Apr. 2004

LANG: multijazykový balíček pro plain

Petr Olšák

Vzhledem k tomu, že nejsem spokojen se systémem babel, vytvořil jsem si vlastní multijazykový balíček, který řeší možnosti přepínání mezi jazyky s maximálním možným komfortem. Načítá vzory dělení jednoho jazyka pro více kódování, je-li to potřeba. Umožňuje velmi přehledně deklarovat nový jazyk a zapsat pro něj stylový soubor s definicemi příkazů závislých na použitém jazyku.

Balíček LANG úzce spolupracuje s makrem OFS, viz `ofsdoc.tex`. Jedná se vlastně o příklad použití OFS pro plain. Pro provoz balíčku LANG je nutné OFS aspoň ve verzi Apr. 2004.

Kromě balíčku LANG je v přípravě ještě balíček IENC, který se stará o konverzi vstupního kódování na kódování fontů. Viz `iencdoc.tex`. Balíčky LANG, IENC a OFS spolu s Knuthovým makrem `plain.tex` dávají uživatelům plainu do ruky nástroj na efektivní správu fontů, jazyků a mapování vstupních kódování. Na rozdíl od LaTeXu však balíčky nedělají nic jiného, tj. návrh značkování dokumentu a tvorba maker definujících vzhled dokumentu zůstává stále v rukou a pod plnou kontrolou uživatele plainu.

LaTeX nepoužívám, tj. neplánuji portování balíku LANG pro LaTeX. Nicméně nabídnu zde použité myšlenky tvůrci balíčku `babel` a požádám ho, aby něco z toho do svého balíčku zahrnul. Jakmile `babel` bude umět to samé co LANG, nebude potřeba udržovat v TeXové distribuci zvláštní formát csLaTeX, protože bude možné veškerou funkcionalitu csLaTeXu zahrnout do nového (zatím bohužel neexistujícího) balíku `babel`.

1. Uživatelské rozhraní

1.1. Přepínání jazyků

`\setlang` Jazyk nebo kódování fontů nebo obojí můžeme přepnout příkazem

`\setlang[`⟨*zkratka-jazyka*⟩`/`⟨*kódování*⟩`]`

přičemž některý z těchto parametrů může chybět. Pokud chybí, ponechá se parametr pokud možno beze změny.

Balíček LANG v souboru `langdef.tex` deklaruje jazyky a jejich vazby na přepínače jazyků a na stylové soubory. Každý jazyk zde má deklarován seznam (třeba i více) kódování fontů, ve kterých je abeceda jazyka dosažitelná: tzv. *seznam kódování daného jazyka*. Každé kódování má přiřazenu implicitní rodinu fontů, která je pro toto kódování registrovaná a je deklarovaná v OFS.

Příkaz `\setlang` načte stylový soubor nového jazyka (pokud tento ještě nebyl načten) a zkontroluje, zda požadované kódování je v seznamu kódování daného jazyka. Pokud není, nastaví kódování na výchozí kódování daného jazyka bez ohledu na to, co bylo psáno v parametru příkazu `\setlang` a vypíše varování o změně kódování. Potom `\setlang` nastaví vzory dělení pro daný jazyk a použité kódování. Dále zkontroluje, zda

je kódování registrováno v aktuální rodině fontů v OFS. Pokud není, nastaví implicitní rodinu fontů daného kódování. Co dělá příkaz `\setlang` přesně, je uvedeno v sekci...

Místo příkazu `\setlang` můžeme použít přepínače jednotlivých jazyků `\english`, `\ukenglish`, `\czech`, `\french`, `\german` atd. Tyto příkazy jsou implicitně definovány tak, že spustí `\setlang[`⟨*zkratka-jazyka*⟩`/]`.

Seznam všech jazyků, které jsou v balíčku LANG deklarovány, včetně jejich přepínačů získáme pomocí příkazu `\showlangs`. Tento příkaz vypíše seznam jazyků na terminál a do log souboru například ve tvaru:

\showlangs

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

První sloupec obsahuje zkratku jazyka, která se používá v argumentu příkazu `\setlang`. Ve druhém sloupci je přepínač jazyka a ve třetím je název stylového souboru, který se automaticky přečte při prvním přepnutí do tohoto jazyka. Pak následuje seznam kódování, která lze pro abecedu daného jazyka použít. Jména kódování by měla být v souladu s dokumentem o názvech fontů v TEXu [1]. Například `8t` je kódování podle Corku v LaTEXu známé jako T1. Za jménem kódování následuje v lomené závorce interní číslo tabulky vzorů dělení, která byla v době iniTEXu načtena primitivem `\patterns` pro daný jazyk a dané kódování. Konkrétní hodnoty těchto čísel nejsou z uživatelského pohledu zajímavé (LANG je při při načítání tabulek vzorů dělení alokuje automaticky). Důležitá je pouze skutečnost, že některá kódování sdílejí stejnou tabulku vzorů dělení, protože abeceda jazyka má v těchto kódováních zcela stejné rozložení znaků.

Některá kódování mají přidělenu prázdnou tabulku vozorů, což je vyznačeno symbolem `<=>`. Znamená to, že slova se v tomto jazyku a kódování vůbec nerozdělují. Některá jiná kódování mají v seznamu uvedeno `<?>`, což zamená, že v době iniTEXu nebyla tabulka vzorů dělení načtena. Chceme-li s ní pracovat, musíme v souboru `langdef.tex` upravit argument příkazu `\loadpatterns` a spustit iniTEX znova. Jedině iniTEX při generování formátu umí načítat tabulky vzorů dělení. Přitom paměť TEXu je omezena, takže pravděpodobně není možné mít trvale načteny všechny tabulky vzorů dělení všech jazyků.

Podrobnější informaci o jednotlivém jazyku získáme tak, že přepneme do zvoleného jazyka a použijeme příkaz `\langinfo`. Například

\langinfo

```
\setlang[de2/]\message{\langinfo}
```

vypíše:

```
German language with hyphenation table adapted to new orthography.
```

Ještě podrobnější informaci lze získat příkazem `\printlangdoc`, viz sekci...

Pokusíme-li se příkazem `\setlang` přepnout do jazyka a kódování, pro které není načtena tabulka vzorů dělení slov, LANG vypíše varování:

```
LANG (1...): WARNING. Hyphen table [⟨jazyk⟩/⟨kódování⟩] was not loaded.
    I use "\defaulthyphentable=255" instead of given table.
    Modify \loadpatterns parameters in langdef.tex and
    run iniTeX again if you need given table exactly.
```

\defaulthyphentable

Znamená to, že místo požadované tabulky se použije tabulka `\defaulthyphentable`. Tento registr je implicitně nastaven na hodnotu 255, což představuje prázdnou tabulku, při které slova nebudou vůbec dělena. Registr můžeme ve svém dokumentu nastavit na

\hyphentable

tabulku vzorů jiného jazyka například pomocí příkazu `\hyphentable`:

```
\defaulthyphentable=\hyphentable[en/8t]
```

V tomto příkladě je registr `\defaulthyphentable` nastaven na na tabulku anglických vzorů dělení při kódování `8t`.

\langusage

Podobně jako v OFS si můžeme příkazem `\langusage` vypsat na terminál základní příkazy balíčku LANG.

\fotenc

Pozor: není vhodné pro přepínání kódování fontů přímo pracovat s makrem `\fotenc`, tj. používat `\def\fotenc{`⟨*kódování*⟩`}`. Tuto práci dělá příkaz `\setlang[/`⟨*kódování*⟩`]`, ale navíc tento příkaz udržuje v konzistenci s nastaveným ⟨*kódováním*⟩ vzory dělení slov, `\lccode` znaků atd.

Změní-li `\setlang` kódování fontů a je-li použit také balíček IENC, pak LANG informuje o této změně svého kolegu IENC, který může na základě této informace upravit kódovací tabulky vstupních znaků. Pokud ale IENC není použit, musíme se sami postarat o to, aby znaky ze vstupního souboru se správně mapovaly na znaky v použitém kódování fontů.

1.2. Inicializace a výchozí nastavení

Balíček LANG je potřeba inicializovat v iniTEXu, protože načítá vzory dělení jazyků, které jsou vyjmenovány v parametru `\loadpatterns` v souboru `langdef.tex`. Příklad použití balíčku najdeme v souboru `oktex.ini`.

Výchozí nastavení po inicializaci balíčku LANG je co nejvíce podobné makru `plain.tex`. Přesněji: nepoznáme rozdíl mezi makrem `plain.tex`, pokud nepoužijeme některé z nově definovaných maker z OFS nebo z balíčku LANG. Ovšem po prvním použití maker `\setlang` nebo `\setfonts` se situace mění, protože makra načítají fonty a soubory, ve kterých jsou předefinována makra plainu závislá na kódování (typicky `\ae` nebo `\v`).

Implicitní kódování je `8t` a implicitní rodina fontů `CMRoman`. Znamená to, že pokud se jako první použije `\setfonts[/]`, inicializuje se rodina `CMRoman` v kódování `8t`, která je implementována prostřednictvím tzv. EC fontů.

Implicitní jazyk je `none`. Toto je zvláštní jazyk, který nenačítá žádný stylový soubor a nepřepíná tabulku vzorů dělení. Akceptuje veškerá kódování, která jsou v balíčku LANG registrována. Pokud tedy první výskyt příkazu `\setlang` má jen parametr `[/`⟨*kódování*⟩`]`, pak LANG zůstává u jazyka `none` a přepne pouze kódování fontů.

První výskyt příkazu `\setlang` v dokumentu má jednu speciální funkci: při prázdném parametru ⟨*kódování*⟩ a neprázdné ⟨*zkratce jazyka*⟩ nastaví ⟨*kódování*⟩ na výchozí kódování zvoleného jazyka. Například `\setlang[cz/]` nastaví kódování na `8z`, protože toto je výchozí kódování pro češtinu. Všechny následující příkazy `\setlang` s prázdným parametrem ⟨*kódování*⟩ se už pokoušejí ctít nastavené kódování.

Implicitní kódování jazyka `none` je `8t`. Pokud tedy je prvním příkazem `\setlang[/]`, pak LANG pouze potvrdí kódování `8t` a inicializuje rodinu `CMRoman` v tomto kódování. Pokud pak následuje třeba `\setlang[cz/]`, zůstane už čeština v kódování `8t`, ačkoli se nejedná o výchozí kódování tohoto jazyka.

Přiřazení provedené příkazem `\setlang` je lokální, ale z tohoto pravidla existuje výjimka: ačkoli dáme první výskyt příkazu `\setlang` do skupiny, po opuštění skupiny se už nevrací kódování do výchozího stavu podle plainTEXu, ale potvrdí se v takovém případě kódování `8t` s jazykem `none`.

1.3. Použití stylových souborů

Stylový soubor se při prvním přepnutí do vybraného jazyka načte automaticky. V něm jsou obvykle připravena makra, která usnadňují sazbu v daném jazyku. Uživatel si může vytisknout dokumentaci k danému stylovému souboru pomocí přepnutí do vybraného jazyka a následném spuštění příkazu `\printlangdoc`. Například `\setlang[de2/]\printlangdoc` vytiskne (do `dvi`) dokumentaci k použití stylového souboru německého jazyka.

`\printlangdoc`

Ve stylovém souboru se obvykle definují pomocí `\langdef` makra, která jsou závislá na aktuálně zvoleném jazyku. Například makro `\today` se v různých stylových souborech definuje různě, ale jejich definice se vzájemně nepřekrývají. Je-li aktuální angličtina, vytiskne `\today` dnešní datum v angličtině, zatímco při aktuální češtině bude `\today` tisknout dnešní datum česky. Podrobněji o `\langdef` viz sekci ...

`\langdef`

Každý stylový soubor definuje (závisle na zvoleném jazyku) makro `\langinit`, ve kterém probíhá inicializace daného jazyka. V tomto makru bývá obvykle nastaveno `\lefthyphenmin` a `\righthyphenmin` pro daný jazyk. Dále se zde nastavuje `\fenchspacing` nebo jiné mezerování. Příkaz `\langinit` se spustí automaticky vždy při nastavení jazyka pomocí `\setlang`. Přiřazení v příkazu `\langinit` jsou lokální. Ve stylovém souboru se navíc zajistí, aby se nastavení z `\langinit` vrátila do původního stavu (podle plainTEXu), pokud se jazyk opustí tak, že se spustí `\setlang` jiného jazyka.

`\langinit`

Každý stylový soubor by měl definovat makro `\langphrases` (závislé na zvoleném jazyku), jehož spuštěním se definují makra `\abstractname`, `\appendixname`, `\bibname`, `\ccname`, `\chaptername`, `\contentsname`, `\enclname`, `\figurename`, `\headpagename`, `\headtoname`, `\indexname`, `\listfigurename`, `\listtablename`, `\partname`, `\prefacename`, `\proofname`, `\seename`, `\alsoseename`, `\refname`, `\tablename`, tak, že expandují na odpovídající fráze podle aktuálního jazyka. Tato makra sama o sobě nejsou závislá na aktuálním jazyku, ale při přechodu do nového jazyka je musíme předefinovat opětovným spuštěním makra `\langphrases`. Důvod tohoto chování je prostý: v dokumentu můžeme docela často přepínat mezi jazyky, ale texty v záhlaví, texty pro názvy kapitol apod. necháme v jediném zvoleném jazyku, který považujeme pro tento dokument za hlavní. Pro tento hlavní jazyk pak spustíme `\langphrases` a dále zůstávají tyto fráze neměnné.

`\langphrases`

Každý stylový soubor také definuje makro `\langinfo` (jak bylo ukázáno v předchozí sekci ...).

Stylové soubory by neměly samy od sebe inicializovat žádné aktivní znaky, ale pomocí (na jazyku závislém) příkazu `\activechar`⟨*znak*⟩ mohou umožnit uživateli nastavit zvolené znaky jako aktivní a využít jejich předdefinované funkce ze stylového souboru. Například styl pro češtinu umožní nastavit aktivní znak „`-`“, aby se tento znak choval jako `\discretionary{-}{-}{-}`. Uživatel ale musí explicitně

`\activechar`

tuto vlastnost znaku „-“ inicializovat příkazem `\activechar-`. Pokud ale uživatel použije např. `\activechar-` při aktuální angličtině, dostane na terminál pouze varování, že stylový soubor pro angličtinu nepodporuje aktivní znak „-“ a znak zůstane neaktivní. Konstrukce „`\czech \activechar-`“ se tedy podobá LaTeXovému „`\usepackage[split]{czech}`“.

Stylový soubor pro češtinu `lang-cz.tex` umožňuje také nastavit aktivní znak pro počítačové uvozovky (`\activechar"`). To způsobí, že `"text mezi počítačovými uvozovkami"` bude nahrazen „textem mezi skutečnými českými uvozovkami“. Je jenom na uživateli, zda si tuto vlastnost aktivuje pomocí `\activechar` nebo ne. Přehled všech předdefinovaných aktivních znaků daného stylového souboru lze získat tiskem dokumentace k jazyku pomocí `\printlangdoc`.

Nastavení zvoleného znaku jako aktivního je lokální uvnitř skupiny, ale pokud opustíme jazyk přepnutím do jiného jazyka (nikoli opuštěním skupiny), zůstává znak aktivní nadále a má pořád funkci, kterou získal v době jeho aktivace příkazem `\activechar`. Pokud to není žádoucí, musí uživatel deaktivovat znak manuálně například pomocí
\deactivechar `\catcode‘`⟨*znak*⟩`=12` (může použít zkratku `\deactivechar`⟨*znak*⟩), nebo musí znak opakovaně aktivovat v kontextu nového jazyka příkazem `\activechar`⟨*znak*⟩.

Uživatel si může zajistit deaktivaci znaku po opuštění jazyka automaticky makrem
\afterlang `\afterlang{`⟨*příkazy*⟩`}`. V takovém případě se ⟨*příkazy*⟩ spustí při opuštění jazyka přepnutím do jiného jazyka. Makro `\afterlang` je možné v rámci jednoho jazyka použít opakovaně. V takovém případě se ⟨*příkazy*⟩ kumulují do zásobníku, tj. při opuštění jazyka se vykonávají v opačném pořadí, než byly použitím maker `\afterlang` do zásobníku vloženy. Pokud je jazyk ukončen koncem skupiny (tj. nikoli přepnutím do jiného jazyka), zásobník se ruší a vůbec se neprovede.

Příklad použití maker `\activechar` a `\afterlang` je uveden v následující sekci … Je tam ukázka možného předefinování přepínače `\czech`.

V mnoha jazycích se používá sada dvojic znaků jako tzv. zkratky (shorthands), umožňující přístup ke speciálním znakům nebo jiným vlastnostem jazyka. Například Němci používají `"u` jako zkratku za ü, `"s` je zkratka za ß atd. Poláci zase pro změnu mají speciální znaky uvozeny běžným lomítkem `/`. Aby toto začalo fungovat, musí uživatel nastavit první znak z dvojice znaků jako aktivní. Je-li ve všech použitých jazycích tento znak definován stejně jako:

`LANG (l...): ‘`⟨*znak*⟩`’ is activated as expandable double-char prefix`

pak jej stačí aktivovat příkazem `\activechar` v kontextu jediného jazyka (tj. není potřeba jej znovu opakovaně aktivovat v ostatních jazycích). Přitom zkratky zůstávají závislé na aktuálním jazyku, tj. například v němčině expanduje `"u` na ü zatímco v maďarštině na ű. Na druhou stranu v češtině nám po takové aktivaci nahrazování palcových uvozovek skutečně českými fungovat nebude, protože v českém stylovém souboru není tabulka dvojic s prefixem `"` vůbec deklarována. Tento znak bude tedy v češtině expandovat jen na svůj neaktivní protějšek.

Stylový soubor může deklarovat k použitému jazyku několik nářečí. Poznáme to
\whichdialect tak, že makro `\whichdialect` expanduje po přepnutí do zkoumaného jazyka na slovo „default“. Pokud nejsou nářečí pro jazyk deklarována, expanduje `\whichdialect` na slovo „none“. Zvolením jiného nářečí tam, kde jsou nářečí deklarována, způsobí mírnou změnu chování některých maker (např. makra `\today` nebo `\langphrases`).

Pokud jsou nářečí pro aktuální jazyk deklarována, je možno přepnout do jiného
\setdialect nářečí než „default“ pomocí příkazu `\setdialect[`⟨*nářečí*⟩`]`. Při opětovném přepnutí do tohoto jazyka pomocí `\setlang` se znovu inicializuje nářečí `\default` a je nutno

znovu použít `\setdialect`. Seznam všech dosažitelných nářečí získáme jednak příkazem `\printlangdoc` a dále také příkazem `\setdialect[]`, který uloží do makra `\tmpc` všechna deklarovaná nářečí aktuálního jazyka oddělená mezerou.

1.4. Předefinování přepínačů jazyka

K přepínačům jazyka `\english`, `\czech`, `\german` atd. se uživatel může chovat „neuctivě“ a před jejich použitím si je může definovat podle své potřeby. Implicitní definice těchto přepínačů obsahují jen volání `\setlang[`⟨*jazyk*⟩`/]` s prázdným parametrem pro kódování. Takový přepínač se snaží zachovat původně nastavené kódování, pokud to je možné. Při prvním výskytu tohoto příkazu se nastaví výchozí kódování daného jazyka.

Typický příklad na předefinování přepínače je volba jiného než výchozího nářečí: `\def\jazyk{\setlang[`⟨*zkratka*⟩`/]\setdialect[`⟨*nářečí*⟩`]}`, nebo volba jiného než výchozího kódování: `\def\jazyk{\setlang[`⟨*zkratka*⟩`/`⟨*kódování*⟩`]`.

Pokud například chceme pro češtinu použít EC fonty, které jsou kódované v `8t`, a navíc využít aktivní znak „`-`“, který se bude při dělení slov rozpadat automaticky na dva spojovníky, pak se vyplatí definovat:

```
\def\czech{\setlang[cz/8t]\activechar-\afterlang{\deactivechar-}}
```

Je potřeba si rozmyslet, které jazyky v dokumentu budeme používat, jaké pro ně zvolíme kódování a podle toho definovat přepínače. V přepínačích může například za `\setlang` okamžitě následovat `\setfonts`, pomocí kterého dáváme najevo, v jaké rodině fontů si přejmeme (například mongolštinu) tisknout. Rovněž zde může být inicializace některých maker specifických pro zvolený jazyk. Vše záleží na volbě uživatele.

Nevhodný příklad: použijeme `\def\czech{\setlang[cz/8t]}`, ale necháme implicitní hodnotu přepínače `\slovak` a začneme po použití přepínače `\slovak` psát slovensky. V tomto jazyku zůstane výchozí kódování fontů `8z`. Pak přepneme do češtiny příkazem `\czech` a zpracováváme češtinu v kódování `8t`. Nyní ale pozor: pokud znovu přepneme do slovenštiny pomocí `\slovak`, máme už tento jazyk v kódování `8t`, protože implicitní přepínač ctí aktuální kódování. To možná není to, co jsme potřebovali.

Jiný příklad: píšeme anglicky v kódování `8t` (po použití přepínače `\english`) a přepneme do ruštiny pomocí `\russian`. V tomto jazyce není možné zůstat u kódování `8t`, protože ruská abeceda v něm není zahrnuta. LANG tedy vypíše varování a přepne do kódování `6a` (výchozí kódování ruštiny). Po opětovném přepnutí pomocí `\english` máme sice znovu angličtinu, ale fonty zůstávají v kodování `6a` a nevracejí se do kódování `8t`, protože anglická abeceda je v kódování `6a` zahrnuta. Je potřeba si rozmyslet, zda to je pro danou úlohu žádoucí chování. Pokud bychom později v anglickém textu chtěli přepnout pomocí `\setfonts` do jiného fontu, který není dostupný v kódování `6a`, ohlásí OFS chybu. Tato chyba je přitom jen důsledkem dřívějšího přechodného přepnutí do ruštiny.

Poznámka: pokud do ruštiny v předchozím příkladě přepneme jen uvnitř skupiny pomocí `{\russian ruský text} anglický text`, výše uvedený problém nenastává. Po ukončení skupiny se totiž kódování fontů vrací do stavu před zahájením skupiny, tj. v našem příkladě se vrací do očekávaného kódování `8t`.

2. Jak je to uděláno, aneb pohled do hloubky

2.1. Deklarace v souboru `langdef.tex`

\declareenc

Pomocí skupiny příkazů `\declareenc` je v souboru `langdef.tex` deklarována množina kódování fontů, se kterou budeme dále pracovat:

`\declareenc` ⟨*kódování*⟩⟨*mezera*⟩⟨*Rodina*⟩⟨*mezera*⟩

Příkaz deklaruje kódování a přiřadí mu implicitní rodinu fontů. Zkontroluje, zda existuje soubor `ofs-`⟨*kódování*⟩`.tex`, zda existuje implicitní rodina a zda je tato rodina v OFS pro dané kódování registrovaná. Pokud něco z toho není splněno, ohlásí chybu a ⟨*kódování*⟩ nedeklaruje.

\declarelang

Následuje skupina příkazů `\declarelang`, které deklarují jednotlivé jazyky:

`\declarelang` \⟨*přepínač*⟩ ⟨*zkratka-jazyka*⟩ ⟨*stylový-soubor*⟩ ⟨*seznam-kódování*⟩ `;`

Jednotlivé parametry jsou odděleny mezerou. Pokud ⟨*stylový-soubor*⟩ neexistuje, příkaz ohlásí chybu a ukončí činnost. Jinak provede:

`\def\`⟨*přepínač*⟩`{\setlang[`⟨*zkratka-jazyka*⟩`/]`

a uloží si do paměti jméno ⟨*stylového-souboru*⟩. Dále zkontroluje, zda jsou všechna kódování ze ⟨*seznamu-kódování*⟩ deklarovaná. Nedeklarovaná kódování ignoruje a napíše o tom varování. Pokud po odstranění nedeklarovaných kódování v seznamu žádné kódování nezbyde, jazyk zůstává nedeklarován.

Jednotlivá kódování jsou v ⟨*seznamu-kódování*⟩ oddělována mezerou a před ukončovacím středníkem je také mezera. První kódování se stává výchozím kódováním jazyka a nesmí mu předcházet samotné rovnítko. Pokud některému následujícímu kódování předchází samotné rovnítko, znamená to, že pro přepnutí do tohoto kódování vystačíme se stejnými vzory dělení slov jako pro předchozí kódování. IniTEX tedy nebude načítat nové vzory. Jinými slovy: abeceda jazyka je v tomto kódování stejná jako v předchozím.

Před kódováním může také předcházet hranatá závorka a rovnítko ve tvaru:

`[`⟨*jiný-jazyk*⟩`/`⟨*kódování*⟩`]=`⟨*nové-kódování*⟩

V takovém případě je pro ⟨*nové-kódování*⟩ použita stejná tabulka vzorů dělení slov, jako pro `[`⟨*jiný-jazyk*⟩`/`⟨*kódování*⟩`]`. Důležité: v parametrech `\loadpatterns` musí být tabulky vzorů dělení slov pro ⟨*jiný-jazyk*⟩ zavedeny dříve, než pro právě deklarovaný jazyk, jinak zůstává ⟨*nové-kódování*⟩ s prázdnou tabulkou.

Místo `[`⟨*jiný-jazyk*⟩`/`⟨*kódování*⟩`]=` může být uvedeno jen `[]=`, což znamená, že ⟨*nové kódování*⟩ bude mít prázdnou tabulku vzorů dělení, kterou LANG rezervuje pod číslem 255.

\loadpatterns

Příkaz `\loadpatterns` načte vzory dělení vyjmenovaných jazyků.

`\loadpatterns` ⟨*seznam-zkratek-jazyků*⟩ `;`

Zkratky jazyků jsou odděleny mezerami a před ukončovacím středníkem musí být také mezera. Za zkratkou jazyka může následovat `{`⟨*aktuální-seznam-kódování*⟩`}` oddělený od předchozí i následující zkratky jazyka mezerou. Jestliže za zkratkou jazyka není uveden `{`⟨*aktuální-seznam-kódování*⟩`}`, pak se načtou vzory dělení pro všechna kódování ze ⟨*seznamu-kódování*⟩. Pokud ale za zkratkou následuje `{`⟨*aktuální-seznam-kódování*⟩`}`, načtou se vzory dělení podle tohoto aktuálního seznamu. Předpokládá se, že aktuální seznam kódování je podmnožinou seznamu kódování a že jej použijeme v případě, kdy chceme načíst menší počet tabulek vzorů dělení, protože šetříme pamětí TEXu.

V ⟨*aktuálním-seznamu-kódování*⟩ je možno použít stejných syntaktických pravidel (s rovnítky a hranatými závorkami), jako bylo vysvětleno výše pro ⟨*seznam-kódování*⟩ u příkazu `\declarelang`.

Příkaz `\loadpatterns` přesně dělá toto:

- Založí novou skupinu (`\bgroup`)
- Pro každý jazyk ze ⟨*seznamu-zkratek-jazyků*⟩ projde ⟨*seznam-kódování*⟩ daného jazyka (resp. ⟨*aktuální-seznam-kódování*⟩, pokud je za zkratkou jazyka uveden). Každé takto vytvořené dvojici ⟨*jazyk*⟩/⟨*kódování*⟩ přidělí tabulku vzorů dělení slov způsobem popsaným níže.
- Ukončí skupinu (`\egroup`)

Pro každou dvojici ⟨*jazyk*⟩/⟨*kódování*⟩ dělá `\loadpatterns` toto:

- Je-li kódování označeno samotným rovnítkem, přidělí dvojici ⟨*jazyk*⟩/⟨*kódování*⟩ tabulku vzorů dělení z předchozí dvojice a činnost pro tuto dvojici ⟨*jazyk*⟩/⟨*kódování*⟩ končí.
- Jestliže ⟨*kódování*⟩ bylo deklarováno pomocí zápisu `[...]=`⟨*kódování*⟩, pak přidělí dvojici ⟨*jazyk*⟩/⟨*kódování*⟩ tabulku, která je vyznačena v hranaté závorce. Činnost pro tuto dvojici ⟨*jakyk*⟩/⟨*kódování*⟩ končí.
- Pokud byla pro danou dvojici ⟨*jazyk*⟩/⟨*kódování*⟩ už tabulka vzorů dělení přidělena, vypíše varování a vzory dělení znovu nenačítá.
- Založí další skupinu (`\begingroup`).
- Nastaví `\fotenc` podle ⟨*kódování*⟩.
- Přečte soubor `hyph-`⟨*lang*⟩`.tex`. Tam se předpokládá deklarace `\patterns` a případně `\hyphenation`. Pokud nejsme v iniTEXu, příkaz `\patterns` ohlásí chybu.
- Ukončí skupinu (`\endgroup`)
- Dvojici ⟨*jazyk*⟩/⟨*kódování*⟩ přidělí stávající hodnotu `\language`.
- Zvětší `\language` o jedničku a přechází na další dvojici ⟨*jazyk*⟩/⟨*kódování*⟩.

Skutečnost, že jsou příkazy `\patterns` a `\hyphenation` uvnitř skupin, nám nevadí, protože jejich přiřazení je globální.

`\declarehyphentable` Soubory `hyph-`⟨*lang*⟩`.tex` by měly obsahovat příkaz

`\declarehyphentable[`⟨*jazyk*⟩`/`⟨*kódování*⟩`]`

který podle aktuální hodnoty `\fotenc` může před skutečným načtením `\patterns` zavést konverzní tabulku. Pro různá `\fotenc` se tedy nastaví různé konverzní tabulky, takže opakovaným načtením stejného souboru `hyph-`⟨*lang*⟩`.tex` dostáváme pro různá `\fotenc` různé vzory dělení. O to šlo. Podrobněji se o této věci zmiňuji v sekci…, kde je seznam doporučení, jak by měl soubor `hyph-`⟨*jazyk*⟩`.tex` vypadat.

Příkaz `\loadpatterns` je možné použít opakovaně k načtení vzorů dělení dalších jazyků. Například v souboru `langdef.tex` můžeme načíst jen vzory dělení „základních jazyků“, zatímco v nějakém externím souboru, který generuje například instalační program, můžeme použít `\loadpatterns` se seznamem dalších jazyků. Tyto jazyky si uživatel například vybral v instalačním programu pomocí nějakého klikacího rozhraní.

2.2. Příkaz `\setlang`

`\setlang` Nyní podrobně popíšeme, co dělá příkaz `\setlang[`⟨*jazyk*⟩`/`⟨*kódování*⟩`]`

- Je-li nějaký parametr prázdný, doplní jej aktální hodnotou jazyka (resp. kódování).

- Je-li nový ⟨*jazyk*⟩ shodný se stávajícím a zároveň nové ⟨*kódování*⟩ shodné se stávajícím, nedělá nic a končí. Výjimkou z tohoto pravidla je první spuštění `\setlang`, které vždy provede nastavení jazyka a kódování: při prázdných parametrech použije jazyk `none` a kódování `8t`, při prázdném ⟨*kódování*⟩ a neprázdném ⟨*jazyku*⟩ použije výchozí kódování ⟨*jazyka*⟩.
- Pokud je ⟨*jazyk*⟩ nedeklarovaný, vypíše chybové hlášení a končí.
- Je-li ⟨*kódování*⟩ nedeklarované nebo není uvedeno v ⟨*seznamu-kódování*⟩ pro daný jazyk, vypíše varování a nastaví výchozí na defaultní kódování daného jazyka.
- Jestliže došlo ke změně kódování nebo jazyka, nastaví tabulku vzorů dělení slov podle zvoleného jazyka a kódování. Pokud tabulka nebyla v době iniTeXu načtena, vypíše varování a nastaví tabulku `\defaulthyphentable`.
- Jestliže byl změněn jazyk, pak spustí `\afterlangstack`, který obsahuje zásobník ⟨*příkazů*⟩ z `\afterlang` pro deaktivaci předchozího jazyka.

`\currentlang` `\currentdialect`

- Byl-li změněn jazyk, nastaví `\def\currentlang{`⟨*jazyk*⟩`}` a `\currentdialect=0` a dále spustí `\langinit` nového jazyka. Před příkazem `\langinit` ještě zkontroluje, zda byl načten stylový soubor jazyka a pokud ne, načte jej. (Při čtení má `\globaldefs=1` a ignoruje mezery na koncích řádků.)
- Jestliže nebylo změněno ⟨*kódování*⟩, pak končí.
- Spustí příkaz `\lccodesback`, který vrátí `\lccode` a `\uccode` do výchozího stavu podle plainTEXu (viz sekce…).
- Je-li použito IENC, spustí `\unsetienc[\ienc/\fotenc]` (podle starého `\fotenc`).
- Provede `\def\fotenc{`⟨*kódování*⟩`}`. Nebyl-li načten soubor `ofs-`⟨*kódování*⟩`.tex`, načte jej. (Při čtení má `\globaldefs=1` a ignoruje mezery na koncích řádků.)
- Provede `\global\let\gfotenc=\fotenc` a `\aftergroup\resetcodes`. Důvod tohoto kroku je vysvětlen v sekci …
- Je-li použito IENC, provede `\setienc [\ienc/\fotenc]`.
- Spustí příkaz `\`⟨*kódování*⟩`:lccodes`, který je deklarovaný pomocí `\modifydef` v souboru `ofs-`⟨*kódování*⟩`.tex`. Tím by se měly správně nastavit `\lccode`, `\uccode` a `\catcode` Při jejich nastavení je přechodně `\globaldefs=1`. Podrobněji o tomto kroku viz sekci …
- Pro další potřebu nastaví ⟨*kódování*⟩ jako výchozí kódování daného jazyka.
- Následuje-li těsně za `\setlang` příkaz `\setfonts` (bez mezery), provede ho, jinak provede `\setfonts[/]`.
- Pokud `\setfonts` skončil neúspěšně, vypíše varování a nastaví výchozí rodinu podle daného kódování. Pro tuto rodinu znovu provede `\setfonts`.

Je vidět, že za příkazem `\setlang` může okamžitě následovat `\setfonts` a pak se mění parametry `\setlang` a `\setfonts` společně. Příkaz `\setfonts` ale nesmí být ukrytý uvnitř makra. Dvojice:

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

nám tedy umožňuje přepínat v pětirozměrném prostoru parametrů, přičemž přepínat můžeme třeba jen podél některé z pěti os. Pokud požadavek na přepnutí není možné splnit (např. ⟨*Rodina*⟩ neobsahuje abecedu požadovaného jazyka), LANG a OFS nastaví případně jiné ⟨*kódování*⟩ a ⟨*Rodinu*⟩ a vypíše o tom varování.

2.3. Příkaz `\langdef`

`\langdef`

Základním příkazem používaným ve stylových souborech je `\langdef`, který má dvě možnosti zápisu:

`\langdef (`⟨*seznam jazyků*⟩`) \`⟨*makro*⟩ ⟨*maska parametrů*⟩ `{`⟨*tělo makra*⟩`}`

nebo

`\langdef \`⟨*makro*⟩ ⟨*maska parametrů*⟩ `{`⟨*tělo makra*⟩`}`

Pokud se použije zkrácená verze zápisu (bez ⟨*seznamu jazyků*⟩ v závorce), `\langdef` si
\langdeflist doplní ⟨*seznam jazyků*⟩ z makra `\langdeflist`. Parametr ⟨*seznam jazyků*⟩ je seznam zkratek jazyků oddělených od sebe mezerami.

Příkaz `\langdef` definuje \⟨*makro*⟩ podobně, jako to dělá `\def`. To znamená, že ⟨*maska parametrů*⟩ a ⟨*tělo definice*⟩ se zapisují stejně jako při použití primitivu `\def`. Rozdíl je ale v tom, že `\makro` má daný význam jen pro jazyky ze ⟨*seznamu jazyků*⟩. Jinde můžete použít `\langdef` stejného \⟨*makra*⟩ pro jiné jazyky, aniž by došlo k překrývání definic. Typickým příkladem použití `\langdef` je definování makra `\today` pro různé jazyky různě.

Chceme-li definovat \⟨*makro*⟩ s prefixem (smysl má asi jen prefix `\long`), pak je nutno před použitím makra napsat `\let\langdefprefix=`\⟨*prefix*⟩. Po každém provedení `\langdef` se vrací `\langdefprefix` na hodnotu `\relax`.

Pokud uživatel použije ve svém dokumentu \⟨*makro*⟩ definované pomocí `\langdef`, pak se použije ta hodnota makra, která odpovídá aktuálně zvolenému jazyku. Jestliže je \⟨*makro*⟩ definované jen pro jiné jazyky než aktuálně zvolený, LANG ještě zkontroluje, zda \⟨*makro*⟩ bylo definované pomocí `\langdef (*)` \⟨*makro*⟩. Pokud ano, spustí toto makro. Jinak LANG vypíše varování o nedefinovaném \⟨*makru*⟩ pro aktuální jazyk a \⟨*makro*⟩ je ignorováno.

Popíšeme nyní podrobně, co `\langdef` dělá. Nejprve definuje \⟨*makro*⟩ jako `\langcommand{`⟨*makro*⟩`}`. To dělá pro všechny jazyky stejně, takže nedochází k překrývání definic stejného makra. Dále provede:

`\langdefprefix\def \c:>`⟨*makro*⟩`(`⟨*jazyk*⟩`)` ⟨*maska parametrů*⟩ `{`⟨*tělo makra*⟩`}`

kde ⟨*jazyk*⟩ je první jazyk ze ⟨*seznamu jazyků*⟩ a `\c:>`⟨*makro*⟩`(`⟨*jazyk*⟩`)` je kontrolní sekvence, kterou zde definujeme. Pokud ⟨*seznam jazyků*⟩ obsahuje další jazyky, `\langdef` pro každý z nich ještě provede

`\let \c:>`⟨*makro*⟩`(`⟨*další jazyk*⟩`) = \c:>`⟨*makro*⟩`(`⟨*jazyk*⟩`)`

\langcommand Další práci provede až příkaz `\langcommand{`⟨*makro*⟩`}` v době použití \⟨*makra*⟩. Tento příkaz překontroluje, zda je definováno makro `\c:>`⟨*makro*⟩`(\currentlang)`. Pokud je definováno, pak toto makro spustí. Není-li toto makro definováno, pokusí se spustit `\c:>`⟨*makro*⟩`(*)`. Pokud ani toto makro není definováno, vypíše chybové hlášení
\langcommandwarn použitím příkazu `\langcommandwarn{`⟨*makro*⟩`}`.

Má-li `\expandaction` hodnotu `\noexpand`, pak `\langcommand{`⟨*makro*⟩`}` zpětně expanduje na \⟨*makro*⟩, takže je „robustní“ v LaTeXovém smyslu tohoto slova. O triku s `\expandaction` je řečeno více v dokumentaci k OFS v sekci 3.2.

Příkaz `\langdef` odmítá předefinovat kontrolní sekvence, které byly definovány jinak než pomocí `\langdef`. Při pokusu o předefinování takové sekvence `\langdef` vypíše
\langdefwarn varování pomocí příkazu `\langdefwarn`\⟨*makro*⟩. Přesněji: má-li kontrolní sekvence \⟨*makro*⟩ jiný význam než `\undefined`, `\relax` nebo `\langcommand{`⟨*makro*⟩`}`, pak `\langdef` ponechá \⟨*makro*⟩ nezměněno a vypíše jen varování. Důvod tohoto opatření je ten, že ve stylových souborech může být definováno pomocí `\langdef` plno maker, o nichž uživatel vůbec netuší a jejichž názvy může uživatel mít ve svých vlastních definicích. Příkaz `\langdef` v takovém případě ctí uživatelovy definice. Podobné opatření je uplatněno na `\characterdef` z OFS.

2.4. Příkaz \setdialect

\setdialect
\whichdialect

Příkaz `\setdialect[`⟨*nářečí*⟩`]` spolupracuje s makrem `\whichdialect`, které může být pomocí `\langdef` (závisle na použitém jazyku) definováno ve stylovém souboru. Pravidla definice makra `\whichdialect` jsou uvedena v sekci...

\currentdialect

Není-li makro `\whichdialect` pro aktuální jazyk vůbec deklarováno, pak příkaz `\setdialect[`⟨*nářečí*⟩`]` vypíše jen varování a končí. Jinak nastaví numerický registr `\currentdialect` na číselnou hodnotu ⟨*nářečí*⟩, která je deklarována v makru `\whichdialect`. Dělá to tak, že nastaví `\currentdialect=0` a dále v cyklu zkouší spustit `\whichdialect`, kde při každém dalším pokusu zvětší `\currentdialect` o jedničku. Cyklus je ukončen v případě, kdy `\whichdialect` vrátí ⟨*nářečí*⟩ stejné jako v parametru příkazu `\setdialect`. Cyklus je ukončen také tehdy, když `\whichdialect` vrátí prázdnou hodnotu. V tomto druhém případě ještě příkaz `\setdialect` vrátí hodnotu `\currentdialect` do stavu, ve kterém byla před zavoláním příkazu, a vypíše varování, že ⟨*nářečí*⟩ není pro aktuální jazyk deklarováno. Všechna přiřazení jsou lokální.

Jestliže použijeme `\setdialect[]`, pak příkaz `\setdialect` provádí stejnou činnost jako je uvedena výše. Navíc každou neprázdnou návratovou hodnotu makra `\whichdialect` přidá do makra `\tmpc` a odděluje je mezerou. V tomto případě nepíše žádná varování.

2.5. Tabulky dvojic znaků, z nichž první je aktivní

V některých jazycích se pracuje s tzv. zkratkami (shorthands), viz též sekci ... LANG umožňuje ve stylových souborech deklarovat tabulky maker dvojznaků nezávisle pro každý jazyk zvlášť. Uživatel pak může nastavit první znak dvojznaků jako aktivní a definovat jej jako `\expdoublechar`. Tím začnou dvojznaky expandovat na deklarovaná makra v závislosti na aktuálním jazyku. Příklad:

```
\def\langdeflist {de de2} % ukázka ze stylového souboru lang-de.tex
\doublechardef "u{\"u}
\doublechardef "o{\"o}
\doublechardef "s{\ss}
... atd.
\def\langdeflist {hu}     % ukázka ze stylového souboru lang-hu.tex
\doublechardef "u{\H u}
\doublechardef "o{\H o}
... atd.
\activedef "{\expdoublechar"}  % toto provede makro \activechar"
```

Po provedení `\activedef"` se stane uvozovka aktivní. Pokud má `\currentlang` hodnotu `de` nebo `de2`, pak dvojice `"u` expanduje na `\"u`, zatímco při `\currentlang` s hodnotou `hu` expanduje tatáž dvojice na `\H u`. Při jiném, `\currentlang` expanduje samotná uvozovka na svůj neaktivní protějšek za kterým následuje znak „u“ jako obyčejný znak.

\doublechardef

Vysvětlíme si nyní podrobně, jak je to uděláno. Makro `\doublechardef` má dvě varianty zápisu stejně jako `\langdef`:

`\doublechardef (`⟨*seznam jazyků*⟩`)` ⟨*znak1*⟩⟨*znak2*⟩⟨*maska parametrů*⟩`{`⟨*makro*⟩`}`

nebo

`\doublechardef` ⟨*znak1*⟩⟨*znak2*⟩⟨*maska parametrů*⟩`{`⟨*makro*⟩`}`

V případě varianty bez explicitního ⟨*seznamu jazyků*⟩ se použije ⟨*seznam jazyků*⟩ z makra `\langdeflist`. ⟨*Maska parametrů*⟩ je samozřejmě nepovinná a většinou se nepoužívá, protože obvyklé zkratky expandují na makra bez parametrů.

Příkaz `\doublechardef` provede:

`\def \d:>`⟨*znak1*⟩⟨*znak2*⟩`(`⟨*jazyk*⟩`)` ⟨*maska parametrů*⟩`{`⟨*makro*⟩`}`

kde `\d:>`⟨*znak1*⟩⟨*znak2*⟩`(`⟨*jazyk*⟩`)` je nově definovaná kontrolní sekvence a ⟨*jazyk*⟩ je první jazyk ze ⟨*seznamu jazyků*⟩. Pro ostatní jazyky makro provede ztotožnění `\d:>`⟨*znak1*⟩⟨*znak2*⟩`(`⟨*další jazyk*⟩`)` se sekvencí `\d:>`⟨*znak1*⟩⟨*znak2*⟩`(`⟨*jazyk*⟩`)` pomocí `\let`.

Místo ⟨*znaku2*⟩ je možno použít symbol `#1`:

`\doublechardef` ⟨*znak1*⟩`#1`⟨*maska dalších parametrů*⟩`{`⟨*makro*⟩`}`

V tomto případě `\doublechardef` provede:

`\def \d:>`⟨*znak1*⟩`(`⟨*jazyk*⟩`) #1`⟨*maska dalších parametrů*⟩`{`⟨*makro*⟩`}`

a provede ztotožnění pro ostatní jazyky ze seznamu parametrů.

`\expdoublechar` `\ssdoublechar`

Informace uložené pomocí `\doublechardef` využije makro `\expdoublechar` nebo `\ssdoublechar`. Obě makra dělají zhruba totéž. Přečtou dva následující znaky ⟨*znak1*⟩ a ⟨*znak2*⟩ a expandují na první definovanou sekvenci z následujícího seznamu sekvencí:

`\d:>`⟨*znak1*⟩⟨*znak2*⟩`(\currentlang)`
`\d:>`⟨*znak1*⟩`(\currentlang)`
`\d:>`⟨*znak1*⟩⟨*znak2*⟩`(*)`
`\d:>`⟨*znak1*⟩`(*)`

Jedná-li se o sekvenci, v jejímž názvu chybí ⟨*znak2*⟩, pak se připojí za tuto sekvenci navíc `{`⟨*znak2*⟩`}`, tj. ⟨*makro*⟩ může pracovat se ⟨*znakem2*⟩ jako s parametrem `#1`.

Není-li žádná z uvedených sekvencí definována, makra `\expdoublechar` nebo `\ssdoublechar` expandují na neaktivní ⟨*znak1*⟩ a připojí za něj ⟨*znak2*⟩. Na kategoriích ⟨*znaku1*⟩ a ⟨*znaku2*⟩ nezáleží.

Makro `\expdoublechar` provádí veškerou svou činnost na úrovni expand procesoru. Není tedy ohrožena tvorba ligatur a kerningových párů s okolními znaky ani dělení slov, pokud se makro vyskytuje uvnitř slova. Bohužel, expand procesor neumí ošetřit, zda náhodou není ⟨*znak2*⟩ mezera. Mezeru bez varování ignoruje a za ⟨*znak2*⟩ považuje až první nemezerový znak.

Pokud se nám to nelíbí, lze použít místo `\expdoublechar` makro `\ssdoublechar` (space-sensitive double-char). Toto makro pracuje s `\futurelet`, tj. využívá příkazy hlavního procesoru. Bohužel, pak nefungují ligatury s okolními znaky ani kerningové páry. Na druhou stranu makro neignoruje mezeru jako ⟨*znak2*⟩ a je odolné i proti symbolům „`{`“ nebo „`}`“ v místě ⟨*znaku2*⟩. Pro tyto speciální případy můžeme dokonce deklarovat vlastní ⟨*makro*⟩ pomocí:

`\doublechardef` ⟨*znak1*⟩`\space` ⟨*maska parametrů*⟩`{`⟨*makro*⟩`}`
`\doublechardef` ⟨*znak1*⟩`\bgroup` ⟨*maska parametrů*⟩`{`⟨*makro*⟩`}`
`\doublechardef` ⟨*znak1*⟩`\egroup` ⟨*maska parametrů*⟩`{`⟨*makro*⟩`}`

Příkaz `\ssdoublechar` nejprve zkontroluje, zda ⟨*znak2*⟩ není některý z těchto znaků: „mezera“, „`{`“ nebo „`}`“. Pokud ano, pak pozmění ⟨*znak2*⟩ na odpovídající kontrolní sekvenci: `\space`, `\bgroup` nebo `\egroup` a spustí `\expdoublechar`. Nyní už bude načten ⟨*znak2*⟩ do parametru maker bez problémů.

Chceme-li deklarovat vlastní makro, které reaguje na „`{`“ nebo „`}`“ v místě ⟨*znaku2*⟩, pak bude asi potřeba, aby makro expandovalo (mimo jiné) na odpovídající `\bgroup` resp. `\egroup`, jinak by mohl zůstat dokument při použití takového makra nevybalancovaný.

Makra `\expdoublechar` i `\ssdoublechar` jsou vybavena testem na to, zda je kontrolní sekvence `\expandaction` rovna `\noexpand`. Pokud ano, makra expandují na `\noexpand`⟨*znak1*⟩, za kterým následuje ⟨*znak2*⟩.

Makra `\expdoublechar` a `\ssdoublechar` pracují stejně v textovém i matematickém módu, pokud pro ⟨*znak1*⟩ není deklarováno matematické makro pomocí:

`\doublechardef` ⟨*znak1*⟩`\mathchar` ⟨*maska parametrů*⟩`{`⟨*matematické makro*⟩`}`

V takovém případě `\expdoublechar`⟨*znak1*⟩ i `\ssdoublechar`⟨*znak1*⟩ expandují v matematickém módu na ⟨*matematické makro*⟩, zatímco v textovém módu se chovají způsobem popsaným výše.

Protože výše uvedené ⟨*matematické makro*⟩ se použije vždy bez závislostí na hodnotě ⟨*znaku2*⟩, je někdy výhodné je vůbec nedeklarovat a místo toho větvit na případ matematického módu jednotlivě u každého ⟨*makra*⟩ deklarovaného pomocí `\doublechardef`.

`\textormath` Můžeme k tomu využít příkaz `\textormath{`⟨*text-makro*⟩`}{`⟨*math-makro*⟩`}`, který expanduje na odpovídající makro podle toho, zda je TEX v textovém nebo matematickém módu. Tato makra mohou za sebou okamžitě číst další znaky ze vstupu bez nutnosti přeskakovat `\else` nebo `\fi`. Příklad použití najdeme v souboru `lang-de.tex`.

2.6. Práce s aktivními znaky

`\activedef` V ukázce na začátku předchozí sekce byl zmíněn příkaz `\activedef`.

`\activedef` ⟨*znak*⟩⟨*maska parametrů*⟩`{`⟨*makro*⟩`}`

Příkaz `\activedef` nastaví ⟨*znak*⟩ jako aktivní a dále jej definuje stejně, jako by to udělal primitiv `\def`. Pokud se ⟨*znak*⟩ vyskytuje v masce parametrů nebo v těle makra, pak tam má ještě svoji původní (pravděpodobně neaktivní) kategorii. Příkaz definuje makro přímočaře, tj. není zde implementována závislost na hodnotě `\currentlang`.

Při tvorbě maker, která pracují s aktivními znaky, se může hodit použít příkaz `\maybeactive` `\maybeactive`, který má dva parametry:

`\maybeactive` ⟨*token1*⟩⟨*token2*⟩

Makro ponechá ⟨*token1*⟩ beze změny, ale než mu předá řízení, může změnit kategorii ⟨*tokenu2*⟩ na aktivní. Udělá to pouze tehdy, když znak s ASCII hodnotou ⟨*tokenu2*⟩ má v době činnosti makra aktivní kategorii. Jinak ponechá ⟨*token2*⟩ beze změny. Příklad použití:

```
\def\test {\maybeactive \ifx *\tmpa ...\else ...\fi}
```

Příklad ukazuje, jak můžeme porovnávat znak v `\tmpa` (třeba tam byl uložen pomocí `\futurelet`) s konkrétním znakem. Nechť v době definice makra `\test` má hvězdička kategorii 12. Je-li pak před spuštěním makra `\test` provedeno `\let\tmpa=*`, pak `\ifx` vrátí hodnotu true nezávisle na tom, zda v tomto okamžiku je kategorie hvězdičky 12 nebo 13.

Makro `\maybeactive` pracuje na úrovni hlavního procesoru a využívá pomocné kontrolní sekvence `\tmpc` a `\tmpd`.

2.7. Princip nastavování `\lccode` znaků

Připomeňme, že `\setlang` při změně kódování fontů spustí `\lccodesback` a dále příkaz `\fotenc:lccodes`, který má být deklarován v souboru `ofs-\fotenc.tex`. Tím by měly být nastaveny `\lccode`, `\uccode` a případně `\catcode` podle nového kódování. Nastavení se provede globálně, protože v době spuštění těchto maker je přechodně nastaveno `\globaldefs=1`.

`\lccodesback` Příkaz `\fotenc:lccodes` by měl definovat nový význam makra `\lccodesback`, které by mělo obsahovat příkazy, pomocí nichž se nové nastavení `\lccode`, `\uccode` a případně `\catcode` vrátí do původního stavu podle standardu z iniTEXu a souboru `plain.tex`. Příště, až bude `\setlang` přepínat na další kódování, se toto makro `\lccodesback` použije, takže nové makro `\fotenc:lccodes` může předpokládat, že výchozí stav `\lccode`, `\uccode` a `\catcode` je podle iniTEXu a plainu. Při prvním nastavení kódování (první volání `\setlang` v dokumentu) je `\lccodesback` nastaveno na `\relax`, protože výchozí stav kódů je skutečně podle iniTEXu a podle plainu.

Příkaz `\fotenc:lccodes` může pro nastavení kódů využít následující připravená makra:

`\lccodes`
- `\lccodes`⟨*znak1*⟩⟨*znak2*⟩ – nastaví `\lccode` ⟨*znaku1*⟩ i ⟨*znaku2*⟩ na ⟨*znak1*⟩ a nastaví `\uccode` ⟨*znaku1*⟩ i ⟨*znaku2*⟩ na ⟨*znak2*⟩. Pokud mají ⟨*znak1*⟩ anebo ⟨*znak2*⟩ kategorii 12, nastaví jim kategorii 11.

`\lccodesloop`
- `\lccodesloop`⟨*znak1*⟩⟨*znak2*⟩⟨*znak3*⟩ – provede cyklus, který spouští následující příkazy: `\lccodes`⟨*znak1*⟩⟨*znak3*⟩, `\lccodes`⟨*znak1+1*⟩⟨*znak3+1*⟩, atd. až po poslední příkaz `\lccodes`⟨*znak1+k=znak2*⟩⟨*znak3+k*⟩. Jinými slovy: cyklus se provádí přes „malá písmena“ v rozsahu ⟨*znak1*⟩ až ⟨*znak2*⟩, přičemž „velké písmeno“ pro ⟨*znak1*⟩ je ⟨*znak3*⟩ a další „velká písmena“ mají od „malých písmen“ stále stejný odstup.

V příkazu `\fotenc:lccodes` je potřeba dodržovat následující pravidlo pro nastavení `\catcode`: změníme kategorii 12 na 11 právě tehdy, když je pro daný znak nastaveno nenulové `\lccode`. Má-li znak kategorii 13 (je aktivní) nebo jakoukoli jinou kategorii, necháme ji nezměněnu. Jde o to, že před zavoláním `\fotenc:lccodes` je zavoláno příkazem `\setlang` případné nastavení vstupního překódování balíčkem IENC. Tento balíček, pokud nemá k dispozici encTEX a pokud musí skutečně provést překódování vstupu na odlišné interní hodnoty, nastaví potřebným znakům aktivní kategorii. Tuto kategorii tedy musíme v příkazu `\fotenc:lccodes` respektovat a nesmíme ji měnit. Připravená makra `\lccodes` a `\lccodesloop` toto pravidlo dodržují.

Při definici příkazu `\lccodesback` můžeme použít následující příkazy:

`\zerolccodes`
- `\zerolccodes`⟨*znak1*⟩⟨*znak2*⟩ – nastaví všem znakům v rozsahu ⟨*znak1*⟩ až ⟨*znak2*⟩ nulový `\lccode` a `\uccode`. Pokud má znak kategorii 11, nastaví mu kategorii 12.

`\plainlccodes`
- `\plainlccodes` – nastaví `\lccode`, `\uccode` znakům v rozsahu `^^00` až `^^7f` podle hodnot obvyklých pro plainTEX. Kategorii nastaví na 12 u potřebných znaků (podle plainTEXu) jen tehdy, když původní kategorie znaku je 11.

Nyní vysvětlíme, jak se mění nastavení `\lccode` atd. při opouštění skupin nebo při nastavení nového kódování příkazem `\setlang`. Tento mechanismus je společný s nastavováním překódovacích tabulek balíčkem IENC. Tyto tabulky jsme občas nuceni

nastavit jen globálně (v encTEXu to ani nijak nejde), takže jsem se rozhodl i `\lccode` atd. nastavit globálně a ošetřit přechody mezi skupinami následujícím trikem.

`\gfotenc` Příkaz `\setlang` při změně kódování globálně definuje makro `\gfotenc` hodnotou nového kódování. Makro `\fotenc` je definováno stejně, ale jenom lokálně. Dále `\setlang` provede `\aftergroup\resetcodes`.

`\resetcodes` Makro `\resetcodes` při svém spuštění na konci skupiny provede následující činnost:

- Je-li `\fotenc` rovno `\gfotenc`, nedělá nic a končí.
- Spustí `\lccodesback` (s přechodným `\globaldefs=1`).
- Je-li použit balíček IENC, spustí `\unsetienc [\ienc/\gfotenc]`.
- Je-li použit balíček IENC, spustí `\setienc [\ienc/\fotenc]`.
- Spustí `\fotenc:lccodes` (s přechodným `\globaldefs=1`).
- Provede `\global\let\gfotenc=\fotenc`.

Uvažujme příklad:

```
\setlang [/A]   % provede \aftergroup\resetcodes, toto \resetcodes
                % se nikdy nespustí, protože nejsme uvnitř skupiny.
                % nastaví IENC podle A,
                % spustí  \A:lccodes, které (mj.) definuje \lccodesback
                % je \fotenc=A, \gfotenc=A
{  ...          % je stále \fotenc=A, \gfotenc=A
  \setlang [/B] % provede \aftergroup\resetcodes
                % spustí \lccodesback definované předchozím \A:lccodes
                % ukončí tabulku IENC podle A a nastaví ji podle B
                % spustí \B:lccodes, které (mj.) definuje \lccodesback
  ...           % je \fotenc=B, \gfotenc=B
  \setlang [/C] % provede \aftergroup\resetcodes
                % spustí \lccodesback definované předchozím \B:lccodes
                % ukončí tabulku IENC podle B a nastaví ji podle C
                % spustí \C:lccodes, které (mj.) definuje \lccodesback
  ...           % je \fotenc=C, \gfotenc=C
}               % je \fotenc=A, \gfotenc=C
                % první \resetcodes (z \aftergroup\resetcodes) provede:
                %   spustí \lccodesback definované předchozím \C:lccodes
                %   ukončí tabulku IENC podle C a nastaví ji podle A
                %   spustí \A:lccodes, které (mj.) definuje \lccodesback
                %   nastaví \gfotenc=A
                % druhé \resetcodes neprovede nic, protože už je
                % \fotenc=A, \gfotenc=A
```

Z uživatelského pohledu se tedy kódy nastavují lokálně, ačkoli ve skutečnosti jsou nastaveny globálně a v makrech se staráme o jejich správné přenastavení.

`\langisused` V popisu činnosti makra `\resetcodes` jsme ještě nezmínili následující vlastnost: pokud má makro `\langisused` význam `\undefined`, pak `\resetcodes` zavolá `\setlang[/]`. První zavolání makra `\setlang` nastaví lokálně pro sekvenci `\langisused` význam `\relax`, takže k výše zmíněné podmínce dojde jen tehdy, když první použití příkazu `\setlang` bylo provedeno ve skupině a TEX je zrovna na konci této skupiny. V této situaci je už velmi obtížné se vrátit k původním makrům z `plain.tex`, neboť kódovací soubory byly přečteny s globálními definicemi. Zůstaneme tedy raději u výchozího kódování `8t` a jazyka `none`. To v tuto chvíli zařídí právě příkaz `\setlang[/]`.

3. Pravidla vytváření podpůrných souborů pro LANG

3.1. Pravidla pro soubory `hyph-`⟨*jazyk*⟩`.tex`

Původně jsem počítal s tím, že LANG bude číst soubory `ofs-`⟨*kódování*⟩`.tex` ještě před načtením souboru `hyph-`⟨*jazyk*⟩`.tex`. V takovém případě by stačilo používat pro speciální znaky jazyka na kódování nezávislé TEXové sekvence. Podobně je to uděláno například ve formátu `csplain`. Ukazuje se ale, že i po opuštění skupiny TEX nechává v paměti všechny použité „multiletter control sequences“, přičemž při načítání souborů `ofs-`⟨*kódování*⟩`.tex` všech možných jazyků světa se těchto „multiletter control sequences“ může vytvořit docela dost. Přitom ve vlastním dokumentu větší část těchto kontrolních sekvencí nepoužijeme (protože asi nebudeme používat všechny jazyky). Rozhodl jsem se tedy šetřit paměť TEXu a soubory `ofs-`⟨*kódování*⟩`.tex` v době iniTEXu nenačítat. Soubory `hyph-`⟨*jazyk*⟩`.tex` se tak stanou méně přehledné, ale to snad nevadí, protože se do nich stejně nikdo nedívá.

`\declarehyphentable` Soubory `hyph-`⟨*jazyk*⟩`.tex` by měly někde na začátku obsahovat příkaz

```
\declarehyphentable [⟨jazyk⟩/⟨kódování⟩] % for LANG package
```

kde ⟨*jazyk*⟩ je zkratka jazyka daných vzorů dělení a ⟨*kódování*⟩ je kódování, ve kterém jsou tyto vzory dělení napsány. Příkaz `\declarehyphentable` zkontroluje, zda je aktuální hodnota `\fotenc` rovna ⟨*kódování*⟩. Pokud ano, pak nastaví všem znakům s kódy 128–255 `\lccode` rovno kódu znaku. Pokud ne, pak zavolá soubor `h`⟨*kódování*⟩`-\fotenc.tex`, ve kterém se předpokládá nastavení `\lccode` znaku podle ⟨*kódování*⟩ na hodnotu podle `\fotenc`. Tím je zaručeno správné překódování obsahu `\patterns` a `\hyphenation` na kódování podle `\fotenc`, protože zmíněné TEXové primitivy konvertují všechna data vzorů dělení podle aktuální hodnoty `\lccode`.

V datech pro `\patterns` a `\hyphenation` je možné psát přímo osmibitové znaky podle deklarovaného ⟨*kódování*⟩. Protože ale můžeme mít obavy, že nebude v době iniTEXu nastaven `xord/xchr` vektor na identické zobrazení, je lepší asi psát osmibitové znaky v `^^ab` notaci.

V souborech ⟨*hyph*⟩ můžeme pro své potřeby provádět lokální nastavení a definice. Balíček LANG po načtení vzorů tato nastavení zapomene.

3.2. Problém přepínání `\lccode` uvnitř odstavce

Při hledání míst vzorů dělení slov v odstavci TEX konvertuje text podle hodnot `\lccode` v době, kdy je odstavec kompletně načten. Pokud v době čtení odstavce byly tyto hodnoty měněny, má nakonec vliv jen jejich poslední stav na konci odstavce. Můžeme to považovat za chybu TEXu, protože přepínání vzorů dělení slov a hodnot `\lefthyphenmin`, `\righthyphenmin` je uvnitř odstavce umožněno (do textu odstavce se vkládají značky, podle kterých se TEX v době vyhledávání míst dělení řídí), zatímco přepínání hodnot `\lccode`, které se vzory dělení také úzce souvisí, naráží na problémy. Autor zřejmě implementoval do TEXu přepínání vzorů a zapomněl na přepínání hodnot `\lccode`.

Je potřeba si uvědomit, že výše uvedená chyba se pravděpodobně projeví ve velmi malém množství případů. Často můžeme měnit vzory dělení, a přitom vůbec neměnit hodnoty `\lccode`. Pokud měníme i tyto hodnoty, pak většina textu v odstavci je stejně napsaná malými písmeny, pro která se hodnoty `\lccode` nemění. Velkými písmeny jsou vesměs jen zkratky, které nesmějí být děleny vůbec. Dále velkými písmeny začínají začátky vět a slov, ovšem první písmeno má pro vyhodnocení míst vzorů dělení malý vliv,

neboť je skryto v oblasti `\lefthyphenmin`. Velká písmena s akcenty nebo nelatinkové abecedy se navíc vyskytují jen zřídka. Netvrdím, že k projevu chyby nikdy nedojde, jen se domnívám, že to nemusí být tak časté.

Pokud chceme chybu zcela odstranit, musíme použít e-TEX. Ten dovolí přepínat hodnoty `\lccode` znovu v době hledání míst dělení slov v odstavci. Bohužel, tato možnost je naprogramována poněkud nekoncepčně: využijí se při tomto přepínání hodnoty, kterými byly naplněny registry `\lccode` v době generování formátu při činnosti příkazu `\pattern`, a nikoli v době, kdy byl postupně načítán text odstavce. Sada hodnot `\lccode` se tak stává nedílnou součástí tabulky vzorů dělení slov. To nám mírně komplikuje život, protože jsme `\lccode` využili pro konverzi na různá kódování právě v době činnosti `\patterns` (viz předchozí sekce). Balíček LANG má, nicméně, pro tento problém řešení: pokud je použit e-TEX, pak lokálně předefinuje na začátku načítání vzorů dělení slov primitiv `\patterns` zhruba takto:

```
\let\oripatterns=\patterns
\def\patterns #1{\lowercase{\resetlccodes\oripatterns{#1}}}
```

kde `\resetlccodes` provede příkaz ⟨*kódování*⟩`:lccodes`, který si LANG při aktivním e-TEXu načte ze souboru `ofs-`⟨*kódování*⟩`.tex`. Abychom šetřili pamětí TEXu a nenačítali množství nových kontrolních sekvencí z tohoto souboru už při generování formátu, je nutné umístit `\modifydef` příkazu ⟨*kódování*⟩`:lccodes` hned na začátek tohoto souboru. LANG si lokálně předefinuje při načítání vzorů dělení příkaz `\modifydef` tak, že za ním spustí `\endinput`. Takže se ostatní část souboru vůbec nečte.

3.3. Stylové soubory, pravidla definování maker pro jazyky

V současných TEXových distribucích jsou stylové soubory pro jazyky definovány různě. Jak šel čas, nabalovaly se další myšlenky a některé původní přístupy se ponechávaly jen kvůli zpětné kompatibilitě. V případě balíčku LANG máme výhodu, že můžeme začít „na zelené louce“ a dělat pro všechny jazyky věci pořádně a s jednotným přístupem. Z tohoto pohledu se jeví vhodné všechny stylové soubory pro jazyky přepsat a nazvat je například jménem `lang-`⟨*zkratka jazyka*⟩`.tex`.

Při psaní stylového souboru pro LANG musíme mít na paměti, že je čten až v době prvního přepnutí do deklarovaného jazyka. Přitom režim čtení je nastaven tak, že probíhá uvnitř skupiny s `\endlinechar=-1` a `\globaldefs=1`.

Nastavení `\endlinechar=-1` způsobí ignorování prázdných řádků a token procesor navíc nevytváří mezery na koncích řádků. Důvod tohoto opatření je zřejmý: není totiž žádoucí, aby se při načtení stylového souboru do sazby vloudila mezera (nebo dokonce konec odstavce), protože je možné, že bude stylový soubor načten v horizontálním módu.

Po napsání stylového souboru je tedy potřeba překontrolovat, zda jeho přečtením nevygenerujeme nežádoucí mezery v horizontálním módu. Díky `\endlinechar=-1` je to méně pravděpodobné, než obvykle, ale bohužel to není vyloučené. Příklad:

```
\langdef \makro {...} % komentar
```

Zde mezera za pravou závorkou „`}`“ zůstává a to je špatně. Musíme tedy zcela odstranit „`% komentar`“ nebo přilepit znak procenta těsně za ukončovací závorku definice.

Díky `\globaldefs=1` jsou všechny definice na vnější úrovni stylového souboru globální, tj. efekt je stejný, jako by byl stylový soubor přečten na začátku dokumentu, kdy ještě není otevřena žádná skupina. Bohužel, při `\globaldefs=1` není možno ve stylových souborech přechodně nastavovat `\catcode` uvnitř „pracovních“ skupin. Jinými slovy následující příklad je špatně:

```
{\catcode`\*=13 \gdef *{...}}
```

Zde se nevrátí kategorie hvězdičky po ukončení skupiny do původního stavu. Můžeme sice vrátit kategorii hvězdičky „ručně“ příkazem `\catcode`*=12`, ale ani to není korektní, protože nemáme jistotu, zda v době, kdy je náš stylový soubor přečten, není náhodou uživatelem nastavena kategorie hvězdičky na jinou hodnotu než 12.

Jediná správná možnost, jak pracovat ve stylových souborech se změnami kategorií, je používat připravená makra `\activedef`, `\maybeactive`. Nebo je možno použít následující postup:

```
\globaldefs=0  % toto přiřazení se provede globálně, ale to nevadí
{\catcode`\...  \gdef... }
\globaldefs=1  % toto přiřazení je lokální, což potřebujeme
... další definice jsou automaticky globální
```

Při načtení stylového souboru provede LANG přesně tuto činnost:

- Pokud už byl ⟨*stylový-soubor*⟩ načten, neprovede nic.
- Jinak založí novou skupinu (`\bgroup`).
- `\plaincatcodes` Zavolá makro `\plaincatcodes`, které lokálně nastaví kategorie všech ASCII znaků na „normální hodnoty“ podle plainTEXu. Dále nastaví `\catcode`\@=11`.
- Provede `\endlinechar=-1`.
- Provede `\globaldefs=1`.
- Spustí `\input` ⟨*stylový-soubor*⟩.
- Ukončí skupinu (`\egroup`). Tím se vrátí `\globaldefs=0` a navíc se uvedou hodnoty kategorií znaků do stavu, jak je měl před `\bgroup` nastaveny uživatel.

Vidíme tedy, že ve stylovém souboru máme zaručeny „normální“ kategorie jako v plainTEXu (plus kategorie `@` nastavenu na 11) nezávisle na tom, jaké kategorie má v době volání souboru nastaveny uživatel.

`\ofsinput` Poznamenejme, že výše popsaný postup čtení je implementován v makru `\ofsinput` z OFS. Rovněž zde zmíněné makro `\plaincatcodes` je definováno v OFS. Tato makra jsou využita i při čtení kódovacích souborů typu `ofs-`⟨*kódování*⟩`.tex`.

Kódování stylového souboru by mělo být jen ASCII. Chceme-li pracovat se znaky, které jsou v deklarovaném jazyku běžné, ale nejsou v ASCII, pak musíme použít TEXové sekvence, které jsou definovány v souboru `ofs-`⟨*kódování*⟩`.tex`. Zaručeně budou fungovat všechny takové sekvence, které jsou deklarovány současně ve všech souborech `ofs-`⟨*kódování*⟩`.tex` pro všechna ⟨*kódování*⟩ ze ⟨*seznamu-kódování*⟩ daného jazyka. Připomínám, že ⟨*seznam-kódování*⟩ byl deklarován příkazem `\declarelang` v souboru `langdef.tex`.

Prvním příkazem ve stylovém souboru by měl být:

`\def\langdeflist {`⟨*seznam zkratek jazyků*⟩`}`

abychom nemuseli psát kulaté závorky ke každému `\langdef`.

`\langinfo` Dále by mělo být pomocí `\langdef` definováno makro `\langinfo`, které expanduje na plný název deklarovaného jazyka. Obsahuje-li ⟨*seznam zkratek jazyků*⟩ více než jeden jazyk, měli bychom definovat `\langinfo` pro každý jazyk samostatně za použití kulaté závorky.

`\whichdialect` Obsahuje-li jazyk více nářečí, pak je nutné definovat makro `\whichdialect` následujícím způsobem:

```
\langdef \whichdialect {\ifcase\currentdialect default\or
```

⟨*nářečí 1*⟩`\or` ⟨*nářečí 2*⟩`\or ... \or` ⟨*nářečí n*⟩`\fi}`

Neobsahuje-li jazyk nářečí, pak makro `\whichdialect` vůbec nedefinujeme. V balíčku LANG je řečeno `\langdef \whichdialect (*) {none}`, takže máme zaručeno, že `\whichdialect` expanduje na „none“ pro všechny jazyky, které si nedeklarují vlastní podobu tohoto makra.

V dalších makrech můžeme používat podmínky `\ifnum` ⟨*číslo*⟩`=\currentdialect` nebo `\ifcase\currentdialect`, abychom makra větvili podle nářečí, které si uživatel nastavil příkazem `\setdialect`. Příklad použití `\currentdialect` najdeme například v souboru `lang-de.tex`, kde `\today` používá pro leden slovo „Januar“, ale pokud je nastaveno nářečí `austrian`, pak se leden vypisuje jako „Jänner“.

`\langinit` V makru `\langinit` bychom měli soustředit výchozí nastavení, protože toto makro se spouští při každém přepnutí do jazyka příkazem `\setlang`. Makro `\langinit` definujeme pomocí `\langdef`. Může zde být nastavení mezerování (např. `\frenchspacing`) a nastavení hodnot `\lefthyphenmin` a `\righthyphenmin`. O tomto nastavení bychom měli informovat uživatele v log souboru příkazem `\langmessage`. Dále je nutné uvnitř makra `\langinit` pamatovat na navrátila a příkazem `\afterlang` deklarovat postup, jak vrátit nastavení do stavu podle plainTEXu.

Pokud si vystačíme pro deklarovaný jazyk s implicitním nastavením parametrů sazby podle plainTEXu, nemusíme ve stylovém souboru makro `\langinit` definovat.

`\today` Předpokládá se, že každý stylový soubor definuje příkazem `\langdef` makro `\today`, které expanduje na zápis aktuálního data v příslušném jazyce. Dále je potřeba definovat `\langphrases` příkazem `\langdef` makro `\langphrases`, které sadou příkazů `\def` definuje fráze pro název kapitoly, seznam obrázků atd. podle použitého jazyka. Seznam maker, které se zde mají definovat, je uveden v sekci … V makru `\langphrases` můžeme například použít větvení podle hodnoty `\currentdialect`, aby mohl uživatel mít přístup k různým variantám frází v závislosti na nastaveném ⟨*nářečí*⟩.

Ve stylovém souboru bychom neměli v žádném případě nastavovat znakům aktivní `\activechar` kategorii. Místo toho bychom měli příkazem `\langdef` definovat makro `\activechar` s jedním parametrem zhruba takto:

```
\langdef \activechar #1{%
   \if\string#1⟨znak1⟩\langmessage
      {Character ⟨znak1⟩ is activated as ...}%
      \activedef ⟨znak1⟩{...}\else
   \if\string#1⟨znak2⟩\langmessage
      {Character ⟨znak2⟩ is activated as ...}%
      \activedef ⟨znak2⟩{...}\else
   ...
   \if\string#1⟨znak n⟩\langmessage
      {Character ⟨znak n⟩ is activated as ...}%
      \activedef ⟨znak n⟩{...}\else
   \activecharwarn#1\fi...\fi\fi}
```

Tím dáváme uživateli možnost, aby mohl příkazem `\activechar`⟨*znak*⟩ nastavit aktivní kategorii jen těm znakům, které uzná za vhodné. Typickou deklarací aktivního znaku v makru `\activechar` je:

```
...\if\string#1⟨znak⟩\langmessage
      {Character ⟨znak⟩ is activated as expandable double-char prefix.}
      \activedef ⟨znak⟩{\expdoublechar⟨znak⟩}\else ...
```

a dále pomocí sady příkazů `\doublechardef`⟨*znak*⟩⟨*znak2*⟩`...` definujeme tabulku maker, která se po `\activechar`⟨*znak*⟩ začne v závislosti na aktuálním jazyku používat. Podrobněji je tato metoda popsána v sekci...

Za posledním `\else` v makru `\activechar` píšeme `\activecharwarn#1`, což vyvolá tisk varování o tom, že `#1` nemá v aktuálním jazyku připravenu deklaraci jako aktivní znak.

Pokud definujeme ve stylovém souboru vlastní pomocná makra, pak by měla mít identifikátor ve tvaru \⟨*zkratka jazyka*⟩`@`⟨*identifikátor*⟩, například `\cz@tryemdash`. Tím snad zabráníme konfliktu s názvy maker, která si definuje uživatel nebo jsou použita v jiném stylovém souboru.

Za `\endinput` by měla ve stylovém souboru pokračovat dokumentace stylového souboru. Ta je čtena příkazem `\printlangdoc` od sekvence `\langdocbegin`. Podrobněji o možnosti psaní dokumentace ke stylovým souborům rovnou ve stylových souborech pojednává sekce...

Při psaní stylových souborů se můžeme inspirovat existujícími stylovými soubory `lang-cz.tex`, `lang-de.tex`, `l-czsk.tex` a dalšími.

3.4. Dokumentování stylových souborů

`\printlangdoc` Příkaz `\printlangdoc` nastaví kategorie znaků podle plainTEXu (použije k tomu příkaz `\plaincatcodes` z OFS) a přečte soubor maker `l-doc.tex`. Tento soubor definuje několik maker, která se dají používat při psaní dokumentace (viz níže) a nastaví znaky `"` a `<` jako aktivní. Pak příkaz `\printlangdoc` přečte stylový soubor aktuálního jazyka. Při čtení souboru se přeskočí balancovaný text od začátku souboru až po sekvenci `\langdocbegin` `\langdocbegin`. Za touto sekvencí by měl následovat vlastní text dokumentace.

V souboru `l-doc.tex` je obsaženo i jednoduché nastavení sazby: `\parindent`, `\parskip`, `\emergencystretch` a dále je spuštěn příkaz `\english`, protože se předpokládá, že dokumentace bude psaná v anglickém jazyce.

Po přečtení a vytištění dokumentace `\printlangdoc` nevrací nastavení a makra do původního stavu. Pouze si poznamená, že už byl soubor `l-doc.tex` přečten a při dalším spuštění příkazu `\printlangdoc` jej nečte znova. Nepředpokládáme totiž, že bude příkaz `\printlangdoc` zařazen do obyčejného dokumentu, ale na druhou stranu počítáme s tím, že může následovat přepínač do dalšího jazyka následovaný dalším příkazem `\printlangdoc`. Tím můžeme získat příručku o jazycích, které nás zajímají.

Pokud se po použití příkazu `\printlangdoc` chceme zbavit nastavení, které bylo provedeno v souboru `l-doc.tex`, pak stačí uzavřít příkaz do skupiny. Při opětovném použití příkazu `\printlangdoc` bude v takovém případě načítán soubor `l-doc.tex` znova.

V dalším textu uvádím makra, která nejsou definována v balíčku LANG obecně, ale jsou k dispozici jen v souboru `l-doc.tex` a mají umožnit snadnější psaní dokumentace.

`\title` Makro `\title` ⟨*text*⟩⟨*prázdný řádek*⟩ vytiskne ⟨*text*⟩ jako nadpis. Měl by zde být uveden plný název jazyka a jeho zkratka.

`\author` Makro `\author` ⟨*text*⟩⟨*prázdný řádek*⟩ vytiskne slovo Author: následované ⟨*textem*⟩.

Text mezi počítačovými uvozovkami: `"takto psaný"` bude tištěn strojopisem ve verbatim režimu, tj. všechny znaky včetně zpětného lomítka se tisknou tak, jak jsou. Makro vytvoří nezlomitelný box s textem a je určeno pro úseky verbatim textu uvnitř odstavce (například při zmiňování kontrolních sekvencí). Mezi `"..."` je přechodně aktivní vykřičník, který je definován jako `\string`. Takže `!"` mezi `"..."` vytiskne počítačovou uvozovku a `!!` vytiskne jeden vykřičník.

Text mezi skobatými závorkami `<takto psaný>` se promění na nezlomitelný symbol ⟨*takto psaný*⟩. Tato vlastnost je implementována pomocí aktivního znaku „<“, který zůstává aktivní i mezi počítačovými uvozovkami. Takže i verbatim text mezi počítačovými uvozovkami je citlivý na výskyt znaku „<“.

\begtt ...\endtt

„Display verbatim režim“ je možný pomocí dvojice `\begtt`...`\endtt`. Například

```
\begtt
tady je text včetně \kontrolních \sekvencí, který bude
vytištěn ve verbatim režimu se stejným členěním na řádky.
Pouze otevírací lomená závorka zůstává aktivní
a vytváří ⟨takové věci⟩.
\endtt
```

Pokud za `\endtt` není prázdný řádek, bude se následující odstavec tisknout bez odsazení. Mezi `\begtt` a `\endtt` má vykřičník kategorii nula, takže je možno psát kontrolní sekvence `!takto`. Přitom makro `\!` je definováno jako obyčejný vykřičník, takže `!!` vytiskne jeden obyčejný vykřičník.

\begitems
\enditems

Pro výčet údajů s odrážkami lze použít dvojici `\begitems`...`\enditems`. Mezi touto dvojicí je přechodně aktivní znak `*`, který ukončuje předchozí údaj a zahajuje nový údaj včetně puntíku na začátku. Příklad:

```
\begitems
* první položka
* druhá položka
* poslední položka
\enditems
```

Jinak je možné použít všechna makra z plainTEXu.

4. Licence

...

5. Rejstřík

Rejstřík obsahuje odkazy pouze na místa v textu, kde je k makru vysvětlující komentář. V tom místě je též uveden pro snadnější orientaci název makra v levém okraji. V rejstříku nejsou odkazy na všechny zmínky o makru v textu.

6. Literatura

[1] Karl Berry: Fontname – Filenames for TeX fonts, `fontname.pdf`