Projekt OkTEX

Petr Olšák

Elektrotechnická fakulta ČVUT, Praha
Email: `petr@olsak.net`

Abstrakt: OkTEX je formát TEXu postavený na plainTEXu a balíčcích OFS, LANG a IENC. Je to pokus vytvořit multijazykové prostředí pro plainTEX, které svými vastnostmi 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.

1 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.

Petr Olšák (editor): SLT 2004 – sborník semináře o Linuxu a TEXu, str. 41–54, 2004.

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.

2 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 CSfonty. 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.

3 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.

4 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.

5 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 `\langinfo` (jméno jazyka), `\langinit` (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.

6 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.

7 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:

$$\text{soubor} \xrightarrow{\text{xord}} \xrightarrow{\text{IENC}} \text{T}_{\text{E}}\text{Xové fonty, } \texttt{dvi}$$

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`.

Reference

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.