ani nemluvě.

Nejsou-li snadno k dispozici (dokonce ani za peníze!) kvalitní české verze PostScriptových písem, měli bychom se i při sazbě matematiky v češtině či slovenštině ptát, jakou cestou se ke kvalitním řezům dopracovat. Je několik možností, jak využít virtuálního mechanismu (*qdtexvpl* E. Mattese, program *accents* J. Zlatušky, *fontinst* napsaný v TEXu A. Jeffreym, *a2ac* P. Olšáka).

Virtuální mechanismus ovšem nemůže dát nikdy uspokojivý výsledek, pokud budeme sestavovat ť a ď jen pomocí apostrofu z příslušné písmové sady. Teď když jsem si pod vlivem skvělých písem ze Střešovické písmolijny začal víc všímat problému nabodeníček a poučil se i u klasiků naší písmové tvorby, dospěl jsem k názoru, že ani zmenšený apostrof není ideálem. Jako šťastné řešení vidím cestu, kterou se vydali polští kolegové, kteří mají mezi sebou šikovné nadšence a ti z volných řezů od URW vytvářejí kvalitní quazi-varianty s ohonky, bohužel ť a ď jim tam chybějí…

Je-li řeč o Timesu, nemohu ovšem nepřipomenout, že dnes máme díky Františku Štormovi a jeho Střešovické písmolijny k dispozici kvalitní českou variantu pod názvem Lido.

Pro porovnání se podívejme, jak se s oběma problematickými písmeny vyrovnává verze Timesu z balíku *Corel 3*, řez z české verze Adobe Illustratoru a Štormovo Lido.



Jednou z možností, jak si úkol usnadnit, je využít virtuální mechanismus (pro ten účel se nejlépe hodí právě a2ac); pomocí *t1utils* spolu s prográmkem *tidy* na opravu Type 1 souborů lze totiž dosáhnout toho, že dostaneme `pfb` soubor bez kompozitů (tidy je nahradí skutečnými znaky) a v něm nakonec můžeme dát akcentům u ť a ď žádoucí podobu buď pomocí komerčního editoru (znám jen Fontographer), anebo pomocí volných utilit, jako je např. *pfaedit* (hodně je o něm slyšet, ale zatím ho neznám); konečně jsou tu i metody využívající

METAFONT, resp. METAPOST: *METATYPE* skupiny kolem B. Jackowského a *t1accents* P. Olšáka.

Lahůdka nakonec

Když jsem začal o tomto příspěvku přemýšlet, vůbec jsem netušil, že se v TEXových archivech objevila kompletní podpora matematické sazby pro rodinu písma Utopia navrženého v roce 1988 Robertem Slimbachem, jehož základní řezy dala volně k dispozici firma Adobe.

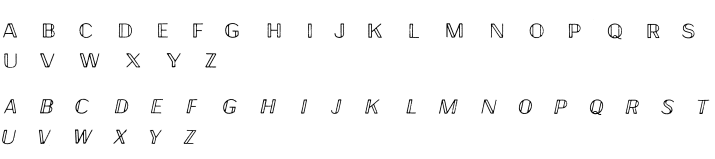
Balíček Fourier-GUTenberg obsahuje svislou i skloněnou řečtinu, krásný skript blízký kaligrafickému řezu `rsfs` (na obrázku jej můžete porovnat se skriptem z kolekce Euler a s rsfs), velice pěknou zdvojenou latinku (viz obrázek v dalším odstavci), výběr zvlášť širokých matematických akcentů, dvojité závorky a trojitou absolutní hodnotu, dvojné, trojné i eliptické integrály. Podporuje i existující expertní sady Utopie, které jsem ke svému potěšení náhodně objevil v jednom archivu na DVD z TEXLive. Zlomek možností vidíte na následující ukázce.

Většina virtuálních metrik zde zmiňovaných matematických rodin je tvořena pomocí TEXové utility *Fontinst* Alana Jeffreyho (např. metriky pro rodiny z Mathematiky). K dosažení řezů, jež se shodují přinejmenším s horní polovinou ASCII tabulky odpovídajících řezů Computer Modern, je virtuální mechanismus přímo ideálním nástrojem, jehož výsledek umožňuje vyhnout se rozdílným definicím v závislosti na použité skupině písem.

Nakonec je potřeba zmínit i utilitu *MathInst* Alana Hoeniga ([1]), která umožňuje vhodně nastavit parametry a vytvořit tak pomocí METAFONTu příslušné matematické řezy vhodné ke zvolenému textovému písmu. Přiznám se, že na její vyzkoušení jsem čas dosud nenašel.

Zdvojená latinka

Tato tzv. zdvojená latinka (blackboard označuje v angličtině tabuli, snad že někomu připomíná jednoduchá písmenka psaná křídou na tabuli?) nepatří rozhodně mezi novinky počítačového věku. To mohu ostatně doložit ukázkou z knihtiskového vzorníku [2]:



S první verzí TEXu, s kterou jsem přišel někdy v roce 1983 do styku, byly spojeny dvě znakové sady `msxm` a `msym` z $\mathcal{AMS}$-TEXu, které byly později nahrazeny analogickými `msam` a `msbm`. Ty se až na jeden či dva přidané znaky s původními

$\mathcal{A}\ \mathcal{B}\ \mathcal{C}\ \mathcal{D}\ \mathcal{E}\ \mathcal{F}\ \mathcal{G}\ \mathcal{H}\ \mathcal{I}\ \mathcal{J}\ \mathcal{K}\ \mathcal{L}\ \mathcal{M}\ \mathcal{N}\ \mathcal{O}\ \mathcal{P}\ \mathcal{Q}$
$\mathcal{R}\ \mathcal{S}\ \mathcal{T}\ \mathcal{U}\ \mathcal{V}\ \mathcal{W}\ \mathcal{X}\ \mathcal{Y}\ \mathcal{Z}$

$\mathscr{A}\ \mathscr{B}\ \mathscr{C}\ \mathscr{D}\ \mathscr{E}\ \mathscr{F}\ \mathscr{G}\ \mathscr{H}\ \mathscr{I}\ \mathscr{J}\ \mathscr{K}\ \mathscr{L}\ \mathscr{M}\ \mathscr{N}\ \mathscr{O}$
$\mathscr{P}\ \mathscr{Q}\ \mathscr{R}\ \mathscr{S}\ \mathscr{T}\ \mathscr{U}\ \mathscr{V}\ \mathscr{W}\ \mathscr{X}\ \mathscr{Y}\ \mathscr{Z}$

$\mathscr{A}\ \mathscr{B}\ \mathscr{C}\ \mathscr{D}\ \mathscr{E}\ \mathscr{F}\ \mathscr{G}\ \mathscr{H}\ \mathscr{I}\ \mathscr{J}\ \mathscr{K}\ \mathscr{L}\ \mathscr{M}\ \mathscr{N}\ \mathscr{O}$
$\mathscr{P}\ \mathscr{Q}\ \mathscr{R}\ \mathscr{S}\ \mathscr{T}\ \mathscr{U}\ \mathscr{V}\ \mathscr{W}\ \mathscr{X}\ \mathscr{Y}\ \mathscr{Z}$

Diracova rovnice pro volnou částici:

$$\mathrm{i}\hbar\frac{\partial\psi}{\partial t} = \hat{H}_f\psi = \left(\mathbb{C}\hat{\vec{\alpha}}\hat{\vec{p}} + m_0\mathbb{C}^2\hat{\beta}\right)\psi$$

Integrální reprezentace Besselovy funkce $J_\nu(z)$:

$$J_\nu(z) = \frac{1}{\pi}\sum\int_0^\pi \cos(z\sin(\theta) - \nu\theta)\,\mathrm{d}\theta \\ - \frac{\sin(\nu\pi)}{\pi}\int_0^\infty \mathrm{e}^{z\sinh t - \nu t}\,\mathrm{d}\,t \quad (|\arg z| < \tfrac{1}{2}\pi)$$

Rozvoj Coulombovy vlnové funkce pomocí Besselových-Cliffordových funkcí:

$$F_L(\eta,\varrho) = C_L(\eta)\frac{(2L+1)!}{(2\eta)^{2L+1}}\varrho^{-L}\sum_{k=2L+1}^{\infty} b_k t^{k/2} I_k\left(2\sqrt{t}\right)$$

with $b_{2L+1} = 1$, $b_{2L+2} = 0$ and $4\eta^2(k-2L)b_{k+1} + k\,b_{k-1} + b_{k-2} = 0$.

Identity s odmocninami:

$$\left\{\sqrt{\frac{1}{2}}\cdot\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}\cdot\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}+\frac{1}{2}\sqrt{\frac{1}{2}}}}\cdots\right\} = \frac{2}{\pi}$$

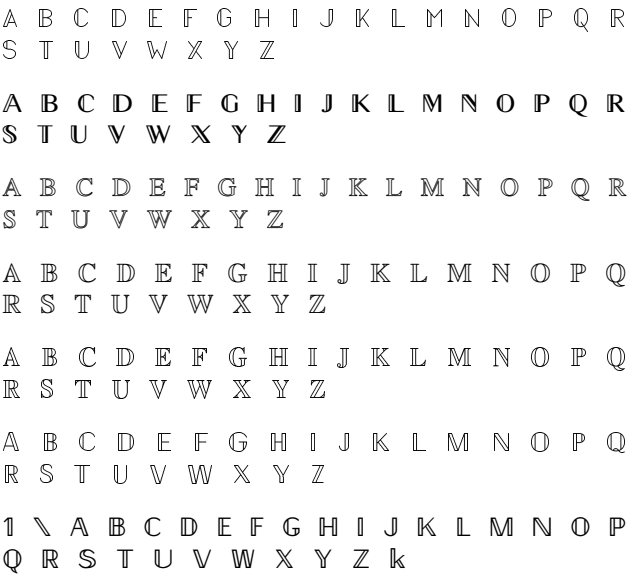
naprosto shodují s výjimkou kresby právě této zdvojené latinky. Marně jsem pátral po původních METAFONTových zdrojích, teprve později se ukázalo, že existovaly v počáteční, dnes už nepoužívané versi METAFONT79, která pak byla Knuthem zásadně přepracována. Dodnes se mi nepodařilo zjistit, zda ta změna byla natolik zásadní, že by bylo pracné převést původní kód do nového jazyka, a hlavně proč tehdejší svou strohostí pro matematiku mnohem vhodnější řez byl v nové versi nahrazen podobným, leč příšernými patkami zkažený. Dnes existuje celá řada podobných znakových sad, některé pokusy zůstávají u patek, ale vycházejí alespoň víc z originálních Computer Modern. Uvedené sadě knihtiskové

dnes asi nejvíce odpovídá zdvojená latinka ze šesté znakové sady MathPi od Adobe nebo LucidaArrows z rodiny Lucida (Bigelow & Holmes Inc.).

Pro svou vlastní potřebu jsem nakonec celou množinu zdvojených verzálek postupně v METAFONTu zrekonstruoval úpravou kódu v dnešních `msbm` a doufal, že se snad časem dostanu i k tomu převést METAFONT kód do stále více i v naší obci žádaného PostScript tvaru (tedy do Type 1). Ovšem po prozkoumání novinek na posledním TEXLive se mi taková snaha už jeví jako zbytečná, zdvojená latinka z balíčku fourier odpovídá myslím původní sadě velmi přesně. Na následující ukázce je postupně zdvojená latinka z `bbold`, z Mathematiky, z TX fontů, `msbm`, `concretemaths`, `msbm` upravené dle původních `msym` a z fouriera.

Tři poznámky na závěr

Ve snaze přiblížit se při psaní tohoto příspěvku co nejvíce aktuálnímu stavu prošel jsem velké množství souborů z posledního vydání TEXLive a pročetl mnoho stran dokumentace. Jednu chvíli jsem tak začal koketovat s využitím OFS pro sazbu ukázek, i když osobně se dívám skepticky na snahy mít zavedeno co nejvíce symbolů — chci-li nakonec nějaký méně běžný symbol použít, rozhodně jeho zavedený název v hlavě nenajdu, což znamená, že se stejně musím podívat do nějaké znakové tabulky (nebo si ji aspoň vysadit, jak OFS dle



manuálu umožňuje), tam si zároveň přečtu příslušný číselný kód a název si zvolím podle momentální chuti… Pro snadné zavedení znaku dobře poslouží `\newsymbol` z $\mathcal{AMS}$-TEXu. Hned první zkušenost žel ukázala, že má skepse je oprávněná: nenašel jsem způsob, jak jednoduše z OFS s volbou AMS vydobýt zdvojené C (a to jsem si naivně myslel, že postačí dodržet způsob psaní obvyklý v $\mathcal{AMS}$-TEXu). Líbí se mi však idea vnést do používání písem trochu pořádek. Osobně jsem nikdy nepodlehl pokusům o standardizaci názvů pomocí osmi písmen — dávám jim prostě názvy, z kterých je na první pohled patrno, o jaké písmo jde.

Zmíním se ještě o jedné skupině písem, která jistým způsobem s TEXem souvisejí: Pro uživatele wordu existuje aplikace, která se nazývá MathType. Dovede zajímavé věci. Kromě toho, že zdá se nahradí editor rovnic, umí zatím nedostupné matematické vzorečky jakkoli ve wordu vytvořené převést do TEXové podoby. Pár věcí jsem takto převedl, a i když vznikne trochu nepřehledná změť se spoustou zbytečných závorek, ne nepodobná zdroji z tzv. Scientific wordu, při větším počtu zmíněných je to vítaná pomoc. Samozřejmě jsem hned zapátral v příslušném adresáři s písmy a vytiskl si přehledné tabulky se všemi charaktery, načež jsem zjistil, že se vesměs jedná o znaky z volně šířených TEXových písem seskupených do rodiny vznešeně nazvané Euclid a podepsané © Design Science, Inc, 1999. No to se to dělá vědecký design z cizích hlav…

Pro práci s novými písmenky ve formátu Type 1 existuje užitečný PostScriptový prográmek `allchar.ps` (v původní podobě nesl název allfont.ps), v němž jsem po drobných úpravách vzhledu stránky nahradil řádek

```
JmenoFontu /DoFont
```

načtením externího souboru

```
% run the external list of fonts
(list.ps) run
```

který mi dle seznamu vytvořeného programem *pfbnames.exe* a přesměrovaného do souboru `dir.!` v daném adresáři napíše TEX zavolaný na `allchar.tex`:

```
\nopagenumbers
\newwrite\fmap   \immediate\openout\fmap=fontmap.
\newwrite\psshow \immediate\openout\psshow=list.ps
\newwrite\dpr    \immediate\openout\dpr=psres.dpr
\newread\list    \openin \list=dir.!
\def\iterate{\let\next\relax \body
              \let\next\iterate \fi\next}
\def\MakeFont #1.#2: /#3 {%
    \write\fmap{/#3  (#1.#2);}
    \immediate\write\dpr{#3=#1.afm}
    \write\dpr{#3=#1.pfb}
    \write\psshow{/#3 DoFont}}
\let~\relax

\loop\read\list to \data
\ifeof\list\else\expandafter\MakeFont\data
\repeat

\end
```

Odkazy

1. Alan Hoenig: *TEX unbound*. Oxford University Press, Oxford-New York 1998.
2. *Vzorník matematických symbolů a značek. Písmo Times New Roman, série 569, 327 a 334*. Knihtisk 5.
3. Petr Olšák: *OFS: Olšákův fontový systém*. 2004
4. Heinz-Otto Peitgen, Hartmut Jürgens, Dietmar Saupe: *Chaos and Fractals. New Frontiers of Science*. Springer-Verlag, 1992.

Slovenské vzory dělení slov: čas pro změnu?

Petr Sojka

Dělení slov neboli algoritmická segmentace velké množiny řetězců nějakého jazyka je problém častější než by se na první pohled zdálo. Pro volně šiřitelné slovenské dělení slov zatím existuje pouze řešení vycházející z definice slabiky ve slovenštině, bez rozsáhlého pokrytí výjimek. Z více než miliónu shromážděných a rozdělených slov se podařilo vygenerovat programem PatGen nové volně šiřitelné vzory, které se s nepravidelnostmi jazyka vyrovnávají lépe než dosud dostupné řešení. Výsledek je použitelný nejen v distribucích TEXu, ale i v dalších systémech jako například OpenOffice.org. Použité a diskutované techniky bootstrappingu, stratifikace a generování vzorů jsou použitelné při řešení širokého spektra dalších „segmentačních“ aplikací.

Klíčová slova: dělení slov, segmentace, PatGen, přebíjející vzory, bootstrapping, stratifikace

Motivace

Dělení slov je v jádru všech aplikací pro zpracování textů. Na kvalitě použitého algoritmu dělení slov závisí množství ruční práce při řádkovém zlomu sazby. Stále častější jsou aplikace, kdy kontrola zlomu se neprovádí vůbec: databázové publikování, dávkové zpracování XML dat může sloužit jako příklad. O to větší je poptávka po kvalitním dělení slov. Obvyklé požadavky na algoritmus dělení slov jsou tyto:

rychlost: při optimalizaci zlomu celého odstavce naráz je potřeba najít dělení *všech* slov v odstavci.

přesnost: algoritmus neoznačí chybně švy slov pro rozdělení.

úplnost: algoritmus najde všechna možná dělení slov.

rozšiřitelnost: algoritmus umožní uživatelem specifikované výjimky – například slova cizího jazyka dle pravidel dělení tohoto jazyka.

adaptivita: jelikož se živé jazyky vyvíjejí (nedávná reforma pravopisu v Německu), je potřebné nemít algoritmus „zadrátovaný“ a draze optimalizovaný tak, že při změně jazyka se musí začínat úplně znova.

parametrizovatelnost: algoritmus umožní jiné chování dle charakteru použití (v úzkých sloupcích například umožní jen dva znaky na novém řádku místo obvykle žádaných tří).

minimální paměťové nároky: aplikace typu zalomení zpráv na displeji mobilního telefonu je třeba navrhovat s minimálními paměťovými, a tedy energetickými nároky.

Základním problémem tedy je vytvořit algoritmus pro zvolený jazyk, který v maximální míře splňuje výše uvedené požadavky. Tento článek popisuje přístup řešení tohoto problému pro slovenštinu a krátce diskutuje výsledek ve formě nových vzorů dělení slov pro slovenštinu.

Stávající stav slovenského dělení slov

Uživatelé sázecího systému TEX používají při sazbě výsledky dizertační práce [16] Knuthova studenta Franka Lianga. Liang navrhl na jazyku nezávislý popis dělení slov, který splňuje většinu výše uvedených požadavků. Dále implementoval program PatGen [17], který umožňuje tento popis generovat ze slovníku již rozdělených slov.

Brzy po rozšíření TEXu do Československa začala být otázka dělení aktuální a vznikly první verze slovenských a českých vzorů dělení [14, 15]. Obě byly psány ručně, bez použití programu PatGen. Ručně psaná pravidla zachycují základní charakteristiky dělení, tedy například slabičný princip, definují co je to slabika. U etymologického přístupu k dělení slov, který je respektován v britské angličtině a standardizován nakladatelstvím Oxford Universitzy Press, však je téměř každé slovo výjimkou a proto je častější generování vzorů ze slovníku. Většina jazyků však oba přístupy kombinuje, ctí se zejména zlom na švech složených slov oproti slabičnému principu. Hranice mezi cítěním složeného slova dle jeho etymologie však může být diskutabilní: máme dělit slabičně *ro-zum* či „etymologicky“ a konzervativně *roz-um*?

Detailní vysvětlení principu přebíjejících vzorů je v [13, příloha H] a v článcích [2, 22, 3]. Zjednodušeně řečeno, vzory specifikují kontextová pravidla, která mezi sebou soutěží o každou mezipísmennou pozici ve slově, a určují, zda na ní dělit či ne. Pravidla specifikují na základě různě širokého kontextu výjimky, výjimky z výjimek,.... Vzájemně se přebíjejí – může existovat několik úrovní priorit vynucení či potlačení dělení slova. Vítězí „nejsilnější“ pravidlo (s nejvyšší prioritou) pro každou pozici ve slově: k pozici se může vyjádřit pravidlo v každé úrovni.

Stávající verze slovenských vzorů dělení existuje ve verzi 2.0 z 24. 4. 1992 (soubor `skhyph.tex`):

```
\patterns{
 % samohl\'asky
a1
\'a1
\"a1
e1
...
auto4rk
auto4rs
5b4lah
5b4ledn
5b4lesk
...
 %koncovky
```

```
 % dvojice spoluhl\'asok
2b1b
2b1c
2b1\v c
2b1d
2b1\v d
...
 % 6 spoluhl\'asok
3c4v4r4n3g4n
3\v s4k4v4r4k3n
3\v s4k4v4\'r4k3n
 % koncovka -n\'y
k4\v c3n\'y.
k4\v c3n\'eho.
k4\v c3n\'emu.
k4\v c3nom.
 % slovn\'e z\'aklady
5alkohol
4b4s4\v t.
4b4s4\v t.
8c4h.
8d4z.
8d4\v z.
4c4ht4.
4j4s4\v t.
4lt.
4m4p4r.
...
 %cudzie slov\'a
akci3a2
akv\'ari3u2m
gymn\'azi3um
le2u3k\'emia
t2ri3u2mf
kli3e2nt
}
```

Z komentářů ve vzorech je vidět, jakým způsobem vzory vznikaly. Po rozgenerování vzorů popisujících slabiku jako sekvenci příslušného počtu souhlásek a samohlásek se vzory autorka snažila zachytit slabičné výjimky na začátku a konci slov a při dělení cizích slov. Lze si ale těžko představit, že by se tímto způsobem podařilo zachytit několik miliónů slovních tvarů, které ve slovenštině existují. Na švech předpon a složených slov jsou mnohé výjimky, které jdou proti základnímu slabičnému principu. Těch jsou ale tisíce, či desetitisíce, a jen s enormním úsilím by se daly vypsat všechny. Pro češtinu byly sepsány Hallerem [10], pro slovenštinu však patrně takový soupis neexistuje.

Na archívu CTAN lze nalézt vzory vytvářené jak ručně výše popsaným způsobem, tak automaticky z již rozděleného slovníku slov daného jazyka. Tento postup má z hlediska požadavků vytčených v úvodu článku mnohé výhody oproti ručně vytvořené verzi. Přístupy se také dají kombinovat: k ručně zadané množině základních vzorů se dogenerují vzory pro výjimky. Nebo naopak ex post nalezené výjimky se dají k již vygenerovaným vzorům přidat jako slova – vzory s nejvyšší prioritou (úrovní), tedy *rozum* jako `.r8o8z9u8m.`.

Generování vzorů ze slovníku rozdělených slov

Problematice generování vzorů na semináři SLT již byl věnován článek [1], proto zopakujeme jen hlavní principy a laskavého čtenáře odkážeme dále na další články věnované této a příbuzné problematice [11, 25, 20, 21].

Generování probíhá ve fázích, které se nazývají *úrovně* (anglicky *levels*). V lichých úrovních se generují pokrývací vzory, tedy vzory, které dle kontextu znaků vynucují dělení, v sudých úrovních se dělení dle kontextu zakazuje. Generované

vzory se kumulují, a výsledné chování určuje výsledná množina vzorů vygenerovaná ve všech úrovních. U většiny generovaných vzorů pro dělení slov v užívaných jazycích stačí čtyři úrovně, ale pro přehlednost, ale také nedostatek času vzory optimalizovat, je v ručně chystaných vzorech úrovní mnohem více – současné slovenské vzory jich mají například osm.

Technologie přebíjejících vzorů je natolik obecná, že její použití je možné pro většinu *segmentačních problémů*. Jako příklad může sloužit problematika segmentace řetězce thajských znaků na slova – v thajském textu nejsou slova oddělena mezerami [23].

Bootstrapping a stratifikace

V rámci bakalářské práce [18] se podařilo shromáždit z různých zdrojů[1] téměř milión slovenských slov. Dnešní výpočetní kapacity umožňují generovat vzory dělení i z takto rozsáhlých slovníků v dobách desítek minut. Časově nejnáročnější operaci – rozdělení slovníku slov pravidly daného jazyka – lze dělat pomocí předchozí verze vzorů a místa dělení slov „pouze“ zkontrolovat. Jelikož však i tato kontrola je časově náročná, lze parametry generování vhodnou heuristikou volit tak, že vygenerovaných vzorů nepokrytých slov je právě tolik, kolik je reálné jich v rozumné době ručně zkontrolovat. To značně urychluje vývoj nových vzorů technikou *bootstrappingu*.

úroveň	dobře	špatně	chybí	# vzorů	velikost
1	99,24 %	17,17 %	0,76 %	2192	
2	98,08 %	1,52 %	1,92 %	3240	
3	100,00 %	1,16 %	0,00 %	3229	
4	99,94 %	0,01 %	0,06 %	2347	56 kB

Tabulka 1. Výsledky jedné iterace bootstrappingu slovenského dělení ze slovníku 822 878 slov

Výsledky jedné z iterací generování vzorů jsou v tabulce 1.

Po několika iteracích lze provést závěrečné generování. Vhodnými parametry pro PatGen lze vygenerovat prostorově úsporné vzory za cenu nižšího pokrytí, nebo naopak maximalistické vzory s nulovou chybovostí a stoprocentním pokrytím.

[1] Bohužel výslednou množinu slov nelze volně šířit. Volně přístupný seznam slov by umožnil ještě mnohem flexibilnější vytváření variant dělicích vzorů optimalizovaných pro konkrétní projekty.

Parametry generování pro prostorově či výkonnostně optimální vzory nelze v „rozumném“ čase spočítat [16]. Optimu se však dá přiblížit vhodnou heuristikou. Vzory pro dělení americké angličtiny – soubor `hyphen.tex` v každé distribuci TEXu – je daleko od obou optim. Množství výjimek dělení slov k těmto vzorům rychle roste [4, 5, 6, 7, 8] a při tomto tempu růstu by zabraly při otištění ještě v tomto století jedno celé číslo časopisu TUGboat. Patrně z důvodu zpětné kompatibility nejsou tyto vzory nahrazeny kvalitnějšími, byť kompatibilita je při přidání výjimek do vzorů ve formátu stejně porušena. Dnešní výpočetní technika již umožňuje četné experimenty a generování opakovat s různými parametry. Vhodnými heuristikami nastavení prahů akceptace adeptů vzorů v jednotlivých úrovních generování se lze dostat na mnohem kvalitativně vyšší parametry vzorů, než které docílil před téměř čtvrtstoletím Liang. Typicky je možné za cenu mírného zvýšení velikosti vzorů docílit stoprocentního pokrytí učící množiny, nebo naopak při zadání velikostních omezení na velikost vzorů lze maximalizovat pokrytí. A to vše s nulovou chybovostí a stejnými *konstantními* výpočetními nároky při aplikaci vzorů. Jinak řečeno, počet instrukcí na nalezení dělicích švů slova je ohraničen shora konstantou, nezávisle na tom, z jak velkého slovníku vzory generujeme.

Další technikou, která se dá při generování vzorů použít, je *stratifikace*. Tato technika spočívá v tom, že se snažíme minimalizovat množinu slov k učení, aniž bychom ale přišli o funkčnost vzorů na výjimkách. Máme-li například slovník generovaný morfologickým analyzátorem, tedy známe od každého slovního tvaru slovní základ, stačí do slovníku slov zahrnout náhodně pouze pár slovních tvarů od jednoho lemmatu. Dělení koncovek se zgeneralizuje, neboť koncovkové množiny se neustále opakují a učící algoritmus bude mít dostatek učících příkladů, aby se pravidelnosti dělení konců slov naučil. Naopak se nesmí v seznamu učících slov zapomenout na negace a předpony. Dělení za první slabikou slov začínajících na *na-* *naj-*, *pre-* *pred-* apod. je nutno nahlížet jako na výjimky.

Shrnutí: čas pro změnu?

Bylo vytvořeno několik variant nových vzorů dělení pro slovenštinu. Jsou k dispozici pro testování ve FTP archívu CSTUGu v adresáři `cstug/sojka/skhyp`. Po nezbytné fázi testování předpokládáme jejich zařazení do běžných TEXových distribucí a projektu OpenOffice.org a budou šířeny bez omezujících licenčních podmínek.

Jelikož změna vzorů dělení pravděpodobně způsobí změnu zalomení již vytvořených dokumentů, je třeba být v případě rozšířeného požadavku na zpětnou kompatibilitu obezřetný. Jelikož na zálohování *úplných* zdrojů včetně zdrojů potřebných na generování formátu se obvykle zapomíná, při požadavku zpětné kompatibility je třeba zvážit všechna pro i proti a nové vzory si třeba zavést jako nový jazyk (`\language`) spolu se starými. Jsme přesvědčení, že čas pro změnu

po více než dekádě používání současných vzorů nastal a kvalita nových vzorů je dostatečným argumentem pro zavedení změny. Po té již ostatně několik let volají také uživatelé OpenOffice.org a dalších sázecích systémů, kteří dosud používají staré vzory dělení.

Odkazy

1. David Antoš a Petr Sojka.
   Generování vzorů dělení slov v UNICODE.
   V Kasprzak a Sojka [12], strany 23–32.
2. David Antoš a Petr Sojka.
   Pattern Generation Revisited.
   V Pepping [19], strany 7–17.
3. David Antoš a Petr Sojka.
   Generování vzorů pomocí knihovny PatLib a programu OPatGen.
   Zpravodaj ČSTUG, 12(1):3–12, 2002.
4. Barbara Beeton.
   Hyphenation exception log.
   TUGboat, 5(1):15, květen 1984.
5. Barbara Beeton.
   Hyphenation exception log.
   TUGboat, 6(3):121, listopad 1985.
6. Barbara Beeton.
   Hyphenation exception log.
   TUGboat, 7(3):146–147, říjen 1986.
7. Barbara Beeton.
   Hyphenation exception log.
   TUGboat, 10(3):336–341, listopad 1989.
8. Barbara Beeton.
   Hyphenation exception log.
   TUGboat, 13(4):452–457, prosinec 1992.
9. Pat Hall a Durgesh D Rao, editoři.
   Proceedings of EACL 2003 Workshop on Computational Linguistics for South Asian Languages—Expanding Synergies with Europe, duben 2003.
10. Jiří Haller.
    Jak se dělí slova.
    Státní pedagogické nakladatelství Praha, 1956.
11. Yannis Haralambous.
    A Small Tutorial on the Multilingual Features of PATGEN2.
    dostupné na CTAN jako `info/patgen2.tutorial`, leden 1994.
12. Jan Kasprzak a Petr Sojka, editoři.
    SLT 2001, Brno, Czech Republic, únor 2001. Konvoj.

13. Donald E. Knuth.
The TEXbook, volume A of *Computers and Typesetting*.
Addison-Wesley, Reading, MA, USA, 1986.
14. Jana Chlebíková.
Ako rozdělit (slovo) Československo.
Zpravodaj CSTUG, 1(4):10–13, 1991.
15. Ladislav Lhotka.
České dělení pro TEX.
Zpravodaj CSTUG, 1(4):10–13, 1991.
16. Franklin M. Liang.
Word Hy-phen-a-tion by Com-put-er.
PhD thesis, Department of Computer Science, Stanford University, USA, srpen 1983.
17. Franklin M. Liang a Peter Breitenlohner.
`PAT`tern `GEN`eration program for the TEX82 hyphenator.
dokumentace programu `PATGEN` verze 2.3 z distribuce web2c na CTAN, 1999.
18. Ján Lieskovský.
Systém pro práci se seznamy slov.
Bakalářská práce, Masarykova univerzita v Brně, Fakulta informatiky, 2003.
19. Simon Pepping, editor.
EuroTEX 2001, Kerkrade, The Netherlands, září 2001. NTG.
20. Petr Sojka.
Notes on Compound Word Hyphenation in TEX.
TUGboat, 16(3):290–297, 1995.
21. Petr Sojka.
Hyphenation on Demand.
TUGboat, 20(3):241–247, 1999.
22. Petr Sojka.
Competing Patterns for Language Engineering.
V Sojka et al. [24], strany 157–162.
23. Petr Sojka a David Antoš.
Context Sensitive Pattern Based Segmentation: A Thai Challenge.
V Hall a Rao [9].
24. Petr Sojka, Ivan Kopeček, a Karel Pala, editoři.
Proceedings of the Third International Workshop on Text, Speech and Dialogue—TSD 2000, Lecture Notes in Artificial Intelligence LNCS/LNAI 1902, Brno, září 2000. Springer-Verlag.
25. Petr Sojka a Pavel Ševeček.
Hyphenation in TEX—Quo Vadis?
TUGboat, 16(3):280–289, 1995.

Webové rozhraní pro sazbu dokumentů

Jan Přichystal, Jiří Rybička

Jedním z nedostatků systému TEX je pomalá a komplikovaná instalace a konfigurace. Tento fakt někdy způsobuje nechuť začínajících uživatelů se s tímto systémem seznámit a pracovat s ním. Tento problém je dobře znám a byl diskutován i ve Zpravodaji CSTUG v anketě uživatelů TEXu (Polách, 2003). Bylo zde zmíněno, že tento problém by pomohlo vyřešit vytvoření webového rozhraní, které by simulovalo funkčnost TEXu. Popisovaný projekt „TEXonWeb“ se snaží tyto nedostatky překonat a nabídnout prostředí pro představení vlastností sázecího systému bez nutnosti jej instalovat na vlastní počítač.

Následně lze tohoto webového stroje použít i pro zpřístupnění určitých typů dokumentů (dopisy, objednávky apod.) běžným uživatelům, přičemž centralizovaná správa pomůže zajistit správnost podle příslušných norem a doporučení.

Úvod

Netriviální instalace a konfigurace systému TEX způsobuje začínajícím uživatelům nemalé potíže. Ti často nejsou schopni instalaci tohoto systému zvládnout sami a potřebují pomoc odborníka, na tento systém zanevřou a raději se zase vrátí k programům méně kvalitním, avšak z jejich pohledu jednodušším. Přestože je již delší dobu k dispozici poměrně propracovaná distribuce TEXLive, setkáváme se často s různými problémy, jejichž společnou příčinou je značná různost operačních prostředí, v nichž se instalace provádí. Podle našich zkušeností se začátečníky je zhruba polovina z nich nucena využívat při instalaci distribuce TEXLive různých forem technické pomoci. Uvedený podíl je značný a nelze do budoucna předpokládat jeho dramatické snížení, bude-li i nadále každá distribuce koncipována jako velmi komplexní, propracovaný a vybavený systém využitelný jen hrstkou skutečných odborníků.

Pod dojmem této skutečnosti bylo navrženo zpracování projektu webového portálu simulujícího funkčnost systému TEX, který by začínajícím uživatelům nabídl prostředí systému TEX, aniž by jej bylo nutné instalovat, a který by byl snadno dostupný a použitelný kdykoliv. Přístup přes webové rozhraní byl zvolen z několika důvodů. Jedním je výhoda vysoké dostupnosti z libovolného místa a operačního systému, kde je přístup k Internetu. Další výhodou je možnost přístupu v jakémkoliv čase, neboť webový server bude neustále v provozu.

Bylo nutné se také zamyslet nad tím, co by takovýto systém měl uživatelům konkrétně nabízet, neboť je jasné, že stavu, aby byl naprosto srovnatelný s lokální instalací, nelze v podmínkách webového prohlížeče dosáhnout. Principiálně chybí například veškeré vymoženosti textového editoru, jako je zvýrazňování syntaxe nebo funkce vyhledávání a nahrazování řetězců v textu. Chybí rovněž interaktivní řízení překladu. Uživatel tedy bude mít možnost zapisovat zdrojový text, připojovat soubory dle potřeby a v omezené míře nastavovat způsob překladu.

V zásadě jsme se soustředili na dva případy:

- Uživatel chce napsat pouze kratší text (například představit vlastnosti systému TEX na několika krátkých příkladech). V takové situaci mu nepohodlí pořizování textu v prohlížeči příliš nepřekáží. Bude potřebovat prostředek pro zapisování zdrojového textu, který bude suplovat funkci textového editoru. Ve spojení s tím může vyvstat požadavek na vkládání vlastních souborů.
- Pro uživatele není problém pořídit zdrojový text na vlastním počítači, nebo ho již dokonce má hotový, a chce provést pouze jednorázový překlad TEXem na počítači, kde tato možnost není. Pravděpodobně je však k dispozici nějaký textový editor, ve kterém text může napsat a připojení k Internetu není v dnešní době nijak výjimečné.

Dalším požadavkem je možnost vytvářet dokumenty jak formátu PostScript tak PDF, což běžná instalace TEXu umožňuje. Je také potřeba ovlivňovat počet průchodů překladače. V některých případech totiž nestačí průchod jeden. Typickým příkladem požadavku na větší počet průchodů je sazba obsahu, kdy se musí informace o názvech kapitol a číslech stránek přenést do místa, kde se objeví obsah. Tento problém lze řešit minimálně dvojím překladem: při prvním průchodu se potřebné informace zaznamenají do pomocného souboru, při druhém průchodu se tyto informace z pomocného souboru vloží ve formě odpovídajících zápisů do vysázeného dokumentu.

Navržené řešení

Na vyhrazeném počítači je nainstalován překladač systému TEX a další nezbytné podpůrné utility. Jde například o program `vlna`, který opatří zdrojový text, v místech, kde je třeba, nezlomitelnými mezerami nebo o program `dvips` generující z formátu DVI formát PostScript.

Celá instalace TEXu a podpůrných programů je umístěna ve vyhrazeném prostoru, který je oddělen od vlastního operačního systému, především pak konfiguračních souborů a souborů obsahujících důležité informace, zneužitelné nepovolaným uživatelem. Odstínění je realizováno programem `chroot` a je tak zajištěna bezpečnost při nepovoleném vniknutí přes webové rozhraní.

Dalším prvkem bezpečnosti je omezení velikosti ukládaných souborů. V současné době je omezena na 1 MB. To by mělo postačovat pro vkládání obrázků, ale zároveň to ztěžuje útočníkovi zaplnit disk serveru a znemožnit tak fungování systému TEXonWeb. Funkčnost vlastního serveru by ani toto nemělo ovlivnit neboť systémové soubory jsou umístěny na jiném logickém disku.

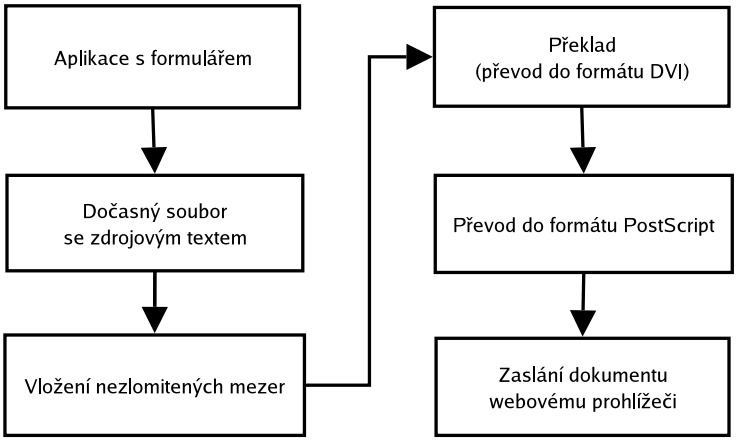
Princip fungování je v zásadě velice prostý. Využili jsme vlastnosti TEXu, že zpracovává data dávkovým způsobem, což nám umožňuje proces tvorby dokumentu algoritmizovat.

V okamžiku, kdy uživatel napíše zdrojový text dokumentu, připojí potřebné soubory, vybere typ překladače a stiskne tlačítko, odešlou se data na server. Tam je z textu obsaženého ve formuláři vytvořen dočasný soubor. Ten je opatřen příkazem `\nonstopmode` zajišťujícím neinteraktivnost překladu. Poté do něj program `vlna` dosadí na patřičná místa nezlomitelné mezery. Následně je soubor přeložen do formátu DVI a poté překonvertován do PostScriptu. V případě PDF je prováděn překlad rovnou do tohoto formátu. Výsledný tvar je poté zaslán zpět webovému klientovi, který jej zobrazí, popřípadě nabídne možnost uložení na lokální počítač. O průběhu překladu je vygenerován logový soubor, který si uživatel může prohlédnout a zjistit případné chyby. Pokud překlad nedopadne dobře a není možno výsledný tvar vygenerovat, systém o tom podá hlášení a nabídne možnost prohlédnutí logového souboru.

Jak bylo zmíněno výše, celý proces překladu probíhá na serveru ve vyhrazeném prostoru, aby se zamezilo případným nepovoleným akcím. Jde například o možnost připojení souboru `/etc/passwd` do dokumentu. Tento soubor obsahuje informace o uživatelských účtech na serveru. Takové informace by mohly případnému útočníkovi usnadnit průnik na server a provádění nekalých činností. Obrázek 1 schematicky naznačuje činnost systému.

Uživatelské rozhraní

Pro přístup k systému lze použít jakýkoliv webový prohlížeč podporující formuláře a kaskádové styly. Doporučeny jsou zejména prohlížeče Mozilla a Internet Explorer. Základem stránky je formulář s textovým polem, do kterého uživatel zapisuje zdrojový text dokumentu a opatřuje jej značkami systému TEX. Pod tímto oknem jsou tlačítka, která umožňují výstup buď ve formátu PostScript nebo PDF. Uživatel může roletkovým menu samozřejmě určit, který překladač se použije. Standardně jsou nabízeny CSLATEX, LATEX, CSPlain, TEX a to i ve variantách pro PDF. Počet průchodů překladače lze ovlivnit také, nabízeny jsou jeden, dva nebo tři. Jiné hodnoty v podstatě nemají význam. Další důležitou položkou ve formuláři je možnost připojovat vlastní soubory. Mohou to být jak obrázky, tak i jiné zdrojové soubory s textem dokumentu nebo soubory rozšiřujících maker. To umožňuje uživateli vytvořit strukturovaný dokument odděleně



Obrázek 1. Schéma činnosti navrženého systému „TEX on Web“

na vlastním počítači a poté jej pouze za použití webového rozhraní přeložit a získat potřebný výstupní formát.

Pozadí systému

Pro tento účel byl vyhrazen jeden stroj architektury IBM PC s procesorem Celeron s nainstalovaným operačním systémem Linux (distribuce Fedora), webový server Apache, programovací jazyk Perl (mod_perl) a typografický systém TEX (distribuce teTEX).

Systém je v zásadě tvořen perlovským modulem, který obsahuje metody zajišťující veškerou práci s překladem různými překladači, převodem do nabízených formátů, ukládání vlastních souborů atd. Na něj je navázán perlovský skript vytvářející uživatelské rozhraní popsané výše.

Dosavadní výsledky

V TEXové komunitě byl požadavek na takovýto projekt již několikrát vznesen, takže se dá usoudit, že o něj bude velký zájem. Důležitým faktem je, že usnadní seznamování začínajících uživatelů se systémem TEX, případně nabídne možnost jednorázového použití lidem, kteří s TEXem pracovat pravidelně nechtějí, ale jsou v situaci, kdy nemají na svém počítači TEX nainstalován nebo se tomu z nějakého důvodu chtějí vyhnout. Jde třeba o případ tvorby diplomových prací, kdy student potřebuje provést kvalitní vysazení dokumentu a nechce se seznamovat s pozadím systému. Z tohoto důvodu je nabízena možnost použití balíku maker pro sazbu diplomových prací.

Popisované řešení ještě není kompletně implementováno. Jakmile však budou práce dokončeny, bude systém zveřejněn a vystaven důkladnému testování a připomínkování. Podnětné nápady budou samozřejmě zváženy a poté zapracovány.

Uvažujeme především o možnosti upravení prostoru pro vkládání zdrojových kódů (formulářový prvek textfield), který simuluje textový editor. Chceme aby se jednak pro větší přehlednost a snadnější odhalování chyb zvýrazňovala syntaxe vkládaných textů a aby se automaticky číslovaly řádky.

Dalším uvažovaným rozšířením je možnost uložení zdrojového textu na disk lokálního počítače. Dosavadní praxe, kdy uživatel vytvořený text označí do bloku a pak jej přes schránku uloží do souboru na svém počítači, se jeví jako poněkud těžkopádná.

Poslední problém, kterým se v současné době vážněji zabýváme, je dlouhá odezva systému na pomalejších linkách. Je to způsobeno především velkým objemem přenášených vysazených dokumentů. Tento problém se výrazněji projevuje u formátu PostScript. Nabízí se možnost výsledný dokument na serveru zkomprimovat a posílat jej v tomto tvaru, což sníží velikost přenášených dat. Komprimace se samozřejmě nebude provádět implicitně, ale jen v případě, že si to uživatel bude přát.

Budeme-li ovšem chtít tímto způsobem rozšiřovat možnosti nastavování systému, nevyhneme se určitému znepřehlednění stránky. Stojí také za zamyšlení, zda nenastavovat všechny záležitosti samostatně v jiném formuláři, a na stránce, kde se vkládají data, mít jediné tlačítko pro vysazení dokumentu. Každá z variant má své výhody a nevýhody, pravděpodobně se optimální koncepce prosadí postupně podle požadavků uživatelů.

Testovací verze je přístupná na adrese `http://tex.mendelu.cz/`.

Předdefinované styly

Zpřístupnění funkční instalace systému TEX/LATEX je motivováno zejména výukovými účely. Přechodem na webový systém můžeme upustit od komplikované správy instalací na různých počítačových učebnách.

Kromě tohoto prvního a zřejmého cíle se však ukazuje, že vzrůstá počet uživatelů, kteří sáhnou po zpracování dokumentů, k nimž existuje funkční a typograficky bezchybný sázecí styl. Tyto jednoznačně pozitivní zkušenosti již existují se stylem pro sazbu diplomových prací.

Je tedy přirozené pokusit se o vytvoření sady stylů pro dokumenty jiných typů. Může se jednat například o dopisy, objednávky, nejrůznější formuláře apod. Zde se otevírá široké pole působnosti, uvědomíme-li si, že takových dokumentů je na hlavním serveru univerzity vystaveno ve formátu Microsoft DOC asi 70. Tyto dokumenty jsou však připravovány zcela nahodile, neodborně a ve své sestavě bezkoncepčně, což bohužel běžný uživatel naprosto nevnímá.

Vytvořením určité alternativy by bylo možné dokázat, že systém TEX s jeho nadstavbami není jen hračkou pro vědce-matematiky.

Dalším stupněm zjednodušujícím využití standardních dokumentů je vytvoření webového rozhraní, kde jsou připravena pole pro vyplnění požadovaných textových segmentů, z nichž se automatizovaně sestaví, vysází a vrátí uživateli bezchybný dokument odpovídající normám a typografickým zásadám. Taková nadstavba je použitelná pro takovou třídu dokumentů, u nichž není počet textových segmentů příliš variabilní, ale takových dokumentů je ve zmíněné sestavě většina.

Závěr

Projekt zpřístupňující precizní sazbu systémem TEX včetně všech nadstaveb je ve své počáteční fázi. Možnosti, které jsou nyní k dispozici, lze shrnout do těchto bodů:

- Zpřístupnění instalace začínajícímu uživateli nebo zájemci, který chce vidět základní možnosti.
- Snížení objemu údržby učebnových instalací systému TEX.
- Využití připravených stylů pro tvorbu standardních dokumentů.
- Vytvoření další obalové aplikace využívající překladu jako jádra své činnosti a zpřístupňující standardní dokumenty přes webové rozhraní.

Z prvních ohlasů lze nabýt přesvědčení, že nabízené služby mohou být pro uživatele atraktivní a že se s minimálními nároky na údržbu může takový systém s úspěchem provozovat.

Odkazy

1. ČSN 01 6910 – Úprava písemností psaných strojem nebo upravených textovými editory. Praha: Normalizační institut, 1997.
2. Seznam formulářů ke stažení. Dokument HTML s odkazy na soubory formátu Microsoft DOC. Dostupné na `http://www.mendelu.cz/formulare`. 20. května 2004.

Ligatura aneb začínáme s TEXem

Milan Šorm

Z dlouholetých zkušeností s TEXem pozoruji, že nainstalovat nějakou TEXovou distribuci na systém Windows s fungující češtinou, editorem, zobrazováním a příp. připravenou nápovědou či ukázkovými styly je pro

běžného člověka velmi obtížné. Proto jsem se rozhodl pro takovéto začátečníky, pro diplomanty naší univerzity, a příp. pro další zájemce o pořádný sázecí program připravit jednoduchou distribuci pro naučení se základům TEXu. Distribuce se skládá z minimální části TEXLive pro tvorbu PDF (`pdfcslatex`), kontroly pravopisu (`ispell`), českého volně šiřitelného textového editoru vlastní provenience (určeného právě pro TEXování) a předpřipravených stylů. Příspěvek má za úkol tento projekt představit a příp. nalákat další zájemce ke spolupráci.

Klíčová slova: ligatura, sazba, TEX, začátečník.

Úvod

Jsem TEXovým nadšencem již osm let a poznal jsem během této doby řadu různých distribucí – emTEX, MikTEX, TeTEX, TEXLive apod. Každá z těchto instalací však obsahovala dvě zásadní nevýhody, se kterými se potýkala mimo mne i řada dalších lidí:

- obtížná instalace a
- nevyhovující editor.

První nevýhoda je dána architekturou TEXových distribucí, které se snaží vyhovět všem a obsahují desítky variant vlastního TEXu, stovky fontů a maker, které jsou stejně málo využívány. To vše ve složitých adresářových strukturách (které jsou sice jasné a přehledné, ale ovšem pouze pro ostříleného TEXového guru) a často rozmístěné v několika stromech po celém diskovém prostoru. Tyto distribuce navíc snahou o podporu maximálního množství platforem a jazyků končí u nepoužitelného gigantu, který neběží pořádně nikde a o funkčnosti češtiny si můžeme na první pokus instalace (babel) nechat jen znát.

Druhá nevýhoda je dána původním posláním editoru jako obecného nástroje pro všechny druhy pořizování textů – od programování přes sazbu, psaní poznámek až po prohlížení velkých souborů, na které nestačí běžné prohlížeče. Tyto editory nejsou obvykle specificky zaměřeny na TEX, ale v nejlepším případě na sazbu obecně.

Vedením diplomových a bakalářských prací jsem navíc na univerzitě nucen spolupracovat s řadou studentů nejen informatických studijních programů, kteří by rádi využili na naší univerzitě doporučovaný a vychvalovaný TEX, kteří však ale díky náročnosti instalace a komplikovanosti editoru nemají příliš šanci se s tímto nádherným systémem sžít.

V neposlední řadě také pozoruji šťastné majitele knih určených začátečníkům (jedna z nich vyšla i v našem sdružení uživatelů), kteří plni nadšení z pěkné knížky začínají s instalací TEXu, aby po několika hodinách či dnech tento systém zavrhli, protože se jim jej nepodařilo kvalitně nainstalovat a začít používat.

Všechny výše uvedené důvody mne vedly k založení nového projektu české miniTEXové distribuce pojmenované Ligatura. Převážná většina funkcí a nápadů realizovaných v této distribuci vychází z reálných potřeb studentů při psaní diplomových prací a potřeb mé ženy při tvorbě běžné korespondence (příspěvky na konference, dopisy, posudky) nebo potřeb mých (dopisy, fakturace, osvědčení, obchodní sestavy).

Vím, že předkládaný článek, projekt i celé téma je v našem sdružení velmi kontroverzní a obvykle vyvolá řadu otázek od Proč? Nač? A proč ne mou oblíbenou distribuci HCHKRDTNTEX? Projekt předkládaný k diskuzi širší TEXové obci si neklade za cíl být jedinou českou distribucí TEXu ani být profesionálním prostředím pro vydavatele a nakladatele. Jeho hlavní cíl je pomoci šíření TEXu a být odrazovým můstkem pro další poznávání TEXu u běžných uživatelů.

Nevymýšlet kolo

Základním principem, který jsem při tvorbě tohoto projektu přijal, je snaha „Nevymýšlet kolo.“. Celý projekt sám o sobě se však může zdát právě porušením uvedeného principu. Upravil bych tedy uvedenou větu do tvaru „Nevymýšlet kolo tam, kde to není nezbytně nutné.“. No a tento projekt minimálně pro mou rodinu a mé studenty nutný je.

Vzhledem k uvedenému principu tedy nebudeme tvořit distribuci na zelené louce pomocí `web2c`, ale vyjdeme z nějaké existující distribuce, kterou ještě důkladně probereme. Původně uvažovaná varianta očesání TEXLive se ukázala jako silně nereálná, protože se jedná o příliš rozsáhlou distribuci. Použil jsem proto značně menší distribuci fpTEX, která je obdobná, ale lze ji nainstalovat do minimálního prostoru v řádech desítek MB. Úkolem nebylo vytvoření nejmenší distribuce, ale malé a výkonné distribuce. Proto jsem ještě ručně nainstalovaný fpTEX probral a zrušil všechny nepotřebné formáty, pomocné programy, varianty a styly, které mi přišly jako redundantní.

Zásadní otázkou bylo, jaký formát zvolit jako jediný správný (možnost volby je věc pěkná a demokratická, která však později komplikuje instalaci a život uživatele) a jaké cílové formáty zvolit. Jakožto přesvědčený LATEXista jsem v první volbě měl jasno a vzhledem k obtížím se zobrazováním formátu PostScript (či dokonce DVI) jsem zvolil verzi produkující přímo formát PDF. Tedy jedinou podporovanou verzí TEXu v projektu Ligatura je varianta PDFLATEX. Dalším zásadním rozhodnutím bylo orientovat se jen na pořádné verze MS Windows, tedy na verze 2000 nebo XP.

Dále jsem v distribuci nahradil veškeré babelizované styly za originální styly a fonty CSTEXu, protože podle tvrzení TEXových odborníků okolo mne je to ta jediná správná česká distribuce s pěknými fonty a českými vzory pro dělení. Po odstranění všech redundancí jsem tak dosáhl 46 MB distribuce, na které budu dále ještě pracovat.

Tuto distribuci bylo nutné dále obohatit o vhodný editor zajišťující vyvolávání překladu, zobrazovač výstupů v PDF a vhodnou kontrolu pravopisu.

Dobrý editor je základ

Při své práci nejraději používám editor `vim`, který mi vyhovuje jak na programování, tak na přípravu textů. Ale např. pro psaní pošty mám raději editor `joe`. Z tohoto příkladu je vidět, že určení textového editoru je věc velmi záludná, protože každý uživatel má jiné představy a zvyklosti. Přesto ve světě MS Windows existuje řada nepsaných dohod, jak by měl textový editor pracovat. Pro práci s TEXem se řadě mých studentů i mně samotnému velmi osvědčil editor TextPad. Bohužel není zdarma a proto jeho užití by bylo velmi problematické. Při dlouhém hledání na Internetu se mi nepodařilo najít podobný editor, který by byl šířený zdarma.

Důsledkem předchozího zklamání z dlouhého hledání jsem se rozhodl jedno kolo znovu vymyslet v podobě naprogramování si v prostředí Borland Delphi nového editoru pro Windows, který bude přímo určen pro práci v Ligatuře. Jedná se tak vlastně o jedinou část projektu, kterou jsem byl nucen naprogramovat a vytvořit podle svých osobních představ.

Vyšel jsem z editoru TextPad a snažil se jej maximálně napodobit způsobem editace (syntax highlighting, číslování řádků, zalamování, doplňující vyhledávací funkce s regulárními výrazy apod.). Díky využití komponenty SynEdit, kterou jsem nalezl na Internetu zdarma k užití, byla tvorba editoru spíše hrou než náročnou prací. S postupem času jsem editor dále obohatil o možností označování pomocných značek vedle řádků (např. obrázek „je zde chyba, ještě se sem vrátím“) a především o funkce pro vkládání celých bloků textů na stisk klávesy či jejich kombinace (makra). Tuto funkci jsem pojmenoval rychlé vsuvky.

Editor jsem propojil pomocí horkých kláves s překladačem `pdfcslatex` (volá se BAT soubor umístěný v cestě, jeho úkolem je nastavit proměnné prostředí pro TEX a provést překlad). Díky dávkovému souboru není problém nahradit malý fpTEX později např. TEXLive a používat nadále editor z Ligatury. Na editor jsou napojeny i další externí programy – kontrola pravopisu, ovlnkování textu a zobrazování přeloženého výsledku.

Zatím ve verzi 0.05 (aktuální verze) obsahuje editor ještě drobné chybky a nepodporuje editaci více souborů najednou, ale jeho funkce postačily k napsání mé rigorózní i disertační práce a mé ženě k vytvoření práce diplomové. Současně i tento článek je tvořen v Ligatuře. Říká se, že pokud dokáže nástroj pracovat sám nad sebou, je konečně otestován. V případě Ligatury to platí dvojnásob, protože je u mne jako autora v každodenním používání.

Editor pomocí nástrojů ToolBar2000 získal vzhled moderní MS Office aplikace, abychom usnadnili uživatelům jejich přechod na tento editor.

Zobrazování výsledků

Zobrazování výsledků v Adobe Acrobat Readeru nese s sebou dva základní problémy. Tím prvním je nutnost zajistit instalaci celého prohlížeče na cílovém počítači a druhým problém nemožnost generovat nové PDF, dokud je staré v Adobe Acrobat Readeru otevřené.

Uvedené skutečnosti mne vedly k analýze dalších možností jak zobrazovat PDF. Při tomto výzkumu jsem u Adobe objevil komponentu `pdf.ocx`, což je ActiveX varianta PDF prohlížeče (Adobe Acrobatu Readeru). Tato komponenta může být integrována do okna Ligatury (tzn. mimo jednoho ActiveX prvku není nutné nic nového instalovat) a okno je částečně pod naší kontrolou. Je možné např. realizovat přegenerování PDF a jeho znovuzobrazení.

Uvedené ActiveX má i několik nevýhod:

- pokud je otevřen plný Adobe Acrobat Reader a následně uzavřen, jsou zničena i ActiveX okna (zjevně chyba),
- ActiveX rozhraní nezná funkci pro zjištění aktuální stránky (nejsme tedy schopni si ji zapamatovat a při znovuotevření stejného dokumentu není možné přejít ihned na příslušnou stránku),
- práce s ActiveX je natolik komplikovaná, že ne vždy se komponenty chovají podle očekávání (to klade zvýšené nároky na programování editoru).

Dosavadní zkušenosti z používání ActiveX komponenty prohlížeče PDF ukazují, že se jedná o plnohodnotnou náhradu Adobe Acrobat Readeru.

Kontrola pravopisu

V tomto okamžiku není propojení na kontrolu pravopisu realizováno. Předpokládám využití softwarových produktů `ispell` nebo `aspell`, přičemž vstupní text bude zasílán do uvedeného programu a na základě reakcí tohoto programu budou zobrazeny variantní správná slova pro každý nalezený gramatický jev. Druhou možností by mělo být trvalé kontrolování periodickým spouštěním tohoto programu na pozadí a podtrhování špatných výrazů červenou vlnovkou tak, jak to dělá program MS Word.

Vzhledem k častému výskytu překlepů a gramatických chyb v mých dokumentech bude nutné se na tuto funkci zapojenou do editoru soustředit co nejdříve.

Další plán rozvoje

Uvedený projekt je teprve v začátcích. Byl započat v prvních jarních měsících tohoto kalendářního roku a v tomto okamžiku umožňuje (s několika známými chybami, které se zatím nedaří opravit) plně editovat, překládat a zobrazovat poměrně rozsáhlé texty (chemická diplomka apod.).

Pro nejbližší dobu předpokládám dokončení stávajících funkcí uvnitř Ligatury a doplnění kontroly pravopisu. V tomto okamžiku při použití vhodného instalačního programu bude možné vyrobit jednoduchý instalační balíček. Průběžně jsou betaverze distribuce testovány širokým okruhem mých známých, kteří s TEXem pracují nebo začínají pracovat.

Další směry rozvoje projektu musí být stanoveny podle potřeby v budoucnosti. Po dosažení určitého základního stavu by bylo vhodné informovat o tomto projektu českou TEXovou veřejnost pro zajištění podpory této distribuce i jinými uživateli. Mezi tyto budoucí potřeby patří také webové stránky produktu, podpůrnou diskuzní skupinu apod.

Určitě bych v budoucnu nechtěl, aby se z programu stala „další“ existující distribuce, kterou by bylo nutné udržovat, ale která by nepomáhala základnímu cíli šířit TEX mezi nové uživatele.

Další zajímavá možnost rozvoje spočívá ve vytvoření desítek standardních českých stylů pro dopis, fakturu, zprávu apod., které by umožnily i běžnému uživateli začít produkovat typograficky kvalitní předpřipavené dokumenty (podobným stylem nabízí tyto dokumenty/šablony např. i program MS Word).

Závěr

Projekt Ligatura je zaměřen na vytvoření malé české TEXové distribuce zaměřené na produkci PDFCSLATEXových výstupů. Vychází z velmi zjednodušené distribuce fpTEXu a je dále obohacena vlastním jednoúčelovým editorem (určeným jen pro TEXování) a zobrazovačem PDF výstupů. Do budoucna se počítá s rozšířením editoru o funkce kontroly pravopisu a také přípravou TEXových stylů a dokumentů pro začínající uživatele. Jedním z konečných cílů je být na CD jako součást knihy o TEXu pro začátečníky. Projekt v současné betaverzi můžete nalézt na webové adrese `http://ligatura.milansorm.cz/`. Program bude po dokončení první produkční verze šířen nadále jako open-source (ale alespoň začátek bych rád podržel v pevných rukou a tak je program v betaverzích šířen jako freeware).

Rád bych na závěr poděkoval všem uživatelům a zejména mé ženě za dosavadní používání tohoto projektu při každodenním TEXování, kterým pomáhají postupně odstraňovat chyby, které brání nasazení produktu u uživatelů – začátečníků.

Uvítám také veškeré zájemce o podporu tohoto projektu ať již přiloženou rukou k vývoji (editor, distribuce), radou (co s chybami `pdf.ocx`) či testováním. Další diskuze o projektu je možná na mém e-mailu `sorm@pef.mendelu.cz`.

Skenujeme v Linuxu programem VueScan

Zdeněk Wagner

Přednáška představuje program VueScan (shareware, od 7.6.71 je linuxová verze pro osobní potřeby zdarma). Jsou shrnuty základní funkce programu a jsou vysvětleny postupy práce s filmovými i stolními skenery včetně skenerů s dianástavcem. Je popsána i metoda kalibrace skeneru pomocí standardního terče a v závěru je zmíněn program SCARSE pro převod naskenovaných obrázků z barevného prostoru RGB do CMYK pro prepress. Přednáška též obsahuje stručné srovnání s jinými skenovacími programy.

Klíčová slova: Skenování, VueScan, SCARSE, RGB, CMYK

Úvod

Je známou skutečností, že obrázek často řekne více než tisíc slov. Mnoho textů, jak odborných, tak beletristických, je proto doprovázeno ilustracemi. Reklamní tvorba je na obrázcích různého charakteru přímo postavena. V odborných publikacích se vyskytují grafy, diagramy a schémata, která lze snadno vytvářet přímo v počítači. Nezastupitelnou úlohu však mají též kresby a fotografie pořízené klasickými filmovými přístroji i reprodukce výtvarných děl z předdigitální éry. Významnou součástí přípravy publikace jak tištěné, tak zveřejněné v elektronické podobě, proto stále zůstává skenování.

Komerční firmy se tradičně zaměřují na svět operačních systémů Windows a Macintosh. V jiných operačních systémech systémech máme často k dispozici jen programy, jejichž ovládání působí amatérským dojmem. Kromě toho neexistuje všeobecná norma pro komunikaci se skenerem. Nestačí, že máme ovladače pro komunikaci přes rozhraní SCSI, USB, či IEEE 1394 (FireWire), musíme ještě mít program, který se domluví s konkrétním typem skeneru. Zatímco se staršími a levnějšími modely si volně dostupné programy docela dobře rozumí, se skenery střední a vyšší třídy mají vážné problémy vedoucí někdy až k nutnosti stisku tlačítka RESET na počítači a vypnutí a zapnutí skeneru, aniž by se cokoliv naskenovalo. Použití profesionálních skenovacích programů je v takových případech nezbytností.

Dobrou zprávou je, že program takových kvalit je dostupný i pro Linux. Jeho masovějšímu nasazení bránilo to, že se jedná o shareware, přičemž neregistrovaná kopie naskenovaný obrázek znehodnotí. Linuxová verze 7.6.71 a vyšší je však pro osobní použití zdarma, čímž se uživatelům nabízejí nové možnosti.

V příspěvku si ukážeme, jak lze levnými prostředky dosáhnout výsledků srovnatelných s tím, co poskytují nepoměrně dražší programy.

Nekupujme zajíce v pytli

Pogram VueScan, jehož autorem je Ed Hamrick, si lze volně stáhnout z webu [1], a to ve verzích pro Linux, Windows a Macintosh. Uživatel tedy není nucen zaplatit za program známý jen z reklamních letáků. Místo toho přistupuje na filozofii sdílenou sharewarovými programy: „Vyzkoušej, a jsi-li spokojen, zaplať.“ Řadu sharewarových programů smí uživatel zkoušet nejvýše jeden měsíc. To však není případ VueScanu. Uživatel smí zcela legálně testovat libovolně mnoho verzí libovolnou dobu. Můžeme si též předem ověřit, že VueScan pracuje s naším skenerem. Ne vždy je totiž skener dodáván současně s programem. Pokud jsme koupili program bez vyzkoušení a dodatečně zjistíme, že našim požadavkům nevyhovuje, nemáme jinou možnost než se smířit s odepsanou investicí. Za VueScan však platíme až v okamžiku, kdy jsme s programem spokojeni.

Skenování bez skeneru

Každý fotograf, který používá klasický filmový přístroj a zabývá se vážněji barevnou fotografií, z vlastní zkušenosti ví, jak obtížné je zajistit správné barevné podání na fotografiích zhotovených v minilabu. Při použití kreativních filtrů, například Cokin nebo Lee Filters, je to téměř nemožné. Pouze fotograf totiž ví, jakého barevného efektu chtěl dosáhnout. Automat v minilabu se snaží zprůměrovat barvy do neutrální šedi, čímž potlačí zamýšlený efekt. Ani dodatečné naskenování hotové fotografie a její barevná úprava problém neřeší, protože informaci, která byla ztracena, již nelze vrátit zpět. Při zpracování tedy fotograf musí být fyzicky přítomen. Zhotovení barevných zkoušek fotochemickou metodou je i při současné automatizaci časově náročné a drahé. Tisk na domácí inkoustové tiskárně se již kvalitou i cenou vyrovná klasické fotografii. A zde může VueScan pomoci, i když nemáme skener. Můžeme si totiž film nechat naskenovat u někoho jiného a uložit v surovém formátu, tedy bez barevných úprav. Pak si sami naskenujeme snímky ze souboru a provedeme úpravy podle svých požadavků. Přitom nevadí, pokud snímky v surovém formátu byly vytvořeny na jiném operačním systému, jen je nutné mít stejnou nebo vyšší verzi programu VueScan. Nejsme-li ani potom s výsledkem spokojeni, můžeme si stěžovat pouze na vlastní nešikovnost.

Jak pracuje VueScan

Skenovací programy mohou pracovat ve dvou režimech. Prvním režimem je spouštění jako plugin v nějakém jiném programu (nemusí to být vždy grafická

aplikace, ve Windows lze skenovat i z Wordu a Excelu). Pro tento účel existují dvě rozhraní: TWAIN a SANE. Integrace do grafického programu se zdá výhodná, ale nevýhodou je, že v době skenování, které může trvat poměrně dlouho, nelze nic jiného v grafické aplikaci dělat. Nejsem si zcela jist, zda při skenování pomocí xscanimage nebo xsane lze v Gimpu provádět jinou činnost. Gimp však přes nesporné kvality není stoprocentně stabilní (což ale není žádný program). Pokud pracujeme s obrázkem s několika vrstvami v tiskovém rozlišení, jehož velikost je nezřídka 80 MiB, Gimp občas spadne. Stane li se to při skenování, může to mít nepříjemné následky.

Skenovací program však může běžet jako samostatná aplikace. Zejména v Linuxu můžeme mít na jedné ploše skenující VueScan a na druhé ploše upravovat v Gimpu obrázek, jenž byl již dříve naskenován. Zapneme-li si ve VueScanu funkci „beep when done“, nemusíme neustále mezi plochami přepínat a VueScan sám upozorní, že svoji práci ukončil a čeká na další úkol.

Hardwarové požadavky

VueScan je sice vysoce náročný na paměť, ale pracuje i na počítaču s malou fyzickou kapacitou RAM. Běží i na počítači vybaveném pouze 32 MiB, ale obvykle je nutno ke swapovacímu oddílu (není-li dostatečně dimenzován) připojit ještě swapovací soubor. To však lze zajistit bez nutnosti restartu systému. Pokud chce uživatel vážně pracovat s grafickými aplikacemi, měl by svůj počítač vybavit výrazně lépe, nicméně v případě nouze lze VueScan úspěšně provozovat i na zastaralém stroji, kde se jiné systémy stěží rozběhnou a profesionální komerční skenovací programy nelze ani nainstalovat.

VueScan podporuje většinu moderních skenerů. Nelze jej použít s některými starými modely, které již nejsou na trhu, a s levnými přístroji komunikujícími přes paralelní rozhraní. Jedná se většinou o hybridní zařízení skener/kopírka/tiskárna nebo skener/kopírka/fax, k nimž chybí potřebná technická dokumentace. VueScan umí též zpracovat soubory v surovém formátu z mnoha typů digitálních fotografických přístrojů. Později si vysvětlíme, proč je taková funkce důležitá.

Instalace

Instalace je velmi jednoduchá. Stačí rozbalit program s dokumentací do libovolného adresáře. Jisté triky může vyžadovat jen oživení skenerů.

Starší linuxová jádra vyžadovala, aby identifikační číslo skeneru připojeného k USB a identifikační číslo výrobce bylo předáno jako parametry modulu `scanner.o` a nastaveno v proměnných SCANPID0 a SCANVID0. Tyto údaje, pokud je neznáme z dokumentace, najdeme v souboru `/proc/bus/usb/devices`, máme-li ovšem v jádře `usbdevfs`. Pro vlastní skenování `usbdevfs` nepotřebujeme. V novějších jádrech již tyto údaje nemusíme zadávat, o vše se postará

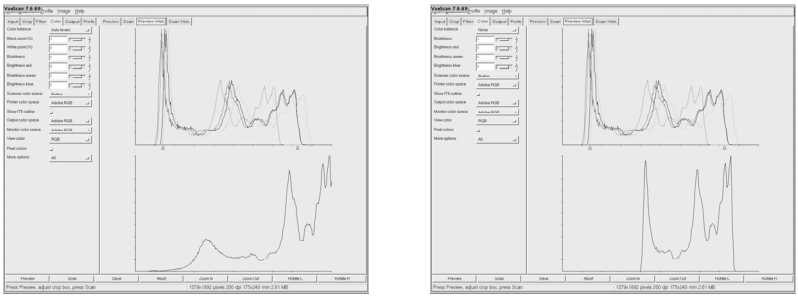
hotplug. Skener připojený na SCSI obvykle nepředstavuje žádný problém. Potřebujeme pouze přístup pro čtení i zápis na fyzické zařízení. V Redhatu 7.2 se o vše postaral Kudzu, v Debianu 3.0r2 bylo nutno přístupová práva nastavit ručně, jinak směl skenovat pouze root. Právo čtení i zápisu je nutno nastavit na všechna `/dev/sg*`, aby směl skenovat libovolný uživatel. Skener však musí být zapnut a inicializován dříve než počítač, aby jej identifikoval SCSI BIOS. V opačném případě nebude viditelný ani v `/proc/scsi/scsi` a skenovací program jej nenajde.

Podrobný návod najdete v dokumentaci programu VueScan.

Potřebujeme větší barevnou hloubku?

Gimp umí pracovat s obrázky v barevném prostoru RGB, přičemž pro každý barevný kanál je vyhrazeno 8 bitů. Jsme tedy schopni zaznamenat $256^3 = 16777216$ barevných odstínů. Je to málo, nebo moc? Abychom to mohli posoudit vyjádříme si barvu v barevném prostoru HSV (Hue-Saturation-Value). Složka H reprezentuje barevný tón, S je sytost barvy a V její jas. Budeme předpokládat, že hodnoty všech složek mohou být racionální čísla. Jsme tedy schopni v modelu RGB vyjádřit 1536 různých barevných tónů, ale empiricky bylo zjištěno, že lidské oko jich rozliší pouze 180 (viz např. [2]). Okem rozlišíme změny sytosti a jasu v jednotkách procent, ale krok v osmibitovém RGB je menší než půl procenta. Zdálo by se tedy, že máme obrovský nadbytek informací. Bohužel je tato domněnka mylná. Ukážeme si to na příkladu. Skenerem, který vnitřně pracuje s šestnáctibitovou barevnou hloubkou, si naskenujeme černobílou fotografii včetně jejího bílého okraje. V prvním případě použijeme optimální nastavení barevného (nyní spíše tonálního) vyvážení, jemuž pomůžeme pouze vhodnou volbou bílého a černého bodu. V druhém případě žádné korekce ve skeneru nepoužijeme a výsledek uložíme v osmibitové hloubce. Nastavení v programu VueScan je ukázáno na obrázku 1. V horní polovině pravého panelu je histogram tak, jak byl při skenování sejmut, v dolní polovině je histogram výsledného obrázku při daném nastavení parametrů barevného vyvážení (ovšem v jiných jednotkách, než používá Gimp). Všimněte si, že se histogram značně liší.

Výsledné fotografie vidíte na obrázku 2 společně s histogramy získanými funkcí Image/Colors/Levels. Na první pohled je zřejmé, že střední fotografie obsahuje pouze střední úrovně šedi, zcela chybí bílá i černá. Abychom dodatečnou úpravou dosáhli věrného podání fotografie, bylo nutno šedou stupnici roztáhnout tlačítkem Auto a ještě obraz zesvětlit zvětšením hodnoty gamma. Při zběžném prohlížení vypadá obrázek dobře, ale histogram napovídá, že ve stínech nemusí být vše v pořádku. Skutečně je to vidět na výřezu na obrázku 3. Ve střeše levé věže i na levé stěně pravé věže je dobře patrná posterizace (tj. ostře ohraničené přechody mezi různými odstíny).



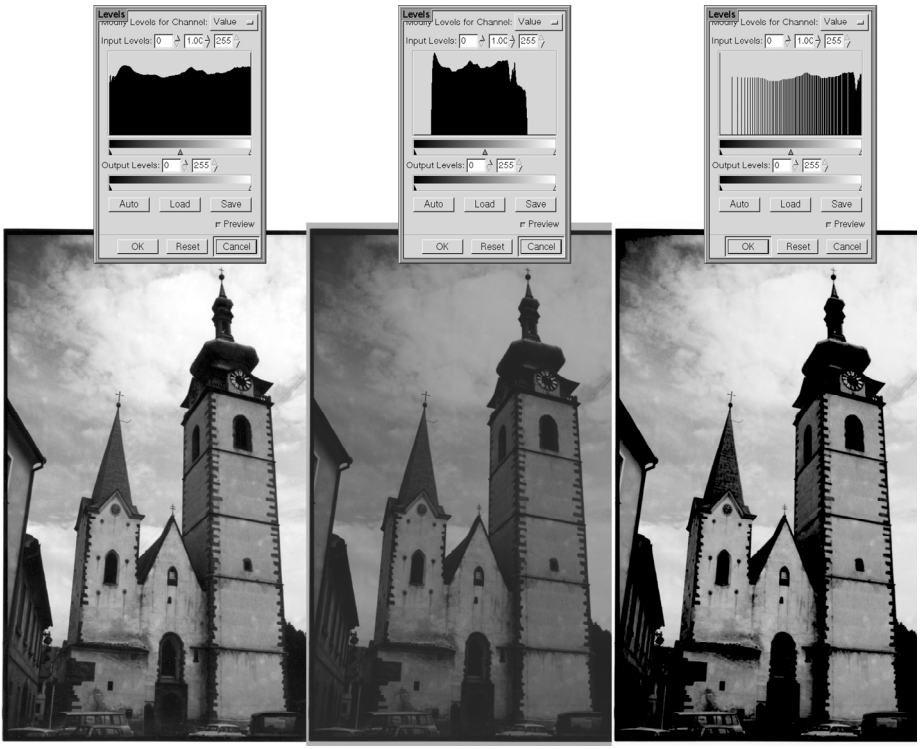
Obrázek 1. Nastavení parametrů barevného vyvážení

Zcela obecně lze konstatovat, že pro zobrazení na obrazovce i tisk je osmibitová barevná hloubka naprosto postačující. Můžeme si bez obav dovolit i malé změny tonality, větší zásahy si dovolit nemůžeme. Slovo *větší* nelze kvantifikovat. Mez přípustnosti je u každého obrázku jiná. Barevné obrázky jsou k posterizaci obecně náchylnější než černobílé. Rozsah odstínů barevných předmětů je výrazně nižší. Proto při barevných úpravách nezřídka vznikají viditelné mapy. Úpravy tedy musíme dělat dříve, než omezíme barevnou hloubku na 8 bitů na kanál.

Volba rozlišení

Technické možnosti se neustále zlepšují. Před 10 lety jsme byli rádi, když poměrně drahý skener zvládal rozlišení 300 dpi. Nyní i levné skenery nabízejí optické rozlišení 1200 dpi. Zamysleme se však nad tím, k čemu se takové rozlišení hodí.

V testech, které se občas objevují v časopisech, lze zjistit, že charakteristická rozlišovací schopnost běžných kinofilmů je 100 čar na milimetr. Z definice rozlišovací schopnosti je zřejmé, že se jedná o 100 černých čar, mezi nimiž je 100 stejně širokých bílých čar. Na reprezentaci v počítači proto potřebujeme 200 pixelů na milimetr. Při klasickém zpracování se počítá s maximálně desetinásobným zvětšením, tedy výsledné rozlišení bude 200 pixelů na centimetr, což je přibližně 500 dpi. To však platí pouze v případě, že má dostatečné rozlišení použitý objektiv. Kromě toho je fotografický papír určen k pozorování prostým okem. Tak malé detaily oko nerozliší, proto je skutečné rozlišení klasické fotografie podstatně menší než hodnota odhadnutá uvedeným teoretickým výpočtem. Nemá smysl skenovat barevnou fotografii s větším rozlišením než 300 dpi a černobílou fotografii na více než 600 dpi, není-li ovšem cílem naší snahy studium tvaru zrn. Stejné rozlišení se používá i při ofsetovém tisku. Předlohu, která má vyšší rozlišení, tedy s největší pravděpodobností nenajdete.



Obrázek 2. Fotografie skenovaná s odlišným nastavením; vlevo s optimálními parametry zjištěnými při skenování, uprostřed bez korekcí, vpravo fotografie naskenovaná bez korekcí s dodatečně upravenou tonalitou. U každého obrázku je připojen histogram.

Obrázek 3. Výřez z obrázku 2, vlevo z verze skenované se správným barevným vyvážením, vpravo z verze skenované bez korekcí, s následnou úpravou tonality v Gimpu

Existuje však situace, kdy má vyšší rozlišení smysl. Ofsetový tisk vytváří polotóny pomocí rastru. Pokud naskenujeme takový obrázek, vznikne téměř vždy moiré a často dojde i k barevnému posunu. S odstraněním rastru volbou *descreening* přímo při skenování nemám dobré zkušenosti. Pokud obrázek naskenujeme v mnohem vyšším rozlišení, v Gimpu jej gaussovsky rozmázneme, převzorkujeme na tiskové rozlišení, zaostříme použitím neostré masky a na závěr vyretušujeme, dosáhneme většinou lepších výsledků. Nezapomeňte však, že stránka formátu A5 naskenovaná v rozlišení 1600 dpi zabere více než 250 MiB, na stránku A4 potřebujete více než 1 GiB. Nemáte-li dostatek paměti, VueScan při skenování spadne.

Zatím jsme se věnovali pouze skenování neprůhledných předloh. K některým stolním skenerům však lze přikoupit nástavec na skenování průhledných předloh (často se mu říká dianástavec). Mnoho uživatelů má pak pokušení skenovat kinofilmy na stolním skeneru s dianástavcem.

Na rozdíl od fotografických papírů jsou filmy určeny k dalšímu zpracování, mají tedy podstatně vyšší rozlišení. Můžeme tedy bez obav využít plné rozlišení skeneru. Rozlišení stolních skenerů je pro skenování filmů malé, skenery střední (poloprofesionální) třídy mívají rozlišení 1600 dpi. Filmové skenery nižší třídy mají rozlišení 2900 dpi. Amatérský fotograf by se mohl rozhodnout, že se spokojí s pohlednicovým formátem, na což je 1600 dpi postačující. Stolní skenery však mají jednu velkou nevýhodu: jsou zaostřeny na pevnou rovinu. Film nemusí na skleněnou podložku skeneru dosednout přesně, takže naskenovaný obraz bude neostrý. Filmové skenery dovedou při skenování zaostřit na citlivou vrstvu. Stolním skenerům s dianástavcem tedy svěřme především skenování ofsetových filmů, k nimž se ztratily předlohy. Na skenování kinofilmů a filmů středního formátu se lépe hodí filmové skenery.

Skenujeme filmy

Z důvodů spíše historických přežívá domněnka, že diapozitiv je výhodnější než negativ. Má sice vyšší densitu, ale i negativ je schopen zaznamenat větší rozsah jasů, než jsme schopni vytisknout. Vyšší densita může dokonce vést k problémům při skenování, kdy dojde ke ztrátě kresby ve stínech. navíc je inverzní proces náročnější a vede většinou k větší zrnitosti. Pokud se nedodrží přesná doba prvního vyvolání, dojde k nenapravitelnému rozladění barev. Je pravdou, že pouhým okem na negativu nevidíme pravé barvy, ale při současném stavu techniky dostaneme z negativu dobrý náhled stejně rychle jako z diapozitivu. Není tedy racionální důvod k tomu, abychom neskenovali negativy.

Když naskenujeme negativ, musíme provést barevnou inverzi. Gimp má k tomuto účelu funkci, ale jak rychle zjistíme, pro inverzi naskenovaného negativu není použitelná. Barevný negativní film má totiž oranžovou masku, o jejímž účelu se dočteme v již zmíněné knize [2]. Od naskenovaného obrazu musíme

barvu masky nejprve odečíst. Není snad nutno připomínat, že odečtení masky musíme provádět ve větší barevné hloubce.

Barva masky se však u každého typu filmu liší a může být jiná i u různých kusů téže značky. VueScan však nabízí řešení: kalibraci pro konkrétní roli filmu. Detaily ovládacího panelu najdete na obrázku 4. Ke kalibraci potřebujeme neexponovaný začátek filmu. Naskenujeme jeho náhled a zvolíme myší oblast, podle níž chceme změřit expozici (nejlépe celé políčko filmu). Pak zapneme Lock exposure. Na obrázku je stisknuto tlačítko Long exposure pass, avšak to se používá pouze pro diapozitivy s vysokou densitou. Doba skenování se pak prodlouží na dvojnásobek.

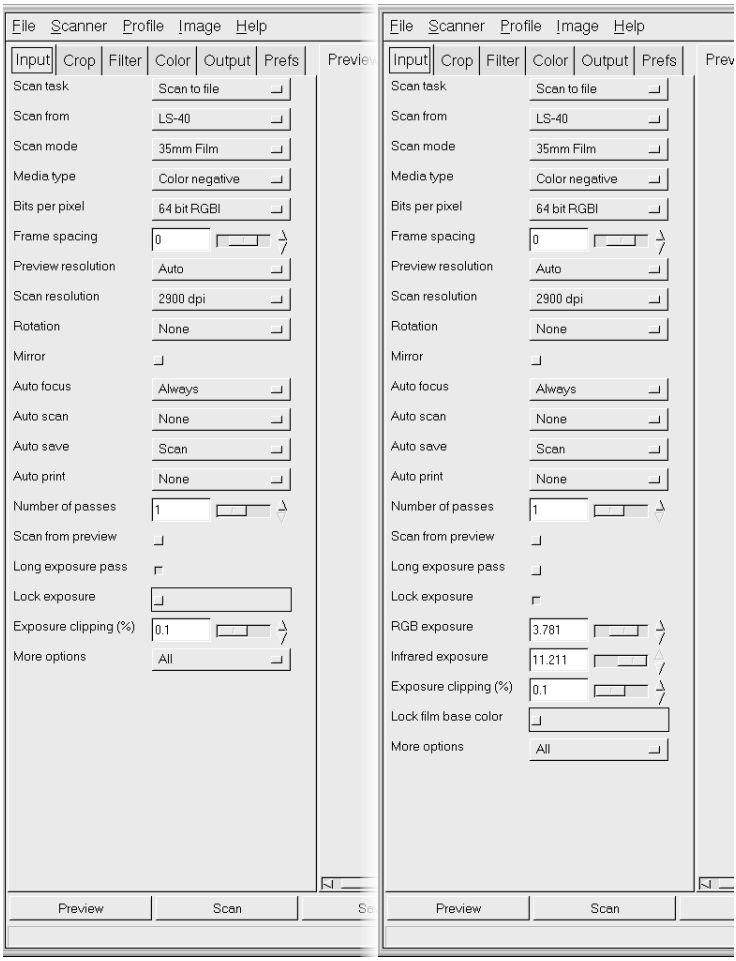
Jakmile zamkneme expozici, objeví se naměřené hodnoty, jak je vidíme v pravé části obrázku. Infračervenou expozici uvidíme pouze v případě, že to skener umí. Expozice v infračervené oblasti se totiž používá k automatickému odstranění nečistot a škrábanců. Se zamknutou expozicí znovu naskenujeme náhled. Až pak se objeví tlačítko Lock film base color. Po jeho stisknutí si VueScan zapamatuje barvu masky, která je pro celou roli filmu stejná. Parametry si při ukončení programu uloží do konfiguračního souboru, aby je při dalším spuštění mohl použít. Lepší však je, když si je uložíme sami funkcí File/Save options do souboru ve vlastním adresáři. Pak můžeme skenování role filmu v polovině přerušit, skenovat něco jiného, a pak si funkcí File/Load options všechny parametry načíst. Nemusíme tedy znovu měřit barvu masky, nastavovat jméno adresáře a souborů a další parametry.

Práce s barvou

V této kapitole se budeme věnovat základním principům práce s barvou. Podrobnější výklad najdete v seriálu Grafika v Linuxu [3].

Již jsme se zmínili o barevném prostoru RGB. Problém je však v tom, že není RGB jako RGB. Jistě víte, že monitory pro počítače kompatibilní s IBM PC používají jinou hodnotu gamma než Macintosh. Aby bylo umožněno sdílení obrázků mezi různými monitory, byl vytvořen na systému nezávislý barevný prostor sRGB. Pro tisk je ale vhodnější Adobe RGB nebo Wide Gamut RGB. Každý skener má svůj vlastní prostor RGB, který odpovídá spektrální citlivosti snímacího čidla. Abychom převedli naskenovaný obrázek z barevného prostoru skeneru do nějakého standardního prostoru, musíme skener zkalibrovat. Provedeme to tak, že naskenujeme kalibrační terč IT8 a výsledek předložíme programu, který jej srovná s hodnotami získanými přesným spektrometrem a z nich vypočte barevný profil ICC. Pokud máme profesionální verzi programu VueScan, máme takovou funkci automaticky k dispozici.

Fotorealistické inkoustové tiskárny pro domácí použití obvykle vyžadují obrázek v prostoru RGB. Ovladač, bohužel jen ve verzích pro Windows a Macintosh, umožňuje zadání barevného profilu získaného kalibrací tiskárny. Tiskárnu také



Obrázek 4. Postup kalibrace pro skenování barevného negativního filmu

umí VueScan zkalibrovat. Nejprve vytiskneme kalibrační terč. Ten pak naskenujeme kalibrovaným skenerem a vypočteme profil tiskárny. Vytisknout kalibrační terč umí i neregistrovaný VueScan bez sériového čísla. Lze jej tedy vytisknout na počítači s Windows, profil tiskárny vytvořit na Linuxu a nainstalovat jej do ovladače tiskárny ve Windows.

Ofsetový tisk ovšem vyžaduje obrázky v barevném prostoru CMYK. Ze základní školy známe poučku o doplňkových barvách. Nabízí se tedy použití rovnice $\mathrm{CMY} = 1 - \mathrm{RGB}$. Tato rovnice je však pouhou matematickou abstrakcí

platnou pro ideální syté barvy s lineární odezvou. V praxi je zcela nepoužitelná. Konverzi musíme provést pomocí barevných profilů, které lze získat např. z programu Photoshop LE, jenž je často dodáván k inkoustovým tiskárnám. K vlastní konverzi můžeme použít volně dostupný program SCARSE [4].

Srovnání s jinými programy

VueScan má snad jedinou nevýhodu: neumí pracovat s levnými hybridními skenery kombinovanými s tiskárnami či faxy. Ty mají obvykle software pouze pro Windows, využívá se v něm rozhraní TWAIN. Uživatel pak může přímo skenovat z libovolné aplikace. Ovládání takových programů však ve mně budí dojem, že se jejich autoři vydováděli na spojení všeho se vším, jen ten skenovací software tam zapomněli přibalit. Ovládání skenovacích funkcí je tam ořezáno na minimum a dostat z takového zařízení obrázek v rozumné kvalitě je téměř nemožné.

VueScan nepodporuje TWAIN ani SANE, nelze tedy skenovat přímo do grafické aplikace. V Linuxu však můžeme na jedné ploše skenovat a na druhé ploše upravovat již naskenované obrázky, čímž lze dosáhnout významné časové úspory.

Komerční programy nabízejí dávkové skenování. VueScan má takovou funkci též. Problém je však v tom, že jen málo předloh se pro dávkové skenování hodí. Je to výhodné spíš v situaci, kdy musíme předlohy rychle vrátit. pak lze spustit dávkové zpracování, obrázky ukládat v surovém formátu a zpracovat je dodatečně.

V reklamních letácích a stručných informacích na WWW stránkách se dočtete, že profesionální programy skenují v prostoru CMYK. To VueScan neumí. Je to ale pravda pouze částečná. Program samozřejmě skenuje v prostoru RGB. Když ale v příslušném dialogu navolíte zdrojový profil RGB a cílový profil CMYK, program provede konverzi naprosto stejným algoritmem, který používá SCARSE. Zatímco běžný uživatel Windows bude klikat obrázek za obrázkem, běžný uživatel Linuxu napíše několik magických slov na příkazový řádek a v době, kdy počítač pracuje sám za něj, odejde na příjemnou večeři s přítelkyní.

Závěr

Program VueScan poskytuje v dostatečné kvalitě vše, co uživatel potřebuje. Ve spojení s programy Gimp, SCARSE, tiff2ps [5] a případně CinePaint [6] jej lze plně nasadit v profesionální praxi. Až na zcela výjimečné případy se vyrovná komerčním programům. Uživatelům, kteří chtějí vážně pracovat s grafikou, lze doporučit zakoupení profesionální verze, která zahrnuje celoživotní licenci na všechny budoucí aktualizace. Přestože jsou mnohé skenery dodávány i se softwarem pro Windows a Macintosh, vlastní komerční operační systém, který bychom si museli koupit, je dražší než celoživotní licence programu VueScan.

Odkazy

1. VueScan, `http://www.hamrick.com/`
2. Šmok J., Pecák J., Tausk P.: Barevná fotografie. Druhé, upravené vydání. SNTL, Praha 1978.
3. Brabec S.: Grafika v Linuxu (seriál), `http://root.cz`
4. SCARSE, `http://www.scarse.org/`
5. LibTIFF – TIFF Library and Utilities, `http://www.libtiff.org`
6. CinePaint, `http://cinepaint.sourceforge.net`

XML versus TEX, výhody a nevýhody

Zdeněk Wagner

Přednáška volně navazuje na přednášky o XML z minulého SLT. Srovnává možnosti obou systémů. Vysvětluje, co poskytuje zejména TEXistům XML. Zamýšlí se nad tím, kdy je vhodné přímé psaní textů v TEXu. Uvádí možnosti, jak z obou tupů zdrojových textů generovat soubory v jiných formátech a s jakými výsledky. Je též popsána možnost spojení TEXu i XML s databázemi.

Klíčová slova: XML, TEX, LATEX, XSLT

Motto

"Well, in *our* country," said Alice, still panting a little, "you'd generally get to somewhere else—if you ran very fast for a long time as we've been doing."

"A slow sort of country!" said the Queen. "Now, *here,* you see, it takes all the running *you* can do, to keep in the same place. If you want to get somewhere else, you must run at least twice as fast as that."

Lewis Carroll [1]

Úvod

Tištěná kniha se začíná šířit v polovině 15. století po vynálezu knihtisku. V průběhu staletí došlo k mnoha vynálezům a zlepšením, jež zlevnily a zkvalitnily knižní produkci. Ve své formě však až do osmdesátých let minulého století kniha zůstává stále knihou tištěnou na papíře.

V závěru 20. století dochází k bouřlivému rozvoji výpočetní techniky i počítačových sítí, jak lokálních, tak internetu. Roste tlak na urychlení výměny informací. Na světlo světa proto vstupují nové formy publikací, které již nejsou

spojeny s papírovým nosičem informací. Kniha v klasické podobě však ještě nevymírá. Naopak, často je požadováno, aby dokument byl publikován současně v podobě papírové i elektronické.

Stejně jako původní Gutenbergovy dřevěné typy byly vytlačeny typy olověnými a přibližně o pět století později ruční sazbu nahradila sazba počítačová, lze předpokládat, že staré softwarové nástroje vyklidí prostor nástrojům novým, nebo se alespoň přizpůsobí vznikajícím potřebám.

TEX patří nesporně k nejlepším sázecím programům. V souvislosti s rozvojem nových publikačních forem však vyplouvají jeho slabé stránky. Současně se objevují nové nástroje a datové formáty. Jedním z těch, jež si snaží získat prostor, je XML.

Tento příspěvek je zaměřen zejména na XML. Nebude se však vůbec zabývat sazbou dokumentů XML, protože této problematice se věnovaly články ve Zpravodajích Československého sdružení uživatelů TEXu 3–4/2002 a 1/2003. Nebudou zde ani probírány konkrétní nástroje. Článek je autorovým zamyšlením nad tím, co XML přináší zejména TEXistům. Jeho cílem je vzbudit v čtenářích zájem k jejich vlastnímu zamyšlení nad tím, jaké postupy budou v budoucnu používat při zpracování svých dokumentů.

Boření mýtů

Při srovnávání se často argumentuje tím, že na rozdíl od TEXu, jehož značkování je převážně prezentační, XML definuje logickou strukturu dokumentu. Vezměme si však jako příklad libovolný dokument XHTML. Kořenovým elementem dokumentu je `<html>`, v něm se vyskytuje hlavička `<head>` následovaná tělem dokumentu `<body>`. To jsou v zásadě jediné strukturní elementy XHTML verze 1. Zbytek značkování je výhradně prezentační.

Základní struktura LATEXových dokumentů je definována logickými značkami `\title`, `\chapter`, `\section` apod. Pak už záleží pouze na autorovi, zda ve zbytku dokumentu použije značkování logické, nebo prezentační. Pro plain TEX platí v zásadě totéž.

Přísně logické značkování popsal autor ve svém starším článku [2], kde se zabýval spojením databáze s LATEXem. Výstup z databáze měl následující formát:

```
\def\cislo{0001}\def\jmeno{Gorin}\def\inic{A. V.}
\def\tit{Prof.}\def\zeme{Russia}\def\zemecesky{RU}
\def\zemenvel{RUSKO}\def\tel{}\def\fax{}\def\email{}
\def\adri{...}\def\adrii{...}\def\adriii{...}\def\adriv{}\def\adrv{}...\exec
```

Definice makra `\exec` plnila stejný účel jako transformační styl XSLT, úlohu formátovacích objektů převzala makra definovaná v příslušném LATEXovém balíčku.

Při srovnávání tedy nelze používat zevšeobecněná tvrzení. Jak jsme si ukázali, volba mezi logickým a prezentačním značkováním není striktně vnucena tím, zda používáme XML či TEX, ale je do značné míry určena autorem dokumentu nebo autorem specifikace datového formátu, který je na jedné z výše uvedených technologií založen.

Formální správnost a úplnost dokumentu

Klikací sázecí programy obvykle nepovolí vytvoření neplatného dokumentu. Jiná je však situace v systémech, kde se do zdrojového textu zapisují logické či prezentační značky, ať už je to TEX, troff nebo XML. Zde máme možnost vytvořit nepřeberné množství syntaktických chyb, jež se neprojeví při psaní dokumentu, ale až při jeho zpracování. Pokud jsme ručně vytvořili jednoduchý dokument, celkem snadno z chybové zprávy pochopíme, co jsme provedli špatně. U složitějších dokumentů to již může být náročnější. Noční můrou pak bývá hledání chybějící pravé závorky v několikasetstránkové knize, když některé makro bylo definováno jako `\long`. Není-li kniha rozumně členěna do malých, samostatně načítaných souborů, TEXová diagnostika nám příliš nepomůže.

Soubor však může být formálně chybný, přestože LATEX při zpracování žádnou chybu nehlásí. Typickým příkladem je vynechání makra `\chapter` v třídě REPORT. Autor se pak diví, proč má `\section` číslo 0.1.

Kontrola správnosti a úplnosti je velmi důležitá i v případě, kdy si soubor pro další zpracování generujeme jako výstup z databáze. Pokud při zpracování dojde k chybě, potřebujeme jednoznačně určit, zda jsme si vygenerovali chybný soubor, nebo zda je chyba v makrech pro tisk. Můžeme si samozřejmě naprogramovat vlastní validátor v TEXu, ale XML má pro tyto účely hotové nástroje.

Předvedeme si to na konkrétním příkladu, kdy jako validační schéma využijeme Relax NG. Uvedeme jen část souboru.

Data exportovaná z databáze mají obsahovat údaje o přednáškách a vývěskách na jisté konferenci. Program konference zahrnuje sekci plenárních přednášek, několik sekcí přednášek a několik sekcí vývěsek. Část specifikace, uložená v souboru `export-base.rng`, vypadá takto:

```
<?xml version='1.0'?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:a="http://hroch486.icpf.cas.cz/rng-comments"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<start>
  <ref name='program.content'/>
</start>

<define name='program.content'>
  <interleave>
    <zeroOrMore>
```

```
      <ref name='def.plenary.lectures'/>
    </zeroOrMore>
    <zeroOrMore>
      <ref name='def.lectures'/>
    </zeroOrMore>
    <zeroOrMore>
      <ref name='def.posters'/>
    </zeroOrMore>
  </interleave>
</define>

<define name='def.time.info'>
  <a:comment>
    Define beginning time of a lecture as a combination of hour and minute.
    These attributes are not required in the preliminary program and
    must be ignored if present.
    Information comes from database, thus it is not necessary to validate
    the values.
  </a:comment>
  <attribute name='hour'>
    <data type='nonNegativeInteger'>
      <param name='minInclusive'>8</param>
      <param name='maxInclusive'>18</param>
    </data>
  </attribute>
  <attribute name='minute'>
    <data type='nonNegativeInteger'>
      <param name='minInclusive'>59</param>
    </data>
  </attribute>
</define>

<define name='choose.time.info'>
  <a:comment>
    To be redefined in the parent grammar, see def.time.info
  </a:comment>
  <notAllowed />
</define>

<define name='def.lecture.attrib'>
  <a:comment>
    Attributes for lectures and plenary lectures.
  </a:comment>
  <ref name='def.serial.num'/>
  <ref name='choose.time.info'/>
</define>

<define name='def.plenary.lectures'>
  <a:comment>
    Section with plenary lectures
```

```
  </a:comment>
  <element name='plenary-lectures'>
    <ref name='def.halfDayId'/>
    <oneOrMore>
      <a:comment>
        The section contains one or more plenary lectures.
      </a:comment>
      <element name='plenary-lecture'>
        <ref name='def.lecture.attrib'/>
      </element>
    </oneOrMore>
  </element>
</define>

...
</grammar>
```

Ze specifikace byly úmyslně pro stručnost vyhozeny definice mnoha atributů a vše si ukazujeme jenom na plenárních přednáškách. Důležitá je v této ukázce definice vzoru `choose.time.info`. Je v ní specifikováno, že použití této konstrukce není povoleno. Tento vzor se však používá v `def.lecture.attrib`. Funguje to jen proto. že validaci provádíme pomocí souboru `export.rng`, v němž je výše zmíněný soubor načten a vzor `choose.time.info` je předefinován:

```
<?xml version='1.0'?>
<grammar xmlns="http://relaxng.org/ns/structure/1.0"
         xmlns:a="http://hroch486.icpf.cas.cz/rng-comments"
         datatypeLibrary="http://www.w3.org/2001/XMLSchema-datatypes">

<start>
  <element name='chisa-program' xmlns="http://relaxng.org/ns/structure/1.0">
    <a:comment>
      preliminary - for preliminary program, no numbers, no time information
      final - for final program, time information required
    </a:comment>
    <choice>
      <group>
        <attribute name='type'>
          <value>preliminary</value>
        </attribute>
        <grammar>
          <include href='export-base.rng'>
            <define name='choose.time.info'>
              <a:comment>Time info is optional.</a:comment>
              <optional>
                <ref name='def.time.info'/>
              </optional>
            </define>
          </include>
        </grammar>
```

```
      </group>
      <group>
        <attribute name='type'>
          <value>final</value>
        </attribute>
        <grammar>
          <include href='export-base.rng'>
            <define name='choose.time.info'>
              <a:comment>Time info is required.</a:comment>
              <ref name='def.time.info'/>
            </define>
          </include>
        </grammar>
      </group>
    </choice>
  </element>
</start>

</grammar>
```

Všimněte si, že vzor `choose.time.info` obsahuje pouze odkaz na vzor `def.time.info`. Rozdíl je v tom, že v předběžném programu jsou časové informace nepovinné. Proto je příslušný vzor uzavřen v elementu `<optional>`. Podařilo se nám tedy mírně komplikovaná pravidla zapsat poměrně snadno. Kdybychom totéž chtěli řešit pouze TEXovými nástroji, vyžadovalo by to jistou dávku programátorského úsilí.

TEXisté mohou namítnout, že by to přece jen šlo naprogramovat snadno. V tomto konkrétním případě však byla situace komplikovaná tím, že soubor, vytvořený jedním člověkem, byl zpracováván jiným člověkem. Validační schéma tedy sloužilo mimo jiné i k tomu, aby v případě potřeby rozhodlo, kdo udělal chybu. Pokud by vše bylo řešeno výhradně na úrovni TEXu, zbývala by stále možnost, že chybu udělal programátor validačního TEXového makra.

Hledání specifických informací

Chceme-li najít určitou informaci, nemusí být fulltextové vyhledávání optimální metodou. Občas nás zajímá slovo či slovní spojení vyskytující se ve struktuře určitého typu. Je tedy nutné, aby zdrojový soubor měl dobré logické značkování. Kromě toho potřebujeme odpovídající prohlédávací nástroje. Takové možnosti nám TEX nenabízí, museli bychom si je naprogramovat. Ve světě XML však takové nástroje máme.

Podívejme se na knihu XML pro každého [3]. Autor chtěl z textu vytáhnout seznam odkazů pro zveřejnění na webu [4]. Dosáhl toho jednoduchým transformačním stylem [5]. Podívejme se na jednu šablonu:

```
<xsl:template match="chapter/section/title">
   <xsl:if test="..//ulink">
      <h3>
         <xsl:value-of select="."/>
      </h3>
   </xsl:if>
</xsl:template>
```

Šablona zpracovává elementy `<title>`, jejichž rodičem je `<section>` mající za rodiče element `<chapter>`. Nadpis se na webové stránce zobrazí jen tehdy, jestliže rodič elementu `<title>` má potomka `<ulink>`. Pokud bychom stejnou úlohu řešili v TEXu, bylo by příslušné makro jistě delší.

Výstup do více formátů

XML nabízí mocný trransformační nástroj, kterým lze generovat výstupy v několika formátech. Implementace formátovacích objektů pro tisk již tak dobrá není. Zde se nabízí tisk prostřednictvím TEXu.

Generování jiných typů výstupů z TEXových souborů je problematičtější. Jedna metoda byla zveřejněna v již zmíněném článku [2], jinou metodu popsal Petr Olšák. V obou případech se však jednalo o převod souboru poměrně pevně definovaného formátu.

Existují nástroje pro konverzi LATEXových souborů do HTML. Nesmíme však zapomenout na známou skutečnost: „Only TEX can read TEX.“ Nemůžeme počítat s tím, že si takové nástroje poradí se všemi konstrukcemi. Čím více se odchýlíme od standardních LATEXových šablon, tím větší je pravděpodobnost, že konverze selže.

Jistou možnost představuje též metoda, o níž se zmínil Sebastian Rahtz [7]. Sice prováděl konverzi z LATEXu do SGML, ale úprava pro XML by měla být snadná. Pokud však původní dokument nebyl dobře logicky strukturován, získáme logické značkování v XML jedině ruční editací.

Vraťme se ještě k validačnímu schématu z kapitoly ?? uvedenému na straně 213. Validovali jsme jím soubor, z něhož měla být vytištěna brožura. Ta měla být následně převedena do PDF pro účely zveřejnění na CD. Současně měly být vytvořeny WWW stránky. Ve všech verzích měl být autorský rejstřík s hyertextovými odkazy. Kromě toho se ze souboru měly tisknout potvrzovací dopisy autorům a vybrané informace měly být načteny do databáze pro další zpracování. Na těchto úkolech spolupracovalo několik lidí, a to na různých operačních systémech. V takové situaci je XML ideálním nástrojem, protože vše se dělá vlastně jen pomocí transformačních stylů.

Nevýhody XML

Nasazení XML není vždy spojeno s optimistickými náladami, jak jsme si to až dosud líčili. Prvním problémem jsou matematické rovnice. Jejich ruční zápis přímo v MathML je s výjimkou triviálních vzorečků úlohou pro masochisty. Nemáme-li konverzní program pro převod z lidské podoby do MathML, je lepší zapsat alternativní text v TEXové notaci a vygenerovat z něj obrázek.

Na druhou nevýhodu narazíme v okamžiku, kdy se snažíme soubor XML vytisknout. Pokud nám záleží na kvalitě, měli bychom jít z XML přes TEX. Přibyl nám tedy jeden krok. Není-li krok navíc kompenzován ziskem lepší struktury nebo jiných výhod, jedná se pouze o časovou ztrátu. Pak je přímé nasazení TEXu rozhodně přirozenějším řešením.

TEX je i pro mírně zkušeného uživatele mocným nástrojem, který lze snadno využít i v osobních tiskovinách a písemnostech. Dobře se hodí zejména v případech, kdy grafický vzhled je výrazně důležitější než logická struktura dokumentu. Nedovedu si představit, že bych DocBook používal na psaní milostné korespondence, sazbu reklam, vizitek či akcidenčních tiskovin, jako jsou třeba svatební oznámení.

Závěr

Co říci závěrem? Na jedné straně jsou situace, kdy XML představuje ideální nástroj, na druhé straně však máme případy, kdy nám XML pouze přidělává práci a přímé využití TEXu či LATEXu je nespornou výhodou. Mezi oběma krajnostmi se prostírá široké pole, kde nelze jednoznačně určit, která z metod je lepší. Vždy bude do značné míry záležet na vkusu, znalostech a dovednostech uživatele, který má konkrétní datový soubor zadaným způsobem zpracovat.

Odkazy

1. Carroll L.: Through the Looking-Glass and what Alice Found There. Chapter II: The Garden of Live Flowers. In: The Annotated Alice. Penguin Books, Harmondsworth, Middlesex, England 1970.
2. Wagner Z.: Spolupráce databáze s LATEXem. Zpravodaj Československého sdružení uživatelů TEXu, **10**(1–3), 49–78 (2000).
3. Kosek J: XML pro každého. Grada Publishing 2000. ISBN 80-7169-860-1. `http://www.kosek.cz/xml/`.
4. `http://www.kosek.cz/xml/odkazy.html`
5. `http://www.kosek.cz/xml/odkazy.xsl`

6. Olšák P.: Jak používám TeX? 3. Makro pro layout požadovaný Verlag Dashofer,
http://www.olsak.net/texpraxe.html; též diskusní příspěvek na
http://groups.google.com/groups?hl=cs&lr=&ie=UTF-8&selm=
200403080914.i289ECsT004145%40relay.felk.cvut.cz
7. Rahtz S.: The inaugural meeting of TUG India. Zpravodaj Československého sdružení uživatelů TEXu, **7**(4), 173–177 (1974).

Loucký klášter ve Znojmě

Příspěvky v tomto čísle Zpravodaje pocházejí ze sborníku z konference SLT 2004 (seminář o Linuxu a TEXu), která proběhla v létě roku 2004 ve Znojmě. Jednání konference probíhalo v reprezentačních sálech zde zobrazeného kláštera. V době vlády komunistů v klášteře sídlili vojáci a podle toho ta stavba navenek vypadá. Pouze nepatrnou část obrovské budovy nyní obývá firma Znovín, která nám umožnila pořádat v těchto prostorách konferenci. Firma využívá mimo jiné rozsáhlý sklep kláštera k archivaci vín. Je tam uskladněno skoro milion lahví. Mimo jednání konference byl zorganizován i zajímavý doprovodný program. Je škoda, že se konference zúčastnilo málo lidí. Kdo tam nebyl, přišel o zajímavý zážitek.

Petr Olšák

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
Obálka:	Antonín Strejc Na obálce použito logo SLT, jehož autorkou je Petra Rychlá
Počet výtisků:	650
Odpovědný redaktor:	Zdeněk Wagner
Redaktor tohoto čísla:	Petr Olšák
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

Odborné příspěvky v tomto čísle byly přetištěny se svolením autorů z TEXové sekce sborníku SLT 2004 (sborník čtvrtého ročníku semináře o Linuxu a TEXu, Znojmo, 24.–27. června 2004).