Článek zveřejněn v časopise Zpravodaj CSTUGu 1/2021
https://www.cstug.cz/bulletin/bul2020-1-2.shtml

OpTEX – pokračování maker OPmac pro LuaTEX

Někteří čtenáři možná znají TEXovou sadu maker OPmac pro plainTEX. Ta umožňuje při zachování jednoduchosti a přímočarosti užití maker plainTEXu využít v podstatě všechny pokročilé vlastnosti při sazbě současných dokumentů. Pokračovatelem této koncepce je sada maker OpTEX [1], která společně s LuaTEXem tvoří ucelený nástroj na přípravu dokumentů. Zůstává zachována přímočarost a věci se navíc zjednodušují, protože nemusíme myslet na rozličné vlastnosti jednotlivých TEXových enginů ani na všemožná starodávná kódování národních abeced. Vše totiž běží výhradně v LuaTEXu a v Unicode.

V únoru 2020 jsem oslovil odběratele TEXového listu `cstex`, že jsem zveřejnil alfa verzi OpTEXu a uvítám případnou diskusi nad koncepcí a požadovanými vlastnostmi. Děkuji všem, kteří se mi v té době ozvali. Tehdy to bylo velmi nehotové dílo, které (ačkoli bylo okamžitě zařazeno do TEXlive i MikTEXu) vlastně vznikalo uživatelům pod rukama. Nyní, po roce intenzivního vývoje, OpTEX dospěl k verzi 1.0 a je tedy ustálen a připraven k běžnému použití. V tomto článku bych chtěl o něm zevrubně seznámit české a slovenské uživatele TEXu.

1 Hlavní vlastnost: přímočaré použití TEXu

OpTEX je LuaTEXový formát, podobně jako LATEX nebo ConTEXt. Na rozdíl od těchto dvou jmenovaných ale nevytváří novou vrstvu pro řízení dokumentu nad primitivním TEXem. Předpokládá se, že uživatel použije makra OpTEXu ve výchozí podobě nebo je pozmění k obrazu svému, když třeba řeší nějaký konkrétní typografický požadavek. Makra tedy neřeší vymezení nějaké odlišné syntaxe od syntaxe TEXu ani stovky nově deklarovaných parametrů, kterými je možné nastavovat vlastnosti při zpracování dokumentu. Mezi takovými parametry nakonec pokročilým uživatelům stejně něco bude chybět a budou muset sáhnout po možnostech, které nabízí výhradně jen primitivní úroveň samotného TEXu. Proč tedy tímto přístupem nezačít rovnou?

Tento koncept si uvedeme na příkladu. Chceme-li v OpTEXu nastavit šířku sazby, použijeme primitivní registr `\hsize` a píšeme třeba `\hsize=10cm`. V LATEXu je naproti tomu pro uživatele deklarovaný registr `\textwidth` a v dokumentaci se dočteme, že je možné jej nastavit pomocí `\setlength{\textwidth}{10cm}`, tedy odlišnou syntaxí od primitivní. Je také potřeba pořádně vědět, v jakých případech se hodnota deklarovaného registru `\textwidth` prokopíruje do primitivního `\hsize`. Věci jsou tedy daleko komplikovanější.

ConTEXt v tomto ohledu není o moc přímočařejší, spíše naopak. Je třeba nastavit parametr `width` v jistém kontextu toho ConTEXtu a v key-value syntaxi. A jak to souvisí s primitivním `\hsize` je těžko dohledatelné. Je zde vytvořena zcela nová vrstva nabízející nastavení sazby nad původní TEX primitivní vrstvou.

V OpTEXu můžeme nastavit umístění sazby na stránce pomocí primitivních `\hoffest` a `\voffset`, ovšem můžeme také celkově nastavit okraje sazby velmi pohodlně jedním příkazem `\margins`, což je zkratka za nastavení `\hsize`, `\vsize`, `\hoffset` a `\voffset` v souvislosti se znalostí fyzického rozměru papíru. LATEX potřebuje k takovému nastavení okrajů stránky externí balíček `geometry`.

OpTEX tedy primitivní vrstvu TEXu nezakrývá, naopak předpokládá její časté použití. Pracujeme s TEXem, stačí nám tedy vědět, co a jak dělá TEX, a nezkoumat nějaké další syntaxe, parametry a nové terminologie a algoritmy s tím spojené.

2 Dokumentace

Na stránce 26 dokumentace k OpTEXu [2] najdete sumární přehled příkazů použitelných pro značkování dokumentů psaných v OpTEXu. Ke každému příkazu vede odkaz na některou z předchozích 25 stránek, kde je příkaz dokumentován z uživatelského pohledu podrobněji. Takže autorovi obsahu dokumentu v zásadě stačí těchto 25 stránek. Ovšem zvídavý autor může v této

části dokumentace znovu kliknout myší na jméno příkazu a octne se v místě, kde je technický rozbor implementace příkazu a výpis odpovídající části zdrojového kódu. Tato technická část dokumentace má zhruba zbylých 170 stránek.

Dokumentace je pochopitelně vytvořená OpTEXem. Při jejím zpracování se čtou aktuální zdrojové kódy OpTEXu a k nim připojené průvodní texty a toto je zařazeno do sazby technické části dokumentace. Takže to připomíná literální programování, kde máme ve stejném místě, jako jsou zdrojové texty maker, rovnou napsaný i technický rozbor problému.

Aby mohl čtenář rozumět technické dokumentaci a mohl si přizpůsobit makra dle svého přesvědčení a navíc přesně rozuměl, co se při sazbě jeho dokumentu děje, musí znát TEX. S ním se může seznámit v jiném dokumentu [3], kde jsem se v kostce snažil komprimovat základní informace o TEXu a plainTEXu včetně přehledu všech jeho primitivních kontrolních sekvencí a jejích vlastností do 26 stránek textu. V takto komprimované podobě se informace o kompletních vlastnostech TEXu (včetně jeho běžných rozšíření) asi nikde jinde nevyskytují.

Třetí dokumentací, která doplňuje předchozí dvě, je technický souhrn chování TEXu, maker plainTEXu a Unicode-math v matematické sazbě [4]. Rovněž se jedná o velice zhuštěnou, ale kompletní, informaci na třiceti stránkách.

A to je úplně všechno. Znalosti uvedené v těchto třech dokumentech umožní uživateli získat absolutní kontrolu nad sazbou. Nepotřebuje k tomu tři tisíce LATEXových balíčků včetně dokumentace ke každému z nich (těch balíčků je cca desetkrát více než je TEXových primitivních příkazů) ani se nepotřebuje vyznat ve 180 PDF souborech monstrózní ConTEXtové dokumentace.

Na cestě k absolutní kontrole nad sazbou může český či slovenský čtenář také využít [5] a ostatní čtenář třeba [6].

3 Základní vlastnosti

OpTEX jsem již představil v článku [7]. Povědomí o základních vlastnostech OpTEXu můžete také získat prolistováním prvních 26 stránek již zmíněné dokumentace [2]. Zde uvedu jen stručný souhrn a k některým věcem se vrátím v dalších sekcích tohoto článku.

- Systém pro výběr fontů,
- Unicode-math,
- možnost nastavení barev,
- vkládání obrázků včetně Inkscape výstupu s popisky,
- kresba jednoduché grafiky v závislosti na velikosti sazby,
- vkládání objektů na přesně vymezenou pozici,
- externí, interní odkazy včetně hyperlinků,
- automatické generování obsahu,
- automatické generování seznamů referencí přímo z `.bib` souborů,
- automatické generování rejstříků,
- pohodlné nastavení okrajů sazby,
- vzory dělení pro všechny jazyky dostupné v distribuci,
- přepínání automaticky generovaných textů podle zvoleného jazyka,
- poznámky pod čarou,
- poznámky v okraji sazby,
- listingy včetně automatického zvýraznění dle syntaxe jazyka,
- záložky pro PDF prohlížeč v Unicode,
- pohodlná tvorba tabulek,
- tisk textů lorem ipsum dolor sit,
- možnost snadné přípravy prezentací pro zpětný projektor,
- předdefinované styly report a letter,
- nástroje pro programátory maker (cykly, čtení údajů klíč-hodnota, ...),
- oddělené jmenné prostory pro uživatele a pro programátory maker.

4 Není třeba externích programů

V LaTeXu se pro sestavování bibliografických referencí používá BibTeX nebo Biber, pro seřazení rejstříku Makeindex nebo Xindy a pro automatické zvýraznění syntaxe v listingu Python nebo podobný nástroj. Nic z toho OpTeX nepotřebuje, pro plnění těchto úkolů si vystačí sám. Vše probíhá na úrovni maker TeXu.

OpTeX čte přímo BibTeXové soubory `.bib` a sestavuje z nich seznam referencí. Stačí tedy dva průchody dokumentem a máte citace v pořádku. Při prvním průchodu se OpTeX seznámí s použitými referencemi v `\cite` příkazech a v místě seznamu literatury je rovnou použije, tj. vyhledá z `.bib` souboru potřebné údaje, správně je seřadí, přidělí k nim čísla a vytiskne seznam v souladu s použitým bibliografickým stylem. V druhém průchodu pak přidělení čísel v seznamu referencí použije v místech `\cite` příkazů. Je to jistě jednodušší než známá čtveřice `latex`, `bibtex`, `latex`, `latex`. K nastavení vlastností bibliografického stylu se používají přímo makra TeXu.

OpTeX sestaví rejstřík přímo z dat z předchozího průchodu tak, že použije sofistikovaný vícekrokový algoritmus řazení, který se dá konfigurovat pro většinu běžných jazyků. Pro češtinu, slovenštinu a angličtinu funguje bez problémů. Dodatečnými makry se pak dá řídit podoba rejstříku. Je daleko příjemnější vědět, že nemusíme pamatovat na závěrečné spuštění `makeindex` po posledních korekturách, protože celý „Makeindex“ je přímo součástí OpTeXu a projeví se při každém zpracování dokumentu. Navíc funguje relativně rychle díky řadicímu algoritmu mergesort, který využívá makrojazyk TeXu velmi efektivně.

Automatické zvýraznění syntaxe tištěných listingů probíhá rovněž na úrovni makrojazyka TeXu. Jednotlivá pravidla pro zvýraznění se konfigurují v příslušných makro souborech `hisyntax-c`, `hisyntax-html` atd. Není zapotřebí externího programu.

5 Systém pro výběr fontů

OpTeX nabízí uživateli pohodlnou manipulaci s fonty a fontovými rodinami se zachováním základní koncepce z plainTeXu, že tedy mezi variantami jednou vybrané rodiny fontů přepínáme pomocí `\rm`, `\bf`, `\it` a `\bi`. Mimoto je možné předsunout před takové přepínače „fontové modifikátory“, např. `\caps`, `\cond`, pokud to fontová rodina podporuje. Rodinu fontu uživatel nastaví pomocí `\fontfam`. Modifikátory mohou přepínat na sobě nezávislé vlastnosti a mohou se všelijak shlukovat a kombinovat.

Uživatel také může použít přímo primitivní příkaz `\font` pro výběr fontů a může snadno zakomponovat vytvořený primitivní přepínač fontu do maker tak, že bude tento font zvětšen podle uživatelem stanovené velikosti kdekoli v dokumentu společně s ostatními fonty zařazenými do fontových souborů. Není tedy třeba (jako v plainTeXu) psát celou sadu primitivních příkazů `\font` pro různé velikosti.

OpTeX umožňuje zařazení deklarací rodin fontů do tzv. fontových souborů. Desítky rodin fontů již takto zařazeny jsou. Uživatel se může inspirovat, jak je to uděláno, a zařadit další rodinu. Nebo se o formátu těchto souborů dočte v technické dokumentaci [2].

Uživatel si může OpTeXem vytisknout vzorníček všech rodin fontů, které jsou zařazeny do fontových souborů, jednoduše pomocí

```
\fontfam[catalog]
\bye
```

V tomto vzorníčku jsou přehledně uvedeny všechny přepínače variant a všechny modifikátory dostupné v každé jednotlivé rodině následované ukázkou textu z takto vybraného fontu. Viz také [8].

OpTeX přirozeně nabízí možnost použití „fontfeatures“ z OpenType fontů. Je k tomu vyhrazen fontový modifikátor `\setff` s parametrem, který obsahuje zkratku příslušné vlastnosti. Jedná se o standardní čtyřpísmennou zkratku známou z dokumentace k fontu, dokumentace k OpenType standardu nebo z výpisu příkazu `otfinfo -f`. Nad těmito názvy „fontfeatures“

není vytvořena žádná nová vrstva jiných názvů, jako to dělá třeba `fontspec` z LaTeXu. Věci se totiž v OpTeXu dělají přímočaře.

Podporuje-li rodina optické velikosti fontů, podporuje je i OpTeX. Výchozí rodinou fontů je Latin Modern, která optické velikosti umožňuje věrna tradici, kterou v této věci zavedl už Donald Knuth, když představil svou rodinu fontů Computer Modern.

V OpTeXu pracujeme výhradně s fonty v Unicode (OpenType). Tím se vyhneme všelijakým obskurnostem s různými starodávnými kódováními. Matematická sazba také probíhá v Unicodemath, což výrazně zjednodušuje práci s matematickým fonty. Máme jenom jeden font se všemi matematickými abecedami a se všemi symboly pohromadě na jednom místě. Práce s více matematickými rodinami fontů (jako v klasickém plainTeXu) tím není ale omezena. Není ovšem nutné ji moc využívat, ledaže by základní matematický font neobsahoval všechny potřebné symboly a bylo nutno sazbu kombinovat s dalším matematickým fontem.

6 Křehká makra v názvech sekcí neexistují

Uživatelé LaTeXu asi znají pojem „fragile command“ a jaké s tím je trápení. Ve stručnosti: nelze napsat do názvu sekce, kapitoly atd. jednoduše každé makro, protože takové makro se může na cestě k automaticky generovanému obsahu rozsypat. Tato cesta totiž vede expanzí přes pracovní textové soubory. Divím se, že po skoro třiceti letech, kdy byly do eTeXu přidány příkazy `\detokenize` a `\scantokens`, stále LaTeXoví uživatelé bojují s křehkými příkazy a snaží se to řešit starodávným a komplikovaným způsobem pomocí `\protect` nebo `\DeclareRobustCommand`, což vkládá tajemnou závěrečnou mezeru do názvu makra. To zase způsobuje méně snadné trasování.

OpTeX čte názvy sekcí ve verbatim módu a tokenizuje je až v okamžiku použití. Tím odpadají veškeré starosti s křehkými makry. Navíc to umožní vkládat do názvů sekcí i verbatim text, například

```
\sec Titul včetně `\takove{` záležitosti
```

Všimněte si, že verbatim konstrukce (uzavřená mezi znaky `` `...` `` deklarované v OpTeXu jako `\verbchar`), umožňuje zapsat i nepárové TeXové závorky. To je v LaTeXu v podstatě nemožné, protože titul je v LaTeXu ohraničen pomocí těchto závorek v páru. V OpTeXu je titul ohraničen koncem řádku. Verbatim konstrukce z titulů sekcí a kapitol správně doputují do obsahu, do plovoucího záhlaví či do PDF záložek. Bez kompromisů, kterým se musel podřídit například LaTeXový balíček `cprotect`.

7 Odlehčené značkování

Značkovací jazyk OpTeXu vychází z předpokladu, že zdrojový text `.tex` je určen především pro člověka. Stroj by se tomu měl podřídit. Člověk tento text vytváří, čte jej, manipuluje s ním. Proto jsem se od počátku vzniku svých maker snažil o co nejstručnější, efektivní a přehledné značkování. Takže jsem se vyvaroval nadbytečnému množství závorek `{...}` a nepřijal jsem vnořující se konstrukce z LaTeXu `\begin{...}...\end{...}`. Text dokumentu *nekódujeme*, ale prostě *píšeme*.

Faktum je, že existují ještě přívětivější způsoby značkování, jako je Markdown. Jenže v tomto jazyku zase nemáme absolutní kontrolu nad sazbou, kterou potřebujeme, když vytváříme PDF verzi dokumentu. Je jistě možné, že nějaký autor dokumentu použije Markdown a pak jej nechá zkonvertovat do OpTeXu. Sazeč poté má k dispozici verzi dokumentu `.tex`, ve které doplní další potřebné věci a vytvoří sazbu pomocí OpTeXu. Takže sazeč manipuluje s `.tex` souborem, a nakonec `.tex` soubor je konečná verze dokumentu určená k archivaci. Soubor `.tex` by měl být ideálně středem všeho, z něj proběhne jednak sazba do PDF a jednak konverze do dalších případných formátů a je určen k archivaci pro možné budoucí použití například při novém vydání dokumentu.

Na základě těchto úvah vznikl dokument OMLS [9], který vymezuje standard značkování souborů `.tex` při použití OpTeXu. Tento standard obsahuje stanovení syntaxe a významu všech

značek, které umožní v OpTEXu vytvořit dokument s obvyklými vlastnostmi (podobně jako Markdown). Je určen pro případné autory konvertorů z `.tex` do jiných formátů a obráceně. Autoři těchto programů se tedy mohou opřít o přesné vymezení možností, které se v `.tex` souborech mohou vyskytovat. A je-li tam nějaká jiná konstrukce mimo standard OMLS, má být ignorována. Pro případné zpracování matematických vzorečků mezi `$...$` nebo `$$...$$` se očekává interpret analogický systému MathJax.

Dokument OMLS [9] je poměrně nová věc, zatím tedy podle něj skutečné aplikace nejsou vytvořeny.

8 Odlehčený formát

Nejen značkování, ale i implementace maker v OpTEXu je odlehčená. Makra jsou co možná nejjednodušší. Uvedu dvě srovnání.

Velikost formátového souboru `optex.fmt` je na mé implementaci TEXlive cca 750 kB. Srovnáme-li to s velikostí ostatních běžných formátových souborů pro LuaTEX (viz tabulku 8.1), shledáme, že je skoro nejmenší.

Tabulka 8.1 Velikosti formátových souborů `.fmt` v TEXlive

OpTEX	eplain	csplain	LuaLATEX	ConTEXt
750 kB	1.2 MB	380 KB	3.2 MB	11 MB

Přitom ve formátu OpTEXu je skrytá srovnatelná inteligence LATEXového jádra `latex.fmt` včetně doplňkových balíčků `xcolor`, `hyperref`, `url`, `listings`, `biblatex`, `graphicx`, `geometry`, `amsmath`, `amssymb`, `fontspec`, `unicode-math`, `cprotect`, `eqparbox`, `tabularx`, `booktabs`, `keyval` a mnoha dalších. K dispozici je možnost použít všechny dostupné vzory dělení všech jazyků. Vzory dělení ve `.fmt` souboru totiž nepřekážejí, LuaTEX jako jediný engine je dokáže načíst až podle potřeby během zpracování dokumentu.

Druhé srovnání vychází z testovacího souboru:

```
\loggingall
\cslang             % české vzory dělení slov
\fontfam[Adventor]  % fontová rodina Adventor (Unicode)

Ahoj světe!
\bye
```

a jeho srovnatelných variant vytvořených pro ostatní dva formáty. Sledujeme počet řádků v `.log` souboru při `\loggingall`. Dále vypneme `\loggingall` a sledujeme čas. Viz tabulku 8.2.

Tabulka 8.2 Počet řádků v `.log` souboru při plném trasování a čas bez trasování

OpTEX	LuaLATEX	ConTEXt
1,8 k	3,6 M	3×231 k
0,5 s	1,0 s	$3 \times 1{,}1$ s

Vidíme, že soutěž o nejmenší počet trasovaných řádků LATEX zcela prohrává. Provádí asi dvatisíckrát více interních úkonů než OpTEX. ConTEXt se zase potřebuje spustit za všech okolností třikrát, i když to v tomto příkladě není nutné. A dobíhá při měření rychlosti do cílové rovinky poslední, i kdyby si dal to kolečko jen jednou. Asi přeci jen nějakou dobu trvá, než se přečte a dekomprimuje 11 MB binární `.fmt` soubor.

Příjemnou drobností je, že OpTEX nevytváří pracovní soubory, když nepotřebuje. Takže výše uvedený příklad vygeneroval jen `.log` a `.pdf`, zatímco LATEX si založil `.aux`, který v tomto případě nepotřebuje, a ConTEXt si založil `.tuc` soubor. Nezakládání zbytečných pracovních

souborů možná oceníte, když si budete dělat krátké testovací dokumenty a nebudete kvůli tomu muset mazat z disku desítky `.aux` souborů a jemu podobných.

OpTEX v rámci pravidla „dělat věci jednoduše“ zakládá (když je to nutné) jediný pracovní soubor `.ref`, kde má pohromadě vše potřebné pro obsah, bibliografii, dopředné reference, rejstřík atd. Nemáte tedy na disku sbírku souborů `.toc`, `.aux`, `.lof`, `.idx`, `.ind`, `.bbl` a dalších.

LATEX občas trpí problémem, že se zasekne při opakovaném čtení `.aux` souboru, například když změníte hlavní jazyk dokumentu. To v OpTEXu nenastane. Je navíc vybaven kontrolou verze svého `.ref` souboru, aby nedošlo ke zmatení čtení například po přidání nového rysu do nové verze. Vidí-li OpTEX nekompatibilní verzi `.ref` souboru, ignoruje jej a založí si nový. Makra OPmac také zakládají `.ref` soubor, ale s jiným číslem verze, takže nikdy nedochází ke vzájemnému zmatení.

9 Balíčky v OpTEXu

Ačkoli OpTEX obsahuje vše potřebné pro tvorbu běžných dokumentů, občas se vyskytne požadavek něco dalšího implementovat pro konkrétní účel. Řešení k velkému množství takových požadavků, z nichž by si asi každý vyžádal samostatný LATEXový balíček, vkládám do souhrnného návodu na stránce OpTEX triků [10]. Zejména, pokud si toto řešení vyžádá jen několik málo řádků makro kódu, což je poměrně obvyklý případ.

Někdy ale je potřeba vyřešit něco obsáhlejšího. OpTEX nabízí možnost tvorby balíčků, které jsou pak do dokumentu zavedeny pomocí `\load`. Balíčky může vytvořit dle pokynů v technické části dokumentace kdokoli, ovšem zatím jsem několik prvních balíčků připravil jako ukázku sám:

- `qrcode` pro výpočet a sazbu QR kódů,
- `vlna` automaticky přidává nezlomitelné mezery za předložkami s využitím `luavlna`,
- `emoji` umožní používat barevné emotikony z rozsáhlé nabídky podle [11].

10 Jmenné prostory

Uživatel může definovat a používat kontrolní sekvence typu `\slovo`. Některé z nich nesou význam primitivní kontrolní sekvence nebo makra OpTEXu, ale uživatel si je může předefinovat bez ztráty funkčnosti OpTEXu. Má tedy k dispozici jmenný prostor typu `\slovo` a když si něco předefinuje a v původním významu nikdy ve svém dokumentu přímo nepoužije, nemá problém. Proč to funguje? OpTEX interně používá duplikáty primitivních sekvencí tvaru `\_slovo` a interní makra má rovněž v tomto tvaru. Uživatel tedy bez problému může provést třeba `\def\fi{finito}` a žádné interní makro se nerozsype, protože interní makra OpTEXu používají konstrukce `\_if...\_fi`. Když ale uživatel sám používá svá makra obsahující `\if...\fi` a napíše omylem `\def\fi{finito}`, samozřejmě se mu jeho makra rozsypou. Ale to se celé děje uvnitř jeho uživatelského prostoru. Uživatel by měl mít představu o jménech, která sám deklaruje a pak používá.

OpTEX nabízí také izolované jmenné prostory pro každý případný balíček. Kontrolní sekvence primitivní či deklarované v OpTEXu tam mohou být použity v read módu ve tvaru `\_slovo` a dále si balíček může definovat vlastní interní sekvence ve tvaru `\_pkg_slovo`. Přitom tvůrce maker nemusí opakovaně psát `\_pkg_foo`, `\_pkg_bar` (až oči přecházejí), ale deklaruje si jednou `\_namespace{pkg}` a používá daleko čitelnější syntaxi `\.foo`, `\.bar`. Konceptu, kdy se v makro kódu neustále opakují názvy jmenných prostorů (jako to dělá Expl3), jsem se s radostí vyhnul.

Z pravidla o uživatelském jmenném prostoru zatím existuje jedna smutná výjimka. Řídicí sekvence `\par` je zadrátována jako jediná hluboko v TEXu samotném a rodí se například na prázdných řádcích. Její předefinování tedy ovlivní chování TEXu. Už jsem ale připravil záplatu ke zdrojovým textům `pdftex.web`, která umožní deklarovat tuto rodící se sekvenci jakkoli jinak, ne nutně jako `\par`. Tuto záplatu mám kompletně připravenou, zdokumentovanou a testovanou. Bohužel jsem ji nestihl podat včas pro nový TEXlive 2021. Takže se to objeví asi až za rok. Do té doby mám v dokumentaci OpTEXu i pro uživatele informaci o výjimce ve jmenném prostoru: „vyvarujte se `\def\par`, pokud netušíte, co tím můžete způsobit“. Předpokládám, že záplatu

z pdfTEXu převezmou v budoucnu i tvůrci LuaTEXu, jako to udělali i s jinými mými návrhy na změnu chování pdfTEXu. Takže to pak s radostí využiji v OpTEXu.

11 Využití rozšířených vlastností LuaTEXu

LuaTEX umí především interpretovat Lua kódy. To jsem využil v minimální míře, některé věci by se daly v budoucnu ještě přepsat. Lua skripty jsem použil při generování Unicode řetězců pro PDF záložky a při interpretaci znaků `_` a `.` v řídicích sekvencích. Dále jsem je využil na drobnosti jako např. porovnávání stringů. OpTEX načítá vlastní sadu základních Lua funkcí `optex.lua` a ukládá si ji přímo do formátu. Při každém startu pak tuto sadu funkcí inicializuje prostřednictvím `\everyjob`.

Dále je v Lua implementován font-loader pro OTF fonty. Ten se spustí až dle potřeby, při prvním `\fontfam`. Příslušný Lua kód je kompletně čten z LaTEXového balíčku `luaotfload`, což bohužel činí OpTEX závislým na tomto balíčku. Tvůrci `luaotfload` přebírají (pravidelně při každé nové verzi) Lua kódy font-loaderu v nezměněné podobě z ConTEXtu a přidávají tam pár svých dalších drobností. Stál jsem před rozhodnutím přebrat font-loader přímo z ConTEXtu (pak by byl ale OpTEX závislý kompletně na ConTEXtu, což je daleko větší bumbrlíček než jen jeden balíček `luaotfload`). Nebo pravidelně přebírat kódy z ContTEXtu a ohýbat je pro své potřeby (to bych ale vlastně dubloval práci tvůrců `luaotfload`). Nebo si font-loader naprogramovat v Lue zcela sám (to ale dosud nikdo jiný nedokázal než Hans Hagen a znamenalo by to vlastně jen vymýšlet jednou objevené kolo a při zveřejnění nového standardu OTF s novými vlastnostmi toto kolo neustále samostatně vylepšovat nezávisle na Hansi Hagenovi). Ponechal jsem tedy kompromis a závislost na `luaotfload`. Netěší mě to, musím neustále sledovat, co tam s těmi kódy v nových verzích jejich tvůrci dělají a zda nedojde k nekompatibilitě se základními Lua funkcemi OpTEXu.

Kromě zmíněného skriptovacího jazyka Lua disponuje LuaTEX dalšími rozšířeními, z nichž některé jsem využil. Například se dají v LuaTEXu velmi pohodlně udělat cykly pracující jen na úrovni expand procesoru, což jsem s radostí využil pro cykly `\fornum` a `\foreach`, které nabízím programátorům maker v kolekci nástrojů rozšířeného API (nad standardními nástroji z plainTEXu).

Velmi praktický je v LuaTEXu například primitivní příkaz `\csstring`, který konečně odstraňuje veškeré tanečky s překážejícím a proměnlivým `\escapechar` při použití klasického `\string`. A takových dobrých drobností by se dalo najít více.

OpTEX tedy využívá na mnoha místech specifické vlastnosti LuaTEXu a není proto snadno přeportovatelný pro jiný TEXový engine (například XeTEX). Ani to nikdy nebylo záměrem a, domnívám se, není k tomu žádný důvod.

12 Některé věci jsou udělány jinak

Makra plainTEXu a OPmac jsou do OpTEXu přepsána. Zpětná kompatibilita ale není z dobrých důvodů zcela dodržena. Odlišnosti od plainTEXu jsou přesně popsány v dokumentaci. Uvedu některé příklady.

Výchozí sada fontů je Latin Modern a ne Computer Modern. Když napíšu `\meaning\makro`, pak rovnou vidím výpis makra v sazbě bez obskurních znaků, které se ve fontech Computer Modern odchylují od ASCII. Odlišnost od ASCII a sedmibitovost dělá v dnešní době rodinu Computer Modern nepoužitelnou.

Knuth rozhodl dosti nešťastně dát počátek souřadnic pro rozmístění sazby do bodu (1in, 1in) od levého horního rohu papíru. To je velmi frustrující při výpočtech skutečných okrajů, zejména když v Evropě počítáme v milimetrech. OpTEX má tento počátek jednoduše v levém horním rohu papíru, takže třeba hodnota `\hoffset` přímo udává velikost levého okraje.

V makrech plainTEXu (a bohužel toto je pak převzato i v jiných formátech) se vyskytují interní řídicí sekvence ve tvaru `\p@`, `\f@@t` a další. To je značně nečitelné. Kdo mě zná, jistě ví, že tento způsob zápisu maker bytostně nesnáším. Takže v OpTEXu nic takového nenajdete.

Rozhodl jsem se uživatele přinutit psát přímo akcentované znaky a nikdy je neskládat pomocí `\"a`, `\v c` a podobných. Takže makra pro akcenty nejsou záměrně v OpTEXu definována. Uživatel může tyto jednoznakové řídicí sekvence s výhodou použít k jakémukoli jinému účelu. Pokud by se chtěl konzervativní uživatel vrátit k nostalgickému značkování akcentů, může použít deklaraci `\oldaccents`, ale nedoporučuji to. Naopak, uživatel může deklarovat `\csquotes`, `\frquotes`, `\dequotes` nebo podobné a pak může psát `\"text"` nebo `\'text'` a má dvojité a jednoduché uvozovky dle pravidel zvoleného jazyka.

Makra OPmac jsou zcela nově přepsána a v mnohém doplněna o další vlastnosti. Značkování na uživatelské úrovni zůstalo stejné. Ale staré soubory doplňujících maker vycházejících z OPmac nejsou bohužel snadno přenositelné pro OpTEX, protože typicky se v těchto souborech autor maker opírá o nějaké vnitřní záležitosti OPmac, které vypadají v OpTEXu zcela jinak. Opustil jsem tak kompatibilitu obrovského množství svých vlastních makro souborů, ale už nyní vidím, že se to vyplatilo. Člověk tak nezůstane stát na jednom místě, ale posune se dál, k modernímu způsobu využití TEXu.

Reference

[1] http://petr.olsak.net/optex/

[2] OpTEX manual, http://petr.olsak.net/ftp/olsak/optex/optex-doc.pdf

[3] Olšák P.: *TEX in a nutshell*, 2020, 29 pp, http://petr.olsak.net/ftp/olsak/optex/tex-nutshell.pdf

[4] Olšák P.: *Typesetting Math with OpTEX* http://petr.olsak.net/ftp/olsak/optex/optex-math.pdf

[5] Olšák P.: *TEXbook naruby*, Konvoj 2001, http://petr.olsak.net/tbn.html

[6] Eijkhout V.: *TEX by topic*, 2007, http://mirrors.ctan.org/info/texbytopic/TeXbyTopic.pdf, `texdoc texbytopic`

[7] Olšák P.: *OpTEX – A new generation of Plain TEX*, in TUGboat 41:3, 2020, pages 348–354, http://petr.olsak.net/ftp/olsak/bulletin/tb128olsak-optex.pdf

[8] OpTEX font catalog, http://petr.olsak.net/ftp/olsak/optex/op-catalog.pdf

[9] OpTEX markup language standard (OMLS) http://petr.olsak.net/ftp/olsak/optex/omls.pdf

[10] OpTEX tricks, http://petr.olsak.net/optex/optex-tricks.html

[11] Documentation of `emoji` package, https://www.ctan.org/pkg/emoji, `texdoc emoji`

The article describes OpTEX [1] – a LuaTEX format based on plain TEX and OPmac. This macro package was introduced in TugBoat [7] and now, it is presented for Czech and Slovak users. The presentation is slighlty different than in the cited article [7]. For example, the comparison with LATEX and ConTEXt is mentioned here including benchmarks in tables 8.1 and 8.2.