

EncT_EX

možnost překódování vstupu v T_EXu

Petr Olšák

www.olsak.net/encdex.html

Toto je dokumentace k verzi Feb. 2003 a Jun. 2004

EncTeX je volné programové vybavení; můžete jej dále šířit a modifikovat podle podmínek „GNU General Public License“, kterou publikovala Free Software Foundation; použijte verzi 2 této licence nebo (podle Vaší volby) libovolnou pozdější verzi.

Balíček najdete na Internetu na <ftp://math.feld.cvut.cz/pub/olsak/enc tex/>.

Tento balíček je rozšiřován v naději, že bude užitečný, avšak BEZJAKÉKOLI ZÁRUKY; neposkytují se ani odvozené záruky PRODEJNOSTI anebo VHODNOSTI PRO URČITÝ ÚČEL. Další podrobnosti hledejte v Obecné veřejné licenci GNU.

Kopii „GNU General Public License“ jste měl obdržet spolu s tímto programem; pokud se tak nestalo, napište o ni Free Software Foundation, Inc., 675 Mass Ave, Cambridge, MA 02139, USA. Český překlad této licence najdete na <http://www.gnu.cz/gplcz.html>.

© 1997, 2002, 2003, 2004 RNDr. Petr Olšák

TeX je ochranná známka American Mathematical Society.

Autor TeXu je profesor Donald Knuth. TeX je volné programové vybavení se specifickou licencí, viz dokumentaci k tomuto programu.

1. Základní informace

Balík `encTeX` je jednoduché rozšíření `TeX`u pro takové implementace, ve kterých se `TeX` instaluje ze zdrojového kódu `tex.web`. Tuto podmínku například splňuje implementace `web2c`, určená pro UNIXy a jiné operační systémy s kvalitním překladačem jazyka C.

Rozšíření `encTeX` je zpětně kompatibilní s originálním `TeX`em. Přidává deset nových primitivů, kterými lze číst nebo naplňovat vnitřní kódovací tabulky, podle nichž je znak transformován na úrovni vstupního procesoru `TeX`u nebo při výstupu na terminál, do `log` souboru a `\write` souborů. Tyto tabulky se ukládají do formátových souborů, takže po načtení formátu se inicializují ve stejném stavu, v jakém byly v okamžiku příkazu `\dump`.

Změna `TeX`u je důkladně testovaná a prošla též testem TRIP s těmito dvěma odlišnostmi:

- Odlišný banner
- Počet „multiletter control sequences“ je o deset větší.

1.1. Instalace

Viz soubor `INSTALL`.

1.2. Verze

V roce 1997 byla zveřejněna první verze `encTeX`u, která umožňovala konverze pouze v režimu „byte“ a nastavovala tisknutelnost znaků (primitivy `\xordcode`, `\xchrcode`, `\xprncode`).

V roce 1998 obnovil někdo z Polska používání TCX tabulek ve `web2c` `TeX`u. Protože `encTeX` z roku 1997 dělal víceméně totéž a používání obou nadstaveb současně by mohlo být pro mnoho uživatelů matoucí, přestal jsem `encTeX` prosazovat. Ovšem v roce 2002 jsem se rozhodl přidat do `encTeX`u podporu UTF-8 kódování, protože tato podpora není a asi nebude v TCX tabulkách implementována. V prosinci 2002 jsem se tedy rozhodl znovu začít propagovat `encTeX`, který navíc zvládne UTF-8 kódování.

Verze `encTeX`u Feb. 2003 přidává dalších 7 primitivů `\mubyte`, `\endmubyte`, `\mubytein`, `\mubyteout`, `\mubytelog`, `\specialout` a `\noconvert`. To umožní definovat vstup pro UTF-8 kódované soubory. Tato verze je samozřejmě zpětně kompatibilní s původní verzí `encTeX`u z roku 1997. Neplánuji do budoucna žádné rozsáhlejší změny. Pokud přeci jenom k nějakým změnám dojde, pak vždy budou zpětně kompatibilní se stávajícím řešením.

Verze Jun. 2004 opravuje jen některé drobné chyby a nepřidává žádné nové funkce (podrobněji viz soubor `enctex.patch-jun2004`).

1.3. EncTeX ve web2c distribuci

`EncTeX` se v `TeX`u, `pdfTeX`u, `e-TeX`u a `pdfeTeX`u v distribuci `web2c` inicializuje pomocí přepínače `-enc` na příkazové řádce. Tento přepínač je nutno použít při inicializaci formátu. V takovém případě `encTeX` uloží své primitivy a svá data do formátu. Při použití formátu už není nutné přepínač `-enc` psát znova. `EncTeX` se v tomto případě inicializuje z formátu a napíše o tom zprávu na terminál a do `log` souboru. Pokud nepoužijete přepínač `-enc` v době generování formátu, ale použijete jej až při použití formátu, pak `TeX` vypíše varování o neexistenci `encTeX`u ve formátu a přepínač `-enc` je ignorován.

Ve `web2c` distribuci pracují se stejnými kódovacími vektory `xord` a `xchr` (pro překódování byte na byte) jednak `encTeX` a jednak tzv. TCX tabulky (při použití přepínače `-translate-file`). Určitému konfliktu se tedy při současném používání obou rozšíření nevyhneme. Je-li v `encTeX`u použita TCX tabulka, pak při startu `TeX`u přepíše hodnoty `\xordcode`, `\xchrcode` a `\xprncode` podle sebe. Znamená to, že při použití `iniTeX`u pak nemusejí platit implicitní hodnoty těchto kódů dokumentované v sekci 2.2 a při použití formátu s `encTeX`em jsou hodnoty těchto kódů, které si do formátu uložil `encTeX`, přepsány podle TCX tabulky. O této skutečnosti `encTeX` náležitě informuje na terminálu a v `log`u. Při běhu programu pak můžete `encTeX`ovými primitivy tyto kódy dále číst a měnit. Použití TCX tabulky tedy nevylučuje možnosti `encTeX`u.

1.4. Problém s licenci TeXu

`EncTeX` rozšiřuje `TeX` o nové primitivy, takže bychom neměli tomuto programu říkat `TeX`. Na druhé straně ale Knuth samotný předpokládá, že vnitřnosti `TeX`u budou odstíněny od prostředí operačního systému. Proto implementoval `xord` a `xchr` vektory. V `encTeX`u můžeme nastavit podle zvyklostí operačního systému vstupní a výstupní překódovací tabulky a pak nastavit všem novým primitivám význam

`\undefined`. Dále se bude `TeX` modifikovaný `encTeX` chovat stejně, jako originální `TeX`. Navíc můžeme překódovací tabulky nastavit při generování formátu a v produkční verzi `TeXu` zakázat přístup k primitivům. Produkční verze `TeXu` se pak chová zcela stejně jako originální `TeX`. Knuth předpokládá, že odstínění od prostředí operačního systému se provede vždy při kompilaci zdrojového kódu `TeXu`, zatímco `encTeX` umožňuje tuto otázku řešit později, například v době generování formátu. Umožnění úpravy některých (například paměťových) parametrů až za běhu `TeXu` také není nic nového a známe to skoro u všech distribucí `TeXu`.

Domnívám se, že druhý řádek na terminálu a v logu dostatečně informuje o tom, že se jedná o modifikovanou verzi `TeXu`. Také se domnívám, že pokud se velmi rozšíří kódování UTF-8, pak není zbylí a takové konverze jsou v 8 bitové verzi `TeXu` nezbytné.

Je důležité rovněž připomenout, že implicitní chování `encTeXu` je takové, že pokud se nepoužijí rozšířené primitivy `encTeXu`, pak se chová naprosto stejně jako originální `TeX`.

Podle mého názoru novější implementace `web2c TeXu` taky není v přísném slova smyslu `TeX`. Umožňuje totiž změnu chování programu, pokud na prvním řádku dokumentu za znaky `%&` je cosi specifického napsáno. To je podle mého názoru větší přestupek oproti licenci `TeXu`, než `encTeX`ovými primitivy nastavit ve formátu prostředí systému a pak tyto primitivy v produkční verzi `TeXu` zakázat.

2. Překódování byte na byte pomocí vektorů `xord`, `xchr`

2.1. Vektory `xord` a `xchr`

Vektory `xord` a `xchr` mají velikost 255 bytů a obsahují informaci o překódování znaku vstupujícího do `TeXu` nebo vystupujícího na terminál a do textových souborů. Jedná se o pole vestavěná do programu, přes která jsou filtrovány veškeré textové vstupní a výstupní informace. Má-li znak na vstupu kód x a chceme, aby měl uvnitř `TeXu` kód y , pak musí být nastaven vektor `xord` tak, aby `xord[x]=y`. Při zpětném výstupu znaku na terminál, do logu a do souborů zpracovávaných pomocí `\write` platí tato pravidla: Není-li znak s kódem y označen jako „tisknutelný“, pak vystupuje pomocí přepisu `^^kód y`. Je-li tisknutelný, pak vystupuje s kódem `z=xchr[y]`.

Standardně bývají v systémech s kódem ASCII nastaveny hodnoty těchto vektorů tak, že

`xord[i]=xchr[i]=i` pro všechna i v rozsahu 0 až 255.

Na systémech, které nepoužívají ASCII, se může mapovat 94 tisknutelných ASCII znaků jinam. Mimo to je deklarovaná vlastnost „tisknutelnosti“ znaku v ASCII takto: Znak je tisknutelný, pokud má kód y v rozsahu 32 až 126. Ostatní znaky se považují za netisknutelné a `TeX` je standardně přepisuje pomocí dvojité stříšky.

Po instalaci balíčku `encTeX` je možno přímo nastavovat a číst obsahy vektorů `xord` a `xchr` prostřednictvím primitiv `\xordcode` a `\xchrcode` a dále nastavovat vlastnost „tisknutelnosti“ znaku pomocí primitivu `\xprncode`. Syntaxe všech tří nových primitiv je naprosto stejná, jakou známe například u primitiv `\lccode` a `\uccode`. Například:

```
\xordcode"AB="CD \xchrcode\xordcode"AB="AB \the\xchrcode200
```

nastavuje `xord[0xAB]=0xCD`; `xchr[xord[0xAB]]=0xAB` a dále vytiskne hodnotu `xchr[200]`.

Na rozdíl od podobných primitiv `\catcode`, `\lccode`, `\sfcode` a dalších však nově zavedené primitivy mají jednu podstatnou výjimku. Reprezentují interní registry `TeXu`, které vždy mají globální platnost. Proto je nastavení `\xordcode` a `\xchrcode` uvnitř skupiny za všech okolností globální, ačkoli to explicitně nepíšeme. Ústupem z požadavku na možnost lokálního deklarování hodnot jsem dosáhl podstatně větší efektivity výsledného kódu programu.

2.2. Tisknutelnost znaků nastavená pomocí `\xprncode`

Primitiv `\xprncode` umožňuje nastavovat vlastnost „tisknutelnosti“ znaku takto: Znak s kódem y je tisknutelný právě tehdy, když je y v rozsahu 32 až 126 nebo je `\xprncode y > 0`. Napíšeme-li například `\xprncode255=1`, bude tisknutelný znak s kódem 255. Na druhé straně, nastavení `\xprncode'a` a třeba na nulu nemá na chování programu žádný vliv, protože kód znaku `a` je v rozsahu 32 až 126. Tímto opatřením program vykazuje určitý pud sebezáchovy, protože zlý uživatel by mu mohl nastavit všechny znaky jako netisknutelné a program by ztratil schopnost se vyjadřovat. Hodnoty `\xprncode` lze nastavit

jako u ostatních nových primitivů v rozsahu nula až 255, ovšem otázka tisknutelnosti je totožná s otázkou na kladnou hodnotu bez ohledu na to, jak velká tato hodnota je.

Výchozí hodnoty pro kódování v době `iniTeXu` jsou následující:

- `\xordcode i = i` pro všechna i v rozsahu 0...255,
- `\xchrcode i = i` pro všechna i v rozsahu 0...255,
- `\xprncode i = 0` pro i v rozsahu 0...31, 127...255,
- `\xprncode i = 1` pro i v rozsahu 32...126.

První dva řádky jsou pravdivé jen na operačních systémech, které přijaly kódování anglické abecedy podle ASCII. Pokud tomu tak není, pak jsou výchozí hodnoty vektorů `xord` a `xchr` pozměněny tak, aby mapovaly tisknutelné znaky podle systému do ASCII uvnitř `TeXu`. Taková změna se týká jen 95 základních tisknutelných znaků, které jsou v ASCII na pozicích 32 až 126.

3. Konverze více bytů na jeden byte nebo kontrolní sekvenci

Od verze Dec 2002 `encTeX` umí také konvertovat na úrovni vstupního procesoru více bytů na jeden byte nebo kontrolní sekvenci. Při výstupu do logu a `\write` souborů je pak tento objekt zpětně převeden na původních více bytů. Tato vlastnost nechce nahradit chybějící interpret regulárních výrazů ve vstupním procesoru `TeXu`. Byla implementována pouze z důvodu umožnit pracovat s UTF-8 kódovanými soubory v běžném 8 bitovém `TeXu` tak, že znaky z UTF-8 z nejčastěji používané abecedy mohou být mapovány na jeden znak, který může mít svůj `\catcode`, `\uccode` atd. Jiné znaky z UTF-8 mohou být mapovány na libovolné kontrolní sekvence.

Pro nastavení takové konverze jsou do `TeXu` přidáno nové pět primitivů: `\mubytein`, `\mubyteout`, `\mubytelog`, `\mubyte` a `\endmubyte`. Primitivy `\mubytein`, `\mubyteout` a `\mubytelog` jsou celočíselné registry implicitně s nulovou hodnotou, tj. konverze vstupu a výstupu podle konverzní tabulky se neprovádějí. Je-li `\mubytein` nastaveno na kladnou hodnotu, `TeX` okamžitě zahájí konverze vstupního řádku podle konverzní tabulky. Je-li `\mubyteout` nastaveno na kladnou hodnotu, `TeX` začne konvertovat do výstupních `\write` souborů podle stejné konverzní tabulky. Při kladném `\mubytelog` bude `TeX` zpětně konvertovat také výstup do logu a na terminál. Implicitně je konverzní tabulka prázdná a jednotlivé řádky se do ní přidávají pomocí dvojice primitivů `\mubyte`, `\endmubyte` s touto syntaxí:

```
\mubyte <first_token><one_optional_space><optional_prefix><byte_sequence>\endmubyte
```

Každá `<byte_sequence>` bude převedena ve vstupním procesoru na `<first_token>`. Je-li `<first_token>` znakem (tj. není to kontrolní sekvence), pak se ignoruje jeho kategorie, protože konverze je prováděna v input procesoru podle schématu: `<byte_sequence>` na jeden `<byte>`. Při výstupu do logu a `\write` souborů se pak každý takový `<byte>` znovu převede na `<byte_sequence>`.

Pokud je `<first_token>` kontrolní sekvence, pak se na úrovni vstupního procesoru promění každá `<byte_sequence>` na tuto kontrolní sekvenci implementovanou ve formě neměnitelného tokenu. Token procesor tuto sekvenci tedy znovu neinterpretuje a zůstává za ní ve stavu neignorování mezer. Při `\mubyteout<2` není při výstupu do `\write` souborů tato kontrolní sekvence zpětně převáděna na původní `<byte_sequence>`, ale podléhá jen běžné expanzi, jako ostatní kontrolní sekvence. Při `\mubyteout>=2` se i tyto kontrolní sekvence převádějí do `\write` souborů na původní `<byte_sequence>`. Aby se ale mohly převést, nesmějí před tím expandovat, tj. musejí mít v době expanze význam neexpandovatelné kontrolní sekvence nebo musejí být označeny pomocí `\noexpand`. Při `\mubyteout>=3` `encTeX` potlačí expanzi kontrolních sekvencí deklarovaných v `\mubyte` automaticky (podrobněji viz sekci 3.8). Výstup do logu a na terminál, který není produktem příkazu `\write`, ponechává i při kladném `\mubytelog` kontrolní sekvence nezměněny.

Význam `<optional_prefix>` je vysvětlen v sekci 3.5.

3.1. Zanášení údajů do konverzní tabulky

Záznamy do konverzní tabulky jsou pomocí primitivů `\mubyte`, `\endmubyte` zanášeny globálně, zatímco hodnoty v registerch `\mubytein`, `\mubyteout` a `\mubytelog` mají obvyklou lokální platnost.

Dvojice primitivů `\mubyte`, `\endmubyte` pracuje analogicky, jako dvojice `\csname`, `\endcsname`. Rozdíl je pouze v tom, že první token `<first_byte>` se neexpanduje a že za ním může (po expanzi) následovat `<one_optional_space>`. Při skenování `<optional_prefix>` a `<byte_sequence>` již probíhá úplná expanze a při

ní se nesmí objevit na vstupu do hlavního procesoru token typu kontrolní sekvence, jinak nastane chyba, kterou už známe z používání `\csname`, `\endcsname`:

```
! Missing \endmubyte inserted.
```

Primitiv `\mubyte` na rozdíl od `\csname` neprovádí činnost na úrovni expand procesoru, ale jedná se o přiřazovací primitiv zpracovaný na úrovni hlavního procesoru. Takže po

```
\edef\A{\mubyte X ABC\endmubyte}
```

bude makro `\A` obsahovat tokeny: `\mubyte X ABC\endmubyte`.

Příklady:

```
\mubyte ^^c1      ^^c3^^81\endmubyte % Ā
\mubyte ^^e1      ^^c3^^a1\endmubyte % á
% atd. -- implementace UTF8
```

```
\mubyte \endash  ^^c4^^f6\endmubyte % příklad na kontrolní sekvenci
\mubyte \integral INT\endmubyte     % příklad pro ilustraci, viz dále.
```

```
\mubytein=1 \mubyteout=1 \mubytelog=1 % od této chvíle je překódování aktivní
```

```
\def\endash {--}
\def\integral {\ifmmode \int\else $\int$\fi}
```

V tomto příkladě je v místě *⟨one optional space⟩* více mezer a tabulátorů. Protože tabulátory mají kategorii mezery, jsou všechny tyto znaky přeměněny token procesorem na jedinou mezeru požadovanou v syntaktickém pravidle pro `\mubyte`, `\endmubyte`.

Po použití definic z příkladu se například slovo `INTEGRAL` promění v token `\integral` okamžitě následovaný písmeny „EGRAL“. V textu `INT EGRAL` bude za tokenem `\integral` mezer a teprve pak písmena „EGRAL“. Také jsou možné konstrukce typu `\defINT{něco}` apod. Když napíšeme `\show INT`, dostaneme odpověď:

```
> \integral=macro:
->\ifmmode \int \else $\int $\fi .
1.18 \show INT
```

a `\string INT` se expanduje na text: `\integral`.

Po deklaraci `INT` podle předchozího příkladu se může stát, že někdo napíše: `\INT`. Správně by to mělo vést na prázdnou kontrolní sekvenci (`\csname\endcsname`) následovanou kontrolní sekvencí `\integral`. Protože se ale s prázdnými kontrolními sekvencemi v \TeX u moc často nepracuje a pro uživatele by to mohlo být matoucí, rozhodl jsem se tuto situaci ošetřit tak, že `\INT` je převedeno pouze na token `\integral`. Pozor na skutečnost, že za sekvencí `\INT` není \TeX ve stavu ignorování mezer a navíc může za ní okamžitě následovat písmeno.

3.2. Vlastnosti konverze

Multibytové sekvence jsou převedeny ze vstupu pouze tehdy, pokud jsou celé obsaženy v jediném řádku. Přesah do dalšího řádku není možný. Připojený `\endlinechar` na konci řádku se může stát předmětem konverze podle konverzní tabulky.

Sekvence `^^c3^^81` se nepromění ani po použití definic z příkladu na byte „Á“, protože převod dvojitých zobáků na jednotlivé byty probíhá v token procesoru, tj. později, než převody více bytů na jeden podle `\mubyte`.

Převod více bytů na jeden byte nebo kontrolní sekvenci probíhá později než konverze podle `\xordcode` a při výstupu do `\write` a log souborů pak převod podle `\mubyte` probíhá dříve než konverze podle `\xchrcode`. *⟨byte_sequence⟩* tedy musí obsahovat sekvenci bytů tak, jak jsou tyto byty konvertovány ze vstupního souboru pomocí `\xordcode`.

Postupné procesy na vstupu a výstupu si můžeme naznačit takto:

```
vstupní text -> \xordcode -> připojení \endlinechar ->
                \mubyte -> token procesor -> expanze ...
argument \write -> expanze -> \mubyte -> \xchrcode -> výstup
```

Při výstupu do `\write` souborů a logů zpětně konvertované *⟨byte_sequence⟩* už nepodléhají další konverzi na formát typu `^^c3^^81` ani opakované konverzi podle `\mubyte`. Tyto *⟨byte_sequence⟩* se pouze převedou podle hodnot `\xchr`.

Pokud se v konverzní tabulce vyskytují *⟨byte_sequence⟩* se společným začátkem, ale různě dlouhé, pak input procesor `encTeXu` (od verze Feb. 2003) převádí vždy podle nejdelší možné *⟨byte_sequence⟩*, která se na vstupu objeví. Příklad:

```
\mubyte X A\endmubyte
\mubyte Y ABC\endmubyte
\mubyte \foo ABCD\endmubyte
```

V tomto příkladě se každé A konvertuje na X, ale pokud následuje BC, pak se skupina ABC konvertuje na Y ovšem s tou výjimkou, že se skupina ABCD konvertuje na `\foo`. Na pořadí vkládání údajů do konverzní tabulky primitivem `\mubyte` v tomto případě nezáleží.

Pokud se v konverzní tabulce objeví dvě stejné *⟨byte_sequence⟩*, pak má přednost ta, která byla zanesena do tabulky později.

3.3. Konverze výstupu do logu a na terminál

Výstup do logu a na terminál je při nulovém `\mubytelog` ponechán bez konverze. Pokud je v takovém případě `\xprncode` znaku nulový, pak se znak vytiskne ve formě `^^A` nebo `^^bc`. Při kladném `\mubytelog` se znaky zanesené do konverzní tabulky konvertují zpět na *⟨byte_sequence⟩*. Do logu a na terminál se ale na *⟨byte_sequence⟩* nikdy nekonvertují kontrolní sekvence.

Zpětná konverze do logu a na terminál je nastavována společně primitivem `\mubytelog` a nelze ji oddělit tak, že by například pro terminál byla zpětná konverze potlačena a pro log soubory nikoli.

Výpisy do logu a na terminál občas obsahují kompletní přepis právě čteného řádku, který je například při hlášení chyby roztržen na dva řádky, aby `TeX` naznačil místo, kde došlo k problému. Pro další výklad budeme těmto částem logu říkat „přepisy řádků“.

Je-li registr `\mubytein` nulový, pak přepisy řádků fungují jako ve standardním `TeXu`. Je-li `\mubytein` kladný a `\mubytelog` nulový, pak je v přepisech řádků zobrazen obsah řádku až po konverzi input procesorem `encTeXu`, tj. v těchto přepisech se mohou například vyskytovat kontrolní sekvence, které tam uživatel vůbec nenapsal. Jsou-li `\mubytein` i `\mubytelog` kladné, pak jsou přepisy řádků realizovány bez multibytové konverze tam ani zpět. Probíhá jen konverze podle `xchr` a `xord` vektorů. Je třeba si uvědomit, že v tomto případě mohou být chybové výpisy poněkud matoucí. Například po

```
\mubyte \sekvence ABC\endmubyte \let\sekvence=\undefined
\mubytein=1 \mubytelog=1
Tady je test ABC a řádek pokačuje.
```

dostáváme na výstupu:

```
! Undefined control sequence.
1.3 Tady je test ABC
          a řádek pokačuje.
?
```

Teprve `\show ABC` odhalí:

```
> \sekvence=undefined.
1.3 \show ABC
```

3.4. Mazání údajů v konverzní tabulce

Údaje v konverzní tabulce lze mazat jen hromadně: příkaz vymaže všechny údaje začínající na společný první znak v *⟨byte_sequenci⟩*. Proveďte se to pomocí příkazu `\mubyte <znak> <znak>\endmubyte`. Například:

```
\mubyte A A\endmubyte
```

odstraní z konverzní tabulky všechny *⟨byte_sequence⟩* začínající písmenem A.

Následující kód promaže celou tabulku:

```
{\catcode'\^^@=12
```

```

\gdef\clearmubytes{\bgroup \count255=1
  \loop \uccode‘X=\count255
    \uppercase{\mubyte XX\endmubyte}%
    \advance\count255 by1
    \ifnum\count255<256 \repeat
  \mubyte ^^@^^\endmubyte
  \egroup}
}
\clearmubytes

```

3.5. Vstupní a výstupní část konverzní tabulky

Konverzní tabulka konstruovaná pomocí `\mubyte`, `\endmubyte` má dvě nezávislé části: vstupní, se kterou pracuje input procesor a výstupní, která se používá při zpětných konverzích. Údaje je možné zanést nezávisle do každé části při použití neprázdného *⟨optional_prefix⟩* (viz syntaktické pravidlo `\mubyte` na začátku této kapitoly). Je-li *⟨optional_prefix⟩* prázdný, pak se požadavek na konverzi zanese dvojmo do vstupní i výstupní části. Je-li ale *⟨optional_prefix⟩* znak kategorie 8 (obvykle znak `_`), pak se údaj zanese jen do vstupní části tabulky. Obsahuje-li *⟨optional_prefix⟩* dvojici znaků kategorie 8 (obvykle tedy `--`), pak se údaj zanese jen do výstupní části tabulky.

Při *⟨optional_prefix⟩* `--` (výstupní část tabulky) je dovoleno mít prázdnou *⟨byte_sequenci⟩*. V takovém případě se původní údaje z výstupní části tabulky, které odpovídají *⟨first_token⟩*, vymažou. Pokud tam žádné takové údaje nebyly, nestane se nic.

Vraťme se ke kódu na vymazání tabulky z předchozí sekce. Tento kód vymaže vše ze vstupní části tabulky a z výstupní jen ty údaje, které jsou vázány na *⟨first_token⟩* ve tvaru *⟨byte⟩*. Údaje z výstupní části vázané na kontrolní sekvence nejsou tímto způsobem promazány. Promazání jednoho údaje uděláme pomocí: `\mubyte \foo --\endmubyte`.

3.6. Vkládání dalších kontrolních sekvencí

Je-li *⟨first_token⟩* ve tvaru kontrolní sekvence a navíc *⟨optional_prefix⟩* je token kategorie 6 (obvykle znak `#`) následovaný *⟨číslem⟩*, pak input procesor ponechá beze změny *⟨číslo⟩* znaků, ale vloží před ně deklarovanou kontrolní sekvenci. Údaj se zanese jen do vstupní části konverzní tabulky. Příklad:

```

\def\abc{ABC}
\mubyte X BC\endmubyte \mubytein=1
\mubyte \foo #3 \abc\endmubyte Nyní ABC přechází na \foo ABC
\mubyte \foo #1 \abc\endmubyte Nyní ABC přechází na \foo AX

```

Parametr *⟨číslo⟩* má stejnou syntaxi, jako gramatická kategorie *⟨number⟩* z `TeXbooku`. Například při přímém zápisu dekadických číslic může následovat jedna nepovinná mezera, jako v předchozí ukázce.

Je-li *⟨číslo⟩* rovno nule, pak bude ponechána beze změny celá *⟨byte_sequence⟩*, která následuje. Výsledek je tedy stejný, jako kdyby *⟨číslo⟩* mělo hodnotu délky *⟨byte_sequence⟩*.

Hodnota *⟨číslo⟩* je akceptována jen v rozsahu 0 až 50. Záporná čísla jsou interpretována stejně jako nula a čísla větší než 50 způsobí, že input procesor vypne po vložení kontrolní sekvence další konverzi až do konce řádku.

Praktický příklad použití:

```

\mubyte \warntwoytes #2^^c3\endmubyte
\mubyte \warntwoytes #2^^c4\endmubyte
\mubyte \warntwoytes #2^^c5\endmubyte
% atd...
\def\warntwoytes #1#2{\message{WARNING: the UTF8 code:
  \noconvert#1\noconvert#2 is not defined i my macros.}}

```

V tomto příkladě byl použit nový primitiv `encTeXu \noconvert`, jehož vlastnosti jsou uvedeny později v kapitole 5. Připomínám také, že od verze Feb. 2003 nezpůsobují jednoznakové *⟨byte_sequence⟩* kolizi s víceznakovými *⟨byte_sequencemi⟩* se stejným začátkem, takže tento příklad neruší konverzi „známých“ UTF-8 kódů.

3.7. Rozpoznání začátku řádku

Existují-li v konverzní tabulce $\langle byte_sequence \rangle$ s prvním znakem shodným s aktuálním $\backslash endlinechar$, tj. $\langle byte_sequence \rangle$ jsou ve tvaru $\langle endlineschar \rangle \langle zbytek \rangle$, pak input procesor navíc ověřuje, zda je $\langle zbytek \rangle$ shodný se začátkem každého řádku. Pokud ano, provede požadovanou konverzi. Příklad použití:

```
\bgroup \uccode'X=\endlinechar \uppercase{\gdef\echar{X}}\egroup
\mubyte \fooB \echar ABC\endmubyte % vyhovuje ABC na začátku řádku
\mubyte \fooE ABC\echar \endmubyte % vyhovuje ABC na konci řádku
\mubyte \fooW \spce\space ABC\space \endmubyte
% vyhovuje ABC jako slovo s mezerami vpředu i vzadu
\mubyte \foo#\echar ABC\endmubyte %
% je-li ABC na začátku řádku, vloží před něj \foo
```

3.8. Potlačení expanze v parametrech write

Chceme-li převádět kontrolní sekvence zpětně na $\langle byte_sequence \rangle$ při zápisu do $\backslash write$ souborů, musíme potlačit případnou expanzi těchto kontrolních sekvencí například pomocí $\backslash let\macro=\relax$. Protože ale $\backslash write$ často pracuje asynchronně a kontrolních sekvencí mapujících UTF-8 můžeme mít stovky nebo tisíce, umožňuje $encTeX$ nastavit v době expanze parametrů $\backslash write$ příslušným kontrolním sekvencím význam $\backslash relax$ automaticky. Dělá to při $\backslash mubyteout \geq 3$ a význam $\backslash relax$ přiřadí právě těm kontrolním sekvencím, které mají ve výstupní části konverzní tabulky neprázdnou $\langle byte_sequenci \rangle$. Jakékoli jiné expanze mimo parametr $\backslash write$ probíhají normálním způsobem. Příklad:

```
\mubyte \foo ABC\endmubyte \def\foo{macro body}
\mubyteout=2
\immediate\write16{testwrite: \foo} % zapíše "testwrite: macro body"
\immediate\write16{testwrite: \noexpand\foo} % zapíše "testwrite: ABC"
\mubyteout=3
\immediate\write16{testwrite: \foo} % zapíše "testwrite: ABC"
\message{testmessage: \foo} % zapíše "testmessage: macro body"
\message{testmessage: \noexpand\foo} % zapíše "testmessage: \foo"
\edef\a{testedef: \foo} % expanduje na macro body
\foo % expanduje na macro body
\immediate\write16{\meaning\foo} % zapíše "\relax"
\message{\meaning\foo} % zapíše "macro:->macro body"
```

Pomocí zápisu $\backslash mubyte \langle control_sequence \rangle \backslash relax \backslash endmubyte$ je možno přidělit kontrolní sekvenci příznak, aby se neexpandovala v parametrech $\backslash write$ při $\backslash mubyteout \geq 3$, ale na druhé straně nebude konvertována do žádné $\langle byte_sequence \rangle$, ale vypíše se jako obvykle. Uvedený zápis má tedy stejný význam jako $\backslash mubyte \langle control_sequence \rangle _ _ \backslash string \langle control_sequence \rangle \backslash space \backslash endmubyte$, ale navíc šetří paměť $TeXu$, neboť TeX není nucen ukládat string $\langle byte_sequence \rangle$ do poolu.

Popsanou vlastnost $encTeXu$ můžete využít k vytváření vlastních maker, která jinak expandují v parametrech $\backslash write$ a jinak „normálně“. Příklad:

```
\mubyte \writeparameter \relax \endmubyte \def\writeparameter{}
\def\mymacro{\ifx\writeparameter\relax Tady expanduji ve write.
\else Tady expanduji normálně.\fi}
```

3.9. Asynchronní zpracování příkazu write

Je známo, že pokud nepoužijeme $\backslash immediate$, pak se argument příkazu $\backslash write$ expanduje až později: ne v okamžiku výskytu příkazu. Příkaz $\backslash write$ si proto uloží do své paměti aktuální hodnotu registru $\backslash mubyteout$ v době prvního zpracování a pak při expanzi a zápisu do souboru tuto hodnotu použije.

Díky této vlastnosti můžeme třeba pro soubor s obsahem zapisovat s hodnotou $\backslash mubyteout=3$ a současně při zápisu do jiného souboru ponecháme hodnotu $\backslash mubyteout=0$. To může být žádoucí například proto, že soubor je určen ke zpracování programem, který nemá implementovanou schopnost práce s UTF-8 kódováním. Vyzkoušejte:

```
\newwrite\tocfile \newwrite\indexfile
\immediate\openout\tocfile=\jobname.toc
```

```

\immediate\openout\indexfile=\jobname.idx
\mubyteout=3
\write\tocfile{parametr se bude později konvertovat do UTF-8}
{\mubyteout=0 \write\indexfile{parametr zůstane nezměněný bez konverze}}
\write\tocfile{zde se znovu provede konverze}
\end % a teprve v tento okamžik se všechny tři zápisy provedou

```

3.10. Hodnoty registru mubyteout

Kromě již zmíněných hodnot 0, 1, 2 a 3 registru `\mubyteout` může být někdy užitečné nastavit tento registr na hodnoty -1 , -2 a -3 . Význam těchto hodnot je vysvětlen v následující tabulce:

<code>\mubyteout</code>	$\langle byte \rangle \rightarrow \langle byte_sequence \rangle$	$\langle cs_name \rangle \rightarrow \langle byte_sequence \rangle$	potlačení expanze
0	ne	ne	ne
1	ano	ne	ne
2	ano	ano	ne
3	ano	ano	ano
-1	ano	ne	ano
-2	ne	ne	ano

Je-li zapnutá konverze $\langle byte \rangle \rightarrow \langle byte_sequence \rangle$, pak se tato konverze provádí i do logu a na terminál, zatímco konverze $\langle cs_name \rangle \rightarrow \langle byte_sequence \rangle$ a potlačení expanze se týkají jen argumentů `\write` a `\special`.

4. Argumenty primitivu special

V argumentech `\special` se často objevují texty v přirozeném jazyce (například texty pro záložky do PDF dokumentu). Při stále častějším používání UTF-8 je žádoucí, aby tyto texty byly kódovány v tomto kódování. EncTeX tuto možnost nabízí.

Argument primitivu `\special` je zpracován podle hodnoty celočíselného registru `\specialout`, který má implicitní hodnotu 0.

- `\specialout=0` – žádná konverze argumentu se neprovede.
- `\specialout=1` – provede se konverze jen podle vektoru `xchr`.
- `\specialout=2` – provede se konverze jen podle hodnoty `\mubyteout`.
- `\specialout=3` – provede se konverze podle hodnoty `\mubyteout` následovaná konverzí podle `xchr`.

Primitiv `\special` expanduje svůj argument okamžitě. Při `\specialout 2` nebo `3` se expanze provede podle hodnoty `\mubyteout` stejně jako u primitivu `\write`. Pak si příkaz `\special` uloží do paměti aktuální hodnoty `\specialout` a `\mubyteout` a tyto hodnoty použije ještě jednou při skutečném výstupu argumentu do `dvi` souboru.

5. Primitiv noconvert

EncTeX zavádí primitiv `\noconvert`, který potlačí případnou konverzi následujícího znaku nebo kontrolní sekvence. Přesněji: primitiv `\noconvert` je neexpandující a v sazbě neudělá nic (podobně jako `\relax`). Pokud se ale tento primitiv vyskytne v argumentu `\message` nebo `\errmessage`, pak jeho kontrolní sekvence není vůbec vytištěna. Navíc pak následující znak není konvertován na $\langle byte_sequence \rangle$, ačkoli třeba je `\mubyteout` kladný a znak je uveden ve výstupní části konverzní tabulky.

Primitiv `\noconvert` se chová stejně i v parametrech `\write` a `\special`. V tomto případě navíc může primitiv `\noconvert` potlačit konverzi následující kontrolní sekvence, ačkoli je třeba `\mubyteout` větší než 2.

Konstrukce `\noconvert\noconvert` vytiskne jedno `\noconvert`.

Pokud se má primitiv `\noconvert` vypsat v jiných situacích (například při výpisu kontextu při hlášení o chybě nebo při různých `\tracing...`), pak jeho kontrolní sekvence nemizí a navíc neovlivní tisk následujícího znaku.

6. Seznam primitivů encTeXu

Pro lepší orientaci uvádím přehled nových primitivů encTeXu:

- `\mubyte` — zanáší údaje do konverzní tabulky, viz kap. 3.
- `\endmubyte` — separátor pro `\mubyte`.
- `\mubytein` — registr integer. 0: vícebytová vstupní konverze potlačena, 1 a více: vícebytová vstupní konverze je aktivována.
- `\mubyteout` — registr integer, ovlivní konverzi do výstupních souborů `\write` a parametrů `\special`, viz 3.10.
- `\mubytelog` — registr integer, 0: vícebytová výstupní konverze do logu a na terminál potlačena, 1 a více: vícebytová výstupní konverze je aktivována.
- `\specialout` — registr integer, ovlivní zpracování parametrů `\special`, viz kap. 4.
- `\noconvert` — podobně jako `\noexpand`, ale pro konverze. Viz kap. 5.
- `\xordcode` — přístup k vektoru `xord`, viz 2.1.
- `\xchrcode` — přístup k vektoru `xchr`, viz 2.1.
- `\xprncode` — přístup k vektoru pro tisknutelnost znaků, viz 2.2.

Přehled prefixů při použití `\mubyte⟨first_token⟩⟨optional_space⟩⟨prefix⟩⟨byte_sequence⟩\endmubyte`. Znak # zde označuje libovolný token kategorie 6 a znak _ označuje libovolný token kategorie 7.

- žádný prefix — záznam do vstupní i výstupní tabulky.
- `_` — záznam jen do vstupní části tabulky
- `__` — záznam jen do výstupní části tabulky
- `#⟨číslo⟩` — vložení kontrolní sekvence, následujících `⟨číslo⟩` znaků ponechá input procesor beze změny.
- `\relax` — kontrolní sekvence nebude expandovat v parametrech `\write`

Další prefixy se mohou objevit v budoucích verzích encTeXu. Vždy budou mít kategorie odlišné od 11 a 12, takže pro začátek `⟨byte_sequence⟩` je vhodné v makrech použít token kategorie 11 nebo 12, aby se předešlo případnému konfliktu s budoucí verzí encTeXu.

7. Dokumentace k přiloženým souborům maker

Tato část dokumentace nebyla ve verzi Dec. 2002 revidovaná a je ponechána ve stavu z roku 1997 s výjimkou následujícího odstavce.

7.1. Kódování UTF-8

Pro vstupní kódování UTF-8 jsou připraveny soubory `utf8-csf.tex` a `utf8-t1.tex`. V tomto případě je překódování implementováno pomocí `\mubyte` a vektory `xord`, `xchr` jsou nastaveny tak, že na jejich úrovni je zachováno identické zobrazení.

7.2. Formáty typu plain-x-y

V balíčku jsou připraveny inicializační soubory pro vygenerování formátu podobnému standardnímu formátu plain. Například příkazem

```
$ tex -ini -enc plain-1250-cs
```

vygenerujeme formát analogický plainu, který čte vstupní soubory v kódování CP1250 a pracuje s CS-fonty.

V balíku jsou k dispozici tyto inicializační soubory pro plain:

<code>plain-il2-cs</code>	...	vstup podle ISO8859-2, textové fonty v TeXu: CS-font
<code>plain-kam-cs</code>	...	vstup podle Kamenických, textové fonty v TeXu: CS-font
<code>plain-1250-cs</code>	...	vstup podle CP1250, textové fonty v TeXu: CS-font
<code>plain-852-cs</code>	...	vstup podle CP852, textové fonty v TeXu: CS-font
<code>plain-il2-dc</code>	...	vstup podle ISO8859-2, textové fonty v TeXu: DC
<code>plain-kam-dc</code>	...	vstup podle Kamenických, textové fonty v TeXu: DC
<code>plain-1250-dc</code>	...	vstup podle CP1250, textové fonty v TeXu: DC
<code>plain-852-dc</code>	...	vstup podle CP852, textové fonty v TeXu: DC

7.3. Poznámka k dlouhým názvům souborů

Všechny soubory `*.tex` v balíčku splňují DOSové omezení na délku názvu 8+3. Výjimkou z tohoto pravidla jsou pouze soubory `plain-x-y` popsané výše a analogické inicializační soubory pro LaTeX . Pokud používáte systém, který je omezen na 8+3, doporučuji pro každé kódování zvolit jedno písmeno (například `c=cs`, `d=dc`, `i=il2`, `w=1250`, `p=852`, `k=kam`, `o=koi8`, `m=mac`) a nahradit názvy souborů v distribuci těmito názvy:

<code>plain-il2-cs.tex</code>	<code>plain-ic.tex</code>
<code>plain-kam-cs.tex</code>	<code>plain-kc.tex</code>
<code>plain-1250-cs.tex</code>	<code>plain-wc.tex</code>
<code>plain-852-cs.tex</code>	<code>plain-pc.tex</code>
<code>plain-il2-dc.tex</code>	<code>plain-id.tex</code>
<code>plain-kam-dc.tex</code>	<code>plain-kd.tex</code>
<code>plain-1250-dc.tex</code>	<code>plain-wd.tex</code>
<code>plain-852-dc.tex</code>	<code>plain-pd.tex</code>
<code>kam-latex.tex</code>	<code>latex-ki.tex</code>
<code>852-latex.tex</code>	<code>latex-pi.tex</code>

Obsah `\message` v souborech `plain-x-y` neměňte. Například formát `plain-wc` se po spuštění představí svým plným jménem

```
The format: plain-1250-cs <Sep. 1997>.
```

7.4. Kódovací tabulky

Protože změna vektorů `xord` a `xchr` může totálně rozhodit chování TeX u zcela k nepoznání, doporučuji používat určité soubory, které nastaví požadované kódování, a dále s primitivy `\xordcode`, `\xchrcode` a `\xprncode` za běhu TeX u moc nelaškovat. V balíčku `encTeX` jsou k dispozici soubory, které změnu vektorů pro běžná kódování definují. Tyto soubory mají obvyklou příponu `tex`. Říkáme jim kódovací tabulky. Rozlišujeme dva typy kódovacích tabulek.

7.5. První typ kódovacích tabulek

První typ tabulek deklaruje vnitřní kódování TeX u ve vztahu ke kódování, které je běžně používané v hostitelském operačním systému. Máme-li například v systému kódování ISO-8859-2 a vnitřní kódování TeX u volíme podle Corku (kódování je označováno jako T1), pak tabulka musí předefinovat `xord` vektor tak, aby mapoval znaky z ISO-8859-2 do T1 a vektor `xchr` musí převádět zpátky z T1 do kódování systému.

Tento typ tabulek je použit v inicializačních souborech `plain-*.tex` a obsahuje v názvu souboru vstupní i cílové vnitřní kódování TeX u. Podívejte se, jak vypadá například tabulka `il2-t1.tex`, která definuje vnitřní kódování TeX u podle Corku a vstupní kódování ISO8859-2.

Každá tabulka prvního typu čte soubor `encmacro.tex` s definicemi maker `\setcharcode`, `\expandto`, `\texaccent`, `\texmacro` a `\redefaccent`.

- `\setcharcode #1 #2 #3 #4 #5 #6 #7` deklaruje TeX ové kódy pro jeden znak. Nastaví `xord[#1]=#2`, `xchr[#2]=#1`, `\xprncode#2=#7` a postupně nastaví `\lccode`, `\uccode`, `\sfcode` a `\catcode` znaku s kódem `#2` na hodnoty `#3`, `#4`, `#5` a `#6`. Je-li `#1` otazník, pak se `xord` a `xchr` nenastaví.
- `\expandto {<definice>}` definuje aktivní podobu znaku `#2` z posledního `\setcharcode` tak, že tento token expanduje na `<definici>`. Podrobněji: je-li v `\setcharcode` uvedeno `#6=13`, pak bude každý výskyt znaku `#2` expandovat na `<definici>`. Není-li v `\setcharcode` řečeno `#6=13`, pak k expanzi znaku `#2` na `<definici>` dojde teprve tehdy, když bude (třeba později) nastaveno `\catcode` znaku `#2` na 13.
- `\texaccent` uvzápis akcentu připravuje expanzi „zápisu akcentu“ na znak s kódem `#2` z naposledy použitého `\setcharcode`. Například zápis `\v C` bude po načtení souboru `il2-t1.tex` expandovat na znak s kódem "83. Pokud zápis pro akcent není v tabulce uveden, zůstává v původním významu, tj. třeba `\v g` expanduje na primitiv `\accent`, který usadí háček nad písmeno `g`. K aktivaci všech „zápisů akcentu“ dojde až po použití makra `\redefaccent` (viz níže).
- `\texmacro #1` deklaruje makro `#1` tak, že bude expandovat na znak s kódem `#2` z naposledy použitého `\setcharcode`. K předefinování makra `#1` dojde (na rozdíl od `\texaccent`) okamžitě. Například makro

`\S` bude po načtení souboru `il2-t1.tex` expandovat na znak s kódem 9F, protože na této pozici je podle Corku znak paragraf.

- `\redefaccent #1` aktivuje expanzi zápisů podle `\texaccent` pro jeden konkrétní akcent #1.

Kromě toho je na začátku tabulky čten soubor definic závislých na kódování textového fontu \TeX u. V naší ukázce jde například o soubor `t1macro.tex`. Definují se tam sekvence `\promile`, `\clqq` a další.

Může se stát, že nechceme uvedená makra použít, ale hodnoty z tabulky načíst chceme. Pak můžeme přistoupit k následujícímu triku: Definujeme si makra `\setcharcode` až `\redefaccent` sami a dále provedeme načtení tabulky takto:

```
\let\origininput=\input \def\input #1 \origininput il2-t1
\let\input=\origininput
```

V balíčku jsou připraveny tyto tabulky prvního druhu:

Název souboru	vstupní kódování	vnitřní kódování \TeX u
<code>il2-csf.tex</code>	ISO8859-2	CS-font
<code>kam-csf.tex</code>	Kamenických	CS-font
<code>1250-csf.tex</code>	CP1250, MS-Windows	CS-font
<code>852-csf.tex</code>	CP852, PC Latin2	CS-font
<code>il2-t1.tex</code>	ISO8859-2	T1 alias Cork
<code>kam-t1.tex</code>	Kamenických	T1 alias Cork
<code>1250-t1.tex</code>	CP1250, MS-Windows	T1 alias Cork
<code>852-t1.tex</code>	CP852, PC Latin2	T1 alias Cork

Za zmínku stojí první uvedená tabulka `il2-csf.tex`, protože ta jediná ponechává vektory `xord` a `xchr` beze změny. Tuto tabulku je tedy možné použít i v \TeX u, který neobsahuje rozšíření `enc \TeX` . Všechny ostatní tabulky `enc \TeX` explicitně vyžadují.

7.6. Druhý typ kódovacích tabulek

Druhý typ tabulek provádí překódování pouze na vstupní straně \TeX u. Poznáme je podle toho, že nemají na konci názvu značku pro vnitřní kódování \TeX u (tj. `t1` nebo `csf`), ale značku používanou pro kódování operačního systému (např. `il2`, `kam`). Třeba tabulka `kam-il2.tex` provádí na vstupní straně konverzi z kódování kamenických do kódování ISO8859-2. Tento typ tabulek pozměňuje pouze vektor `xchr`, ale výstupní vektor `xord` ponechává beze změny. Takovou tabulku použijeme, pokud \TeX em načítáme soubor, který je v jiném kódování, než běžně používáme na našem operačním systému. Přitom výstup do log, aux apod. ponecháme v kódování podle našeho systému. Tyto změny kódování je možné provádět i v průběhu zpracování jediného dokumentu.

Druhý typ tabulek navazuje na vstupní kódování deklarované dříve tabulkou prvního typu. Nastavení vnitřního kódování \TeX u není vůbec druhým typem tabulek měněno. Uvedeme příklad. Při generování formátu jsme použili tabulku prvního typu `il2-t1.tex`, takže vnitřní kódování máme podle Corku. Nyní můžeme při zpracování dokumentu na přechodnou dobu vybrat některou z tabulek `*-il2.tex`, třeba:

```
\input kam-il2
\input dokument
\restoreinputencoding
nyní mohu pracovat v původním kódování...
\end
```

V době, kdy probíhá načítání souboru `dokument.tex` se provádí překódování z Kamenických do T1, uvnitř \TeX u se vše zpracovává v T1 a výstup na terminál a do logu máme v ISO8859-2. V tomto kódování je také zapsán další text pod `\restoreinputencoding`. Tabulka totiž deklaruje toto makro, aby byl možný návrat k původnímu nastavení vektoru `xord`.

Při použití tabulek druhého typu musíme dát velký pozor, abychom něco neudělali špatně. V našem příkladě jsou všechny výstupy do souborů typu `aux` v ISO-8859-2, takže je při opakovaném spuštění \TeX u nesmíme načítat v okamžiku, kdy máme nastaven vstupní kód podle Kamenických. To je také důvod, proč nedoporučuji generovat formát příkazem `\dump` v situaci, kdy máme načtenou tabulku druhého typu.